ARTICLE IN PRESS

Information and Software Technology xxx (2014) xxx–xxx

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

# Identifying refactoring opportunities in object-oriented code: A systematic literature review

Jehad Al Dallal

*Department of Information Science, Kuwait University, P.O. Box 5969, Safat 13060, Kuwait*

A R T I C L E  I N F O

*Article history:*
Received 8 April 2014
Received in revised form 13 July 2014
Accepted 6 August 2014
Available online xxxx

*Keywords:*
Refactoring activity
Refactoring opportunity
Systematic literature review

A B S T R A C T

*Context:* Identifying refactoring opportunities in object-oriented code is an important stage that precedes the actual refactoring process. Several techniques have been proposed in the literature to identify opportunities for various refactoring activities.
*Objective:* This paper provides a systematic literature review of existing studies identifying opportunities for code refactoring activities.
*Method:* We performed an automatic search of the relevant digital libraries for potentially relevant studies published through the end of 2013, performed pilot and author-based searches, and selected 47 primary studies (PSs) based on inclusion and exclusion criteria. The PSs were analyzed based on a number of criteria, including the refactoring activities, the approaches to refactoring opportunity identification, the empirical evaluation approaches, and the data sets used.
*Results:* The results indicate that research in the area of identifying refactoring opportunities is highly active. Most of the studies have been performed by academic researchers using nonindustrial data sets. Extract Class and Move Method were found to be the most frequently considered refactoring activities. The results show that researchers use six primary existing approaches to identify refactoring opportunities and six approaches to empirically evaluate the identification techniques. Most of the systems used in the evaluation process were open-source, which helps to make the studies repeatable. However, a relatively high percentage of the data sets used in the empirical evaluations were small, which limits the generality of the results.
*Conclusions:* It would be beneficial to perform further studies that consider more refactoring activities, involve researchers from industry, and use large-scale and industrial-based systems.

**Contents**

*E-mail address:* j.aldallal@ku.edu.kw

http://dx.doi.org/10.1016/j.infsof.2014.08.002
0950-5849/© 2014 Elsevier B.V. All rights reserved.

## 1. Introduction

Refactoring is the maintenance task of restructuring software source code to enhance its quality without affecting its external behavior [23]. Fowler et al. identified 22 code "bad smells" and proposed 72 refactoring activities to improve the code design and remove the bad smells. Identifying refactoring opportunities is the first stage of the refactoring process [64,36,46]. Manually inspecting and analyzing the source code of a system to identify refactoring opportunities is a time-consuming and costly process. Researchers in this area typically propose fully or semi-automated techniques to identify refactoring opportunities. These techniques may be applicable to code written in different programming languages and should be evaluated empirically.

Although identifying refactoring opportunities is a highly active area of research, there is a lack of systematic literature reviews in this area to keep researchers and practitioners up to date with the state of research in the area of identifying refactoring opportunities. The goal of this paper is to report a systematic literature review (SLR) that (1) identifies the state-of-the-art in identifying refactoring opportunities and (2) assesses and discusses the collected and reported findings. Our SLR is intended both to help researchers identify related topics that are not well explored and to guide practitioners to existing techniques and approaches for identifying refactoring opportunities.

To focus our SLR and reduce overlap with related SLRs (e.g., [61,63], we limited our SLR to studies that identify opportunities for refactoring object-oriented code to remove code bad smells. Consequently, we excluded studies of refactoring design artifacts, such as Unified Modeling Language (UML) models, and studies of refactoring non-object-oriented code. In addition, we did not consider studies on identifying opportunities for refactoring activities unrelated to code bad smells, such as activities required to refactor object-oriented code to service or aspect-oriented code. Finally, we limited the SLR to studies that report empirical evaluations for the proposed identification techniques because these studies provide evidence for the practicality, effectiveness, or usefulness of the proposed techniques. Studies on detecting the refactored pieces of code (e.g., [17,55,39,53] are out of the scope of this SLR.

We extracted data for 2338 potentially relevant articles published before the end of 2013 from seven scientific databases. After a careful screening of these articles, performing a pilot search, and contacting researchers in this area, we identified 47 relevant primary studies (PSs). We analyzed and classified these PSs based on different perspectives, including the refactoring activities considered, the approaches followed for refactoring opportunity identification and for empirical evaluation, and the data sets used. In addition, we assessed the quality of the PSs from several perspectives, including the study design, conduct, analysis, and conclusions. We identified several topics in need of further research and challenges to be addressed in future related research.

**Table 1**
Stages of the paper selection process.

| Stages | Number of distinct articles included (+) or removed (−) | Total number of distinct articles |
|---|---|---|
| **Stage 1**: Identifying potentially relevant articles | +1092 | 1092 |
| **Stage 2**: Excluding articles based on titles and abstracts | −965 | 127 |
| **Stage 3**: Excluding articles based on full text | −88 | 39 |
| **Stage 4**: Including pilot search-based articles | +2 | 41 |
| **Stage 5**: Including articles after contacting PS authors | +6 | 47 |

**Table 2**
Number of articles found in the selected digital libraries.

| Stage | ACM | Springer | Scopus | IEEE | ISI | ScienceDirect | Compendex and Inspec | Total | Distinct |
|---|---|---|---|---|---|---|---|---|---|
| Stage 1 | 542 | 413 | 384 | 281 | 256 | 56 | 406 | 2338 | 1092 |
| Stage 2 | 98 | 9 | 75 | 42 | 58 | 11 | 83 | 375 | 126 |
| Stage 3 | 27 | 2 | 32 | 11 | 25 | 7 | 34 | 137 | 38 |

This paper is organized as follows. Section 2 reviews the related work. Section 3 describes our method of systematic review. Section 4 reports the results of the systematic review and answers the research questions. The results are discussed in Section 5. Finally, Section 6 presents our conclusions and outlines possible related open research topics and areas.

## 2. Related work

Researchers have performed various surveys and literature reviews related to software refactoring. Mens and Tourwé [36] performed an extensive survey of research in the area of software refactoring. They discussed and compared the existing research in terms of the refactoring activities considered and their supporting techniques, the refactored software artifacts, tool support, and the impact of refactoring on the software process. Identifying refactoring opportunities is only one of the refactoring activities considered. Therefore, in contrast to this SLR, which is more narrowly focused, the previous survey does not provide a detailed overview and discussion of the studies concentrating on identifying refactoring opportunities. Because of its broader topic, and because it was performed a decade ago, the previous survey considered only a few studies on identifying refactoring opportunities. In addition, the study did not follow a systematic approach and is more a literature survey than an SLR, as defined in this study.

Zhang et al. [63] performed an SLR of 39 studies published on code bad smells. They overviewed the code smells, the goals of studies on code bad smells, the approaches followed to investigate code bad smells, and the evidence provided to support the claim that code bad smells indicate problems in the code. Identifying code bad smells and identifying refactoring opportunities are related but different problems. Therefore, the research considered in this current SLR is different from the work performed by Zhang et al. Although studies that consider both code smells and refactoring opportunities are considered in both SLRs, we discuss these studies from a different perspective. Studies identifying code smells without relating them to refactoring activities are excluded from this SLR.

Wangberg's literature review examined 46 papers (2010) that considered both code smells and refactoring. Unlike this SLR, which only considers papers that provide empirical evidence, Wangberg's review considered not only empirical studies (11 papers) but also design research, contribution summaries, and theoretical contributions (35 papers). Wangberg's literature review is broader than this SLR, considering studies not only on identifying refactoring opportunities but also on other refactoring activities, such as performing refactoring and analyzing its impact on code quality. In addition, Wangberg's review includes refactoring opportunity identification using design artifacts and non-object-oriented systems. None of the considered studies provide empirical evidence for a proposed refactoring opportunity identification technique. This SLR is limited to studies that report empirical evidence for techniques proposed to identify refactoring opportunities from object-oriented code artifacts.

Misbhauddin and Alshayeb [38] conducted an SLR of existing approaches to refactoring UML models. The review discussed and analyzed 94 PSs based on several criteria, including the UML models, the formalisms used, and the impact of refactoring on model quality. The SLR reported in this paper is limited to code-based refactoring approaches and therefore does not overlap with the Misbhauddin and Alshayeb review, which is limited to design-based refactoring approaches.

## 3. Research method

This SLR summarizes the current state-of-the-art in the field of identifying refactoring opportunities in object-oriented (OO) software. Based on the Kitchenham and Charters [30] guidelines, this SLR was performed in three main phases: planning, conducting, and reporting the review. The first stage includes the recognition of the need for a systematic review and the development of the review protocol. For this purpose, we have identified the objectives of this SLR in Section 1 and discussed the key differences between this systematic review and existing related reviews in Section 2. This section reports the protocol followed to perform the review and reduce the chances of researcher bias. The protocol includes identifying the research questions, defining the search strategy, determining the study selection criteria, identifying the PS classification dimensions and attributes, and performing the corresponding data extraction and analysis.

### *3.1. Research questions*

Based on the PSs determined by the study selection process, this SLR seeks to answer the following research questions:

RQ1. What are the refactoring activities considered in the PSs?
RQ2. What are the approaches followed by the PSs to *identify* the refactoring opportunities?
RQ3. What are the approaches followed by the PSs to empirically *evaluate* the proposed or existing identification techniques for refactoring opportunities?
RQ4. What data sets were used to evaluate the identification techniques proposed in the PSs?

The research questions are based on the PICOC criteria [30]:

*Population:* object-oriented software, classes, methods, attributes (related to RQ1 and RQ2).
*Intervention:* techniques for identifying refactoring opportunities (related to RQ1 and RQ2) and approaches for evaluating the identification techniques (related to RQ3).
*Comparison:* not applicable
*Outcomes:* accuracy of the techniques in identifying refactoring opportunities (related to RQ3).
*Context:* context of the empirical evaluation in terms of data set size and venue (i.e., academia or industry) (related to RQ4).

This SLR seeks to identify techniques that have been empirically evaluated and applied to identify refactoring opportunities related to bad smells in object-oriented code. To achieve this goal, we searched the literature for the corresponding PSs.

### 3.2. Search strategy

To obtain a comprehensive list of PSs, we performed an automatic search in the most popular and relevant digital libraries. The search and PS selection process was performed in five stages, as listed in Table 1; the list of selected libraries is given in Table 2. Selecting all these libraries together can lead to overlapping results, which requires the identification and removal of redundant results; however, this selection of libraries increases confidence in the completeness of the review. To increase the comprehensiveness of the review, our search considered all years through the end of 2013 (we conducted the search on September 23, 2013 and modified the results on January 1, 2014).

We constructed the search string based on the following factors:

1. The major terms extracted from the research questions.
2. Alternative spellings and synonyms of the major terms.
3. Research terms used in relevant papers.
4. The use of Boolean AND to connect the major research terms and Boolean OR to connect alternative spellings and synonyms of the major terms.

The resulting general search string is as follows: (object-oriented OR object oriented OR class OR classes OR method OR methods OR attribute OR attributes) AND refactor* AND (estimat* OR predict* OR determin* OR identif* OR indicat* OR detect* OR track*).

The corresponding search string used for each the seven libraries is given in Appendix A. We followed the conventional practice of performing the automatic search within the titles and abstracts of the articles included in the digital libraries. To ensure that duplicates were ignored, we collected the search results in an Excel spreadsheet. For each article, we recorded the title, authors, journal/conference name, abstract, and year of publication of the article and the list of libraries in which the article was found. For each article found in each library, the spreadsheet was checked for the presence of the article. If the article was already included in the spreadsheet, the library name was added to the list of the libraries in which the article was found. Otherwise, a new record for the article was added to the spreadsheet. This process resulted in data for 2338 articles, of which 1092 were distinct; these articles were considered potentially relevant. The distribution of these articles among the digital libraries is given in Table 2.

To increase confidence in the completeness of the article list, we performed a pilot search using relevant literature reviews and the reference list of a recently published article [1] whose "related work" section gives an overview of key research papers in the field. This pilot search identified 17 relevant papers, of which only two (S16 and S47) were not in the list obtained using the digital libraries. The reasons the search did not find these articles are (1) article S47 was not present in any of the libraries and (2) article S16 did not include the "refactor*" key term in either its title or its abstract.

**Table 3**
Mapping between the identifiers and references of the PSs.

| | |
|---|---|
| S1: | Al Dallal [1] |
| S2: | Al Dallal and Briand [2] |
| S3: | Alkhalid et al. [3] |
| S4: | Alkhalid et al. [4] |
| S5: | Alkhalid et al. [5] |
| S6: | Bavota et al. [6] |
| S7: | Bavota et al. [7] |
| S8: | Bavota et al. [8] |
| S9: | Bavota et al. [9] |
| S10: | Bavota et al. [10] |
| S11: | Bavota et al. [11] |
| S12: | Cassell et al. [13] |
| S13: | Czibula and Czibula [15] |
| S14: | Czibula and Serban [16] |
| S15: | Du Bois et al. [18] |
| S16: | Fokaefs et al. [19] |
| S17: | Fokaefs et al. [20] |
| S18: | Fokaefs et al. [21] |
| S19: | Fokaefs et al. [22] |
| S20: | Higo et al. [24] |
| S21: | Higo et al. [25] |
| S22: | Hotta et al. [26] |
| S23: | Kanemitsu et al. [27] |
| S24: | Kataoka et al. [28] |
| S25: | Kimura et al. [29] |
| S26: | Lee et al. [32] |
| S27: | Liu et al. [33] |
| S28: | Mahouachi et al. [34] |
| S29: | Melton and Tempero [35] |
| S30: | Ments et al. [37] |
| S31: | Oliveto et al. [41] |
| S32: | Pan et al. [42] |
| S33: | Pan et al. [43] |
| S34: | Pan et al. [44] |
| S35: | Pan et al. [45] |
| S36: | Rao and Reddy [48] |
| S37: | Sales et al. [49] |
| S38: | Sales et al. [50] |
| S39: | Serban and Czibula [51] |
| S40: | Tairas and Gray [54] |
| S41: | Tourwé and Mens [56] |
| S42: | Tsantalis and Chatzigeogiou [57] |
| S43: | Tsantalis and Chatzigeogiou [58] |
| S44: | Tsantalis and Chatzigeogiou [59] |
| S45: | Tsantalis and Chatzigeogiou [60] |
| S46: | Yang et al. [62] |
| S47: | Zhao and Hayes [64] |

### 3.3. Study selection

To select PSs from the potentially relevant studies, we applied the following inclusion and exclusion criteria.

Inclusion criteria:

Articles published in peer-reviewed journals or conference proceedings found in the selected digital libraries on January 1, 2014 and including the following:

1. A technique or a method to identify opportunities for refactoring activities to eliminate code bad smells in object-oriented software.
2. An empirical evaluation for the technique/method using a large- or small-scale data set.

Both criteria had to be satisfied to ensure that the selected PS was within the targeted area of research and that the study was empirically evaluated. Papers not fulfilling all inclusion criteria were excluded, as follows:

1. Studies that do not explore the identification of refactoring opportunities to eliminate code bad smells but instead address other refactoring aspects, such as performing refactoring activities or estimating refactoring costs.
2. Research without an empirical evaluation of the refactoring identification technique(s).
3. Studies that consider refactoring code other than object-oriented software.
4. Studies that consider activities other than code refactoring, including the refactoring of design artifacts such as UML models.
5. Studies that consider the identification of refactoring opportunities that are not related to code bad smells (e.g., refactoring opportunities of object-oriented code to service or aspect-oriented code).

We found some duplicate articles published in different venues, having the same authors and reporting the same identification technique with empirical evaluation using the same data sets. In these cases, we considered only the superset version. Articles that report the same identification technique but different empirical studies are included in the study.

The author (Researcher 1) and his research assistant (Researcher 2) each individually screened each title and abstract in the Excel spreadsheet to determine whether an article should be included or excluded. From the relevant papers identified in the pilot search, we noted that some papers report empirical studies within the articles' text without indicating this fact in the abstract. Therefore, we decided to screen the collected articles in two stages (Stage 2 and Stage 3 reported in Tables 1 and 2). In Stage 2, we screened the titles and abstracts of the papers to determine only whether the first inclusion criterion was applicable. In Stage 3, we screened the full text of the selected papers, considering all inclusion and exclusion criteria. In Stage 2, both researchers agreed to include 71 articles and to exclude 965 articles. They disagreed regarding the remaining 56 articles. To determine the degree of agreement, we applied Cohen's Kappa coefficient [14]. Based on the Landis and Koch [31] categories, the agreement between the two researchers in the first phase was found to be substantial (0.69). To ensure the comprehensiveness of our study, we considered each paper found by either of the two researchers as a potential PS for further inspection; this procedure resulted in considering 127 articles (i.e., 71 + 56) as an input for the second screening phase. When the full texts of the 127 articles were screened in Stage 3, both researchers agreed to include 37 articles and to exclude 75 articles, and they disagreed regarding the remaining 15 articles. The corresponding degree of agreement is 0.74 (substantial). The two researchers discussed the 15 cases of disagreement and reached a consensus to include two articles and exclude the rest. In Stage 4, we added the two papers found in the pilot search (as detailed in Section 3.2). In Stage 5, we contacted the corresponding authors (27 distinct authors) of the 41 articles (i.e., 37 + 2 + 2) by email to explore whether they were aware of relevant papers not included in our list. The authors' replies resulted in adding 6 articles, so that the final list of PSs included 47 articles (Table 3). Two of the six suggested papers were not detected in the earlier stages because they do not satisfy the search string, and the other four papers had not yet been added to the digital libraries at the time of the search (January 1, 2014) because they were either published on-line, but not yet in a regular issue, or published at the end of 2013.

## 3.4. Study classification

Based on the research questions identified in Section 3.1, we categorized the PSs by four main dimensions: (1) refactoring activities (related to RQ1), (2) refactoring opportunity-identification approaches (related to RQ2), (3) evaluation approaches (related to RQ3), and (4) data sets used in the evaluation process (related to RQ4). The author of this SLR screened all the PSs to identify the attributes of each dimension. The attributes of the first dimension are first set to be the refactoring activities proposed by Fowler [23]. However, after screening all the PSs, we added two more activities that were not identified by Fowler [23]. This classification process resulted in identifying 22 attributes for the refactoring activities classification dimension, each attribute representing one of the refactoring activities considered by at least one of the PSs. The considered attributes (i.e., refactoring activities) are listed in Section 4.2.

The identification of attributes for the second and third dimensions was performed incrementally. That is, for the second dimension, we started with an empty set of attributes. The author of this SLR screened and analyzed each PS for the approach followed by the study to identify refactoring opportunities. Based on the inputs and the identification process, a classification attribute was identified. If the attribute had not already been identified when a previously screened PS was considered, the attribute was added to the set of classification attributes. The same approach was followed to identify the attributes for the third classification dimension. This classification process resulted in identifying six attributes for each of the second and third dimensions, as listed in Sections 4.3 and 4.4.

We subdivided the fourth classification dimension into three subdimensions, including the data set programming language, venue, and size. The attributes of the programming language subdimension were determined by screening the PSs for the data set programming languages. This process resulted in identifying three attributes, each representing a programming language (i.e., Java, C++, and SmallTalk). To identify the attributes of the data set venue subdimension, we started with the categorization of Sjoeberg et al. [52], who identified five corresponding attributes including open source, self-constructed project, academic project, commercial project, and student project. The author of this SLR screened the PSs to determine the suitability of this categorization and found that four attributes (open source, academic project, commercial project, and student project) were enough to classify the PSs based on the data set venue subdimension. Finally, we followed the guidelines suggested by Radjenović et al. [47] to categorize the studies by the sizes of the data sets in terms of the number of classes and KLOC (thousands of lines of code, LOC). Small studies consider systems of fewer than 50 KLOC or 200 classes, medium studies consider systems ranging from 50 KLOC or 200 classes to 250 KLOC or 1000 classes, and large studies consider systems with more than 250 KLOC or 1000 classes. If a study considered more than one system, the sum of the sizes of the systems was categorized. If both the LOC and the number of classes were reported, the study was classified according to the larger category. For example, if a study considered a system of 60 KLOC and 150 classes, it would be classified as medium. As a result, three attributes (small, medium, and large) were considered to classify the PSs based on the data set size subdimension.

## 3.5. Study quality assessment

We followed the guidelines suggested by Kitchenham and Charters [30] to construct the quality checklist given in Table 4. The checklist considers several quality aspects, including the study

**Table 4**
PS quality assessment questions and results.

| ID | Question | Percentage of PSs | | | |
|---|---|---|---|---|---|
| | | Yes (%) | Partially (%) | No (%) | N.A. |
| **Design** | | | | | |
| QA1 | Are the applied identification techniques for refactoring opportunities clearly described? | 93.6 | 6.4 | 0 | 0 |
| QA2 | Are the refactoring activities considered clearly stated and defined? | 57.4 | 36.2 | 4.3 | 2.1 |
| QA3 | Are the aims of the study clearly stated? | 97.9 | 2.1 | 0 | 0 |
| QA4 | Was the sample size justified? | 31.9 | 23.4 | 44.7 | 0 |
| QA5 | Are the evaluation measures fully defined? | 31.9 | 31.9 | 36.2 | 0 |
| **Conduct** | | | | | |
| QA6 | Are the data collection methods adequately described? | 61.7 | 31.9 | 6.4 | 0 |
| **Analysis** | | | | | |
| QA7 | Are the results of applying the identification techniques evaluated? | 85.1 | 2.1 | 12.8 | 0 |
| QA8 | Are the data sets adequately described? (size, programming languages, source) | 78.7 | 19.2 | 2.1 | 0 |
| QA9 | Are the study participants or observational units adequately described? | 29.8 | 55.3 | 14.9 | 0 |
| QA10 | Are the statistical methods described? | 61.7 | 4.3 | 4.3 | 29.8 |
| QA11 | Are the statistical methods justified? | 48.9 | 17 | 4.3 | 29.8 |
| QA12 | Is the purpose of the analysis clear? | 78.7 | 14.9 | 6.4 | 0 |
| QA13 | Are the scoring systems (performance evaluation) described? | 12.8 | 6.4 | 80.8 | 0 |
| **Conclusion** | | | | | |
| QA14 | Are all study questions answered? | 85.1 | 14.9 | 0 | 0 |
| QA15 | Are negative findings presented? | 42.5 | 12.8 | 27.7 | 17 |
| QA16 | Are the results compared with previous reports? | 29.8 | 0 | 70.2 | 0 |
| QA17 | Do the results add to the literature? | 76.6 | 21.3 | 2.1 | 0 |
| QA18 | Are validity threats discussed? | 31.9 | 23.4 | 44.7 | 0 |

design, conduct, analysis, and conclusion, and was applied to assess the corresponding quality of each of the PSs considered. Each question is evaluated as "Yes", "Partially", or "No" with a corresponding score of 1, 0.5, or 0, respectively. Some of the questions were not applicable to some PSs; these PSs were not evaluated for those questions. The PSs were assessed individually by the author and his research assistant, and the results were compared. Disagreements were discussed until agreement was reached. For each question in Table 4, we reported the percentage of PSs with each of the three possible answers. In addition, for each PS, we added the scores for each question and found the percentage over the applicable questions for that PS. The results reported in Table 4 are discussed in Section 4.6. The details of the study quality assessment results are provided in an online appendix.[1]

## 3.6. Data extraction and analysis

The research assistant extracted and reported the data corresponding to the research questions and study quality assessment. For each PS, the extracted data included the full reference (i.e., title, authors, journal/conference name, year, volume, and page numbers), abstract, refactoring activities considered, approach followed to identify refactoring opportunities, context of the study, data set(s) (i.e., name, version, programming language, size, and availability of each data set considered), and validation method. In addition, the paper text related to each quality assessment question was highlighted, and the question was answered. The author checked all this work. Disagreements were discussed until a consensus was reached. The process of having one data extractor and one data checker had been previously found useful in practice [12].

### 3.6.1. Refactoring activities

The refactoring activities considered were categorized and analyzed following the proposal of Fowler [23]. The extracted data were used to address RQ1. Collecting these data allowed the identification of the most and least studied refactoring activities. This identification might suggest that future research should concentrate more on those refactoring activities that have been less well studied. In addition, categorizing the studies by the refactoring activities makes it possible to compare the results of the identification techniques for the same refactoring activities.

### 3.6.2. Data set details

We considered several factors related to the data sets used in the empirical studies reported, which allowed us to address RQ4. The factors include the name and size of the data sets and the programming languages in which the data sets were developed. Categorizing the studies by the names of the systems helps in comparing results related to the same refactoring activities. As illustrated in Section 3.4, we followed the guidelines suggested by Radjenović et al. [47] to categorize the studies by the sizes of the data sets in terms of the number of classes and KLOC (thousands of lines of code, LOC) into small, medium, and large studies. The size of the data sets gives the reader an idea of the external validity of a study, where studies of larger data set size potentially have higher external validity [47].

### 3.6.3. Study context

The study context included two factors, the researcher and the data set. The authors of the PSs worked in either academia or industry. Based on the authors' affiliations, we categorized the articles as authored by researchers working in academia, industry, or both. Extracting these data allows identification of the fraction of people working in industry in research related to identifying refactoring opportunities. In addition, it allows identification of the context in which the techniques were evaluated.

The second context factor relates to the data sets considered in the empirical validation and allows RQ4 to be addressed. Based on the categorization of Sjoeberg et al. [52], the data set can be open source or the result of a self-constructed project, academic project, commercial project, or student project. Studies based on open-source data sets are deemed to be repeatable.

## 3.7. Validity threats

This SLR has several limitations that may restrict the generality and limit the interpretation of our results. The first is the

[1] http://www.cls.kuniv.edu/drjehad/research.htm

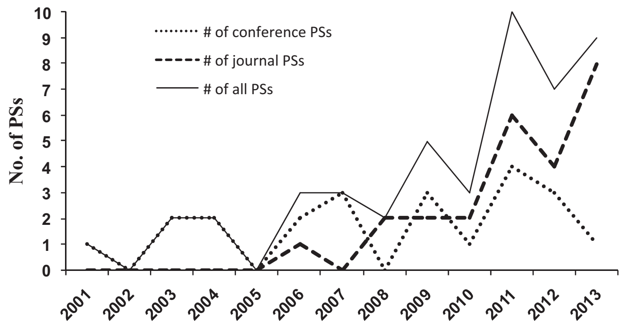

**Fig. 1.** Distribution of PSs over the years.

possibility of paper selection bias. To ensure that the studies were selected in an unbiased manner, we followed the standard research protocol and guidelines reported by Kitchenham and Charters [30]. We constructed the search string based on the research questions and related search terms. To decrease the possibility of missing a relevant study, we employed several strategies. The first strategy was to perform the search using the most commonly relevant digital libraries that include both relevant journals and conferences. The second strategy was to construct a wide search string, which resulted in scanning a relatively large number of papers, including gray literature. The third and fourth strategies were to perform pilot searching and to contact the corresponding authors of the identified PSs to inquire about any missing relevant studies. The number of additional studies identified by the pilot search (only two studies) indicated that the libraries considered were comprehensive enough and the search string was strong enough to detect most of the relevant studies. Contacting the corresponding authors led to adding six studies. However, only two of these studies were not included initially because they lacked some of the search string terms. The other four were either published on-line, but not yet in a regular publication, or were published at the end of 2013 and had not yet been added to the libraries as of the original search. The fifth strategy we used to decrease the possibility of missing relevant studies was a multi-stage selection process. After applying the search string, the titles and abstracts of the identified articles were screened for whether they indicated any relationship to the problem of identifying refactoring opportunities. At this stage, we did not consider whether the study reported an empirical evaluation because some studies report empirical studies without indicating this fact in their title or abstract. Checking whether the study reports an empirical evaluation was delegated to the next stage, in which the full texts of the studies were screened.

To increase confidence that no study was incorrectly excluded, two researchers independently screened the titles and abstracts of the studies. Studies were only removed when both researchers agreed on the exclusion, and the full texts of the remaining studies were screened by both researchers independently. The degree of agreement was substantial at both stages, which increases the reliability of the study selection results. Finally, to further ensure unbiased study selection, the final decision on the articles with selection disagreements was performed based on consensus meetings.

Assessing the quality of a study is subjective. To limit this validity threat, instead of proposing nonstandard quality factors, we adapted the well-defined study quality factors suggested by Kitchenham and Charters [30]. In addition, we defined three levels for each assessment question. The PSs were assessed by one researcher and checked by the other, a technique applied in similar studies (e.g., [47]. A consensus was reached to resolve any disagreement.

## 4. Results

The findings of the SLR are reported in this section. First, we provide an overview of the PSs included in the review; then, we present the answers to the research questions and the results of the research quality assessment.

### 4.1. Overview of the PSs

As discussed in Section 3, the search and selection processes resulted in the 47 PSs listed in Table 3. The details of the extracted data are provided in Appendix B and in an online appendix.[2] Fig. 1 shows the distribution of the conference, journal, and total PSs over the years. The figure shows that the first PS we considered appeared in 2001, approximately two years after the publication of Fowler's book. From 2001 to 2010, approximately two PSs were published per year on average. The problem of identifying refactoring opportunities began to attract more research attention after 2011; 55.3% of the PSs were published after 2011, with an average of more than eight per year from 2011 to 2013. This observation indicates that this area of research is currently highly active and is considered an important software problem.

The PSs were published in 33 different sources. Over half of the PSs (53.2%) were published in journals; the rest were published in conference proceedings. The slight difference between these percentages shows the importance of considering both journals and conferences in this literature review. That is, relying only on journals or only on conference proceedings results in an incomplete literature review. In addition, the slight difference in the percentages indicates that researchers in this research area have not had a clear overall preference for journals over conference proceedings, or vice versa, when publishing their results. However, we note that the first journal PS was published in 2006, after the first five conference PSs were published. However, in the three years since 2011, only 8 (36.4%) conference papers were published, whereas 18 (72%) of the journal PSs were published during the same years. This observation shows that researchers in this area have shifted their publishing interests in recent years from conferences to journals. This indication is positive, as journal papers, especially ones published in more prestigious journals, are typically more complete and report more extensive studies.

Table 5 lists publication sources with two or more publications. The list includes five journals and three conferences, which together published only 46.8% of the PSs. This result indicates that limiting the search to certain related journals and conferences, rather than performing an automatic search in the main science libraries, would potentially cause the literature review to miss approximately half of the relevant PSs. The results reported in Table 5 show that *Journal of Software and Systems* attracted more researchers in the area of identifying refactoring opportunities than other journals in the field. In addition, the results show that researchers presented findings in this area at the *IEEE International Conference on Software Maintenance* more than at other conferences.

Except for S20, the authors of all PSs are from academia. PS S20 was performed by researchers from both academia and industry. This observation indicates that that there is potentially a large gap between academic research in this area and industry. Therefore, based on the selected PSs, no strong evidence was found regarding the importance of this research in industry or on whether this area is of interest for industrial people. However, it is unfair to conclude based only on the selected PSs that the software industry is not interested in identifying refactoring opportunities. Such a conclusion requires more investigation using other

[2] http://www.cls.kuniv.edu/drjehad/research.htm

methods, such as surveys performed in industrial venues. Based on the PSs we considered, we could draw no conclusions regarding the refactoring activities that are of interest to industry practitioners. Academic researchers in this area of research are advised to invite experts from the industrial domain to participate and share their experiences.

### 4.2. RQ1: What are the refactoring activities considered in the PSs?

The PSs considered 22 refactoring activities; 20 of these activities are among the 72 activities identified by Fowler [23], and two have been proposed by others (i.e., *Eliminate Return Value* (in S24) and *Move Class* (in S5, S7, S9, S32, and S33). *Move Class* refactoring refers to moving a class from one package to a more appropriate package. Fig. 2 shows the distribution of the ten refactoring activities that were considered by at least two PSs. The remaining activities that were each mentioned by a single PS are as follows: *Pull Up Constructor, Parameterize Method, Extract Subclass, Replace Method with Method Object, Replace Data Value with Object, Separate Query from Modifier, Encapsulate Downcast, Replace Temp with Query, Replace Type Code with State/Strategy, Eliminate Return Value, Replace Conditional with polymorphism,* and *Extract Interface*. PS S27 claims that their technique is applicable for *generalization* refactoring activities without specifying specific ones. In addition, PS S28 states that their identification technique can be applied to predict the opportunities for any refactoring activity. These two PSs are not included in Fig. 2 because they do not specify the refactoring activities considered.

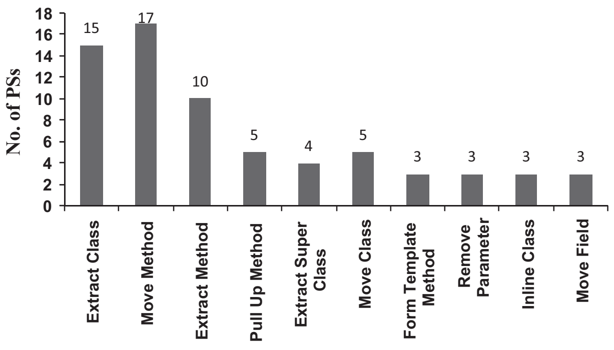

**Fig. 2.** Distribution of the refactoring activities.

Fig. 2 shows that identifying the opportunities for each of the *Move Method*, *Extract Class*, and *Extract Method* refactoring activities was considered by more than 20% of the PSs. Identifying the *Pull Up Method*, *Extract Superclass*, *Move Class*, and *Form Template Method* refactoring opportunities received less attention. The remaining refactoring activities were either not considered or were considered by three or fewer PSs.

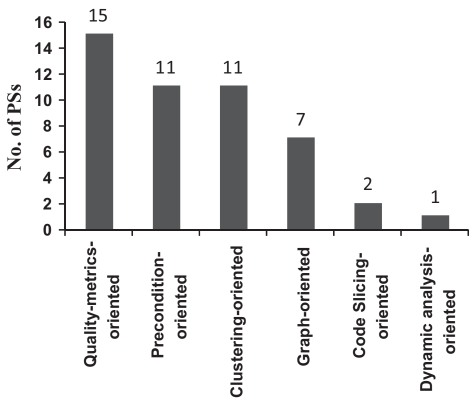

**Fig. 3.** Distribution of the PSs over the approaches to identifying refactoring opportunities.

**Table 5**
Key publication sources.

| Source | No. of PSs | Percentage of PSs (%) |
|---|---|---|
| IEEE International Conference on Software Maintenance (ICSM) | 5 | 10.6 |
| Journal of Systems and Software(JSS) | 4 | 8.5 |
| European Conference on Software Maintenance and Reengineering (CSMR) | 3 | 6.4 |
| ACM Transactions on Software Engineering and Methodologies (TOSEM) | 2 | 4.3 |
| IEEE Transactions on Software Engineering (TSE) | 2 | 4.3 |
| Empirical Software Engineering (EMSE) | 2 | 4.3 |
| Information and Software Technology (IST) | 2 | 4.3 |
| International Conference on Software Engineering (ICSE) | 2 | 4.3 |
| Total | 22 | 46.8 |

### 4.3. RQ2: What are the approaches followed by the PSs to identify the refactoring opportunities?

Based on the general approaches followed by the PSs, we classified the PSs into six main categories. The distribution of the PSs over these approaches is shown in Fig. 3, and the approaches are detailed below.

#### 4.3.1. Quality metrics-oriented approach

Fifteen (31.9%) of the PSs (S1, S2, S6, S7, S9, S10, S11, S19, S20, S21, S28, S31, S37, S46, and S47) propose techniques that depend on quality metrics to identify or predict refactoring opportunities. In S1, a statistical model based on size, cohesion, and coupling metrics was constructed to predict *Extract Subclass* refactoring opportunities. In S2, Al Dallal and Briand proposed a cohesion metric and showed its usefulness in detecting *Move Method* and *Extract Class* refactoring opportunities. In S6, a metric that measures the structural and semantic similarities between methods in a class (a class cohesion aspect) was used to predict *Extract Class* refactoring opportunities. In S11, Bavota et al. proposed a metric that measures the structural and semantic similarity between one method and other methods in the same class (a class cohesion aspect) and between one method and other methods in other classes (a class coupling aspect); they explained the use of an approach based on game theory to find an optimal balance between cohesion and coupling and suggested a corresponding *Extract Class* refactoring opportunity. In S19 and S37, a metric that considers the distance (similarity) between a method in a class and members of other classes (a class coupling aspect) was used as a basis to suggest *Move Method* refactoring opportunities. In S31 and S10, the structural and conceptual relationships among methods in the same class (a class cohesion aspect) and between methods in the same class and methods in other classes (a class coupling aspect) were used to determine *Move Method* refactoring opportunities. Similarities based on structural and semantic relationships between the system's classes were used in S7 and S9 to identify *Move Class* refactoring opportunities. In S46, Yang proposed dividing the code of a method into fragments, based on a code statement classification schema, and then assessing the coupling between the fragments to suggest *Extract Method* refactoring opportunities. In S47, Zhao and Hayes proposed using a set of complexity metrics to predict classes in need of *Extract Class* refactoring. In S20 and S21, code clones were detected, and a set of quality metrics were used to suggest opportunities for several refactoring activities, including *Extract Class*, *Extract Superclass*, *Extract Method*, *Pull Up Method*, *Form Template Method*, *Move Method*, *Parameterize Method*, and *Pull Up Constructor*. Finally, in S28, Mahouachi et al. proposed a technique that considers quality metrics, refactoring operations, and a set of examples to build a model to map a detected bad smell to a certain refactoring activity. The technique was claimed to be applicable to predicting the opportunities for any of the refactoring activities.

### 4.3.2. Precondition-oriented approach

Different sets of preconditions were defined by eleven (23.4%) of the PSs (S15, S22, S24, S26, S27, S30, S38, S40, S41, S43, and S44) and applied to identify refactoring opportunities. Du Bois et al. (S15) and Kataoka et al. (S24) considered several refactoring activities. For each activity, they suggested inspecting the code for its compliance to a set of conditions. For example, a method was suggested to be moved if it did not use local resources, was rarely invoked, and was mostly referenced by another class. In S38, Seng et al. suggested a set of preconditions to apply *Move Method* refactoring and suggested using a set of quality metrics to choose the best refactoring solution. In S22, S26, S27, and S40, a set of conditions were applied to detected code clones to suggest which refactoring activity could be applied to improve the code. In S30, S41, and S44, a set of conditions were applied to detected bad smells to decide which refactoring activities were applicable. In S43, Tsantalis and Chatzigeorgiou proposed detecting the *Feature Envy* bad smell and checking the compliance of a set of preconditions to identify *Move Method* refactoring opportunities.

### 4.3.3. Clustering-oriented approach

Clustering techniques were applied in eleven (23.4%) PSs (S3, S4, S5, S12, S13, S14, S16, S17, S18, S36, and S39) to identify refactoring opportunities. In S3, a clustering algorithm based on a proposed similarity measure for the lines of code in a method was introduced to identify *Extract Method* refactoring opportunities. In S4, Alkhalid et al. suggested using the similarities between one method and other methods, both in the same class and in other classes, as a basis for clustering the methods and proposing *Move Method* refactoring opportunities. At a higher level, in S5, Alkhalid et al. suggested using the similarities between the classes of a system as a basis for clustering the classes into packages and proposing *Move Class* refactoring opportunities. In S13 and S39, clustering techniques that consider the distances between the methods and attributes within a class and in different classes were used to identify *Move Method*, *Move Field*, *Inline Class*, and *Extract Class* refactoring opportunities. A similar clustering technique was considered in S14 to identify *Move Method*, *Move Field*, and *Inline Class* refactoring opportunities. Fokaefs et al. (in S16, S17, and S18) and Rao and Reddy (in S36) proposed applying different clustering algorithms that account for the distances between the methods and attributes of a class to identify *Extract Class* refactoring opportunities. Finally, in S12, a clustering technique that considers the structural and semantic distances between the elements of a class was used to identify *Extract Class* refactoring opportunities.

### 4.3.4. Graph-oriented approach

The refactoring opportunity identification methods proposed in seven (14.9%) PSs (S8, S23, S29, S32, S33, S34, and S35) are graph theory-based techniques. In S29, a graph that represents the dependencies among classes of a system was proposed, and the reachability of each node (which represents a class) was analyzed to identify *Extract Interface* refactoring opportunities. In S34 and S35, a graph that represents the relations between methods and attributes was proposed. In S35, an evolutionary algorithm was applied, and in S34, a community detection algorithm was used to analyze the graph and construct an optimized class structure. In both techniques, the original and optimized graphs were compared to identify *Move Method* refactoring opportunities. Pan et al. (in S32 and S33) suggested representing classes and their dependencies by a graph, applying community detection algorithms to obtain an optimized graph, and comparing the original and optimized graphs to identify *Move Class* refactoring opportunities. In S8, the relationships between the methods in a class were represented by a graph, and the graph was partitioned using an existing graph-based algorithm to suggest *Extract Class* refactoring opportunities. Finally, in S23, Kanemitsu et al. proposed measuring the distances between the nodes of a program dependency graph of a method to help identify *Extract Method* refactoring opportunities.

### 4.3.5. Code slicing-oriented approach

Tsantalis and Chatzigeorgiou (in S42 and S45) suggested slicing the program dependence graph of a method to identify *Extract Method* refactoring opportunities.

### 4.3.6. Dynamic analysis-oriented approach

In S25, Kimura et al. suggested analyzing method traces (invocations of methods during program execution), detecting corresponding relationships between methods, and identifying *Move Method* refactoring opportunities.

## 4.4. RQ3: What are the approaches followed by the PSs to empirically evaluate the proposed or existing identification techniques for refactoring opportunities?

We have identified six different approaches followed by the SPs to empirically evaluate the identification techniques. Each PS applied one or more of these approaches. The distribution of the PSs over the evaluation approaches is shown in Fig. 4, and the evaluation approaches are detailed below.

### 4.4.1. Intuition-based evaluation

We identified two levels of intuition-based evaluation. On the first level, the systems considered in the empirical study were shown to one or more experts to intuitively identify refactoring opportunities. The proposed identification technique was applied to the same systems, and the results were compared with the experts' intuitions. This level of intuition-based evaluation detects both the counterintuitive refactoring cases that are identified by the technique and the intuitive refactoring cases that are undiscovered by the technique. This type of evaluation was applied in seven PSs (S18, S19, S27, S44, S45, S46, and S47). On the second level of intuition-based evaluation, the proposed identification technique was applied to the systems that were involved in the empirical study. The refactoring cases identified by the technique were then provided to one or more experts to assess whether they match intuition. This level of intuition-based evaluation is weaker than the first because it detects only the counterintuitive refactoring

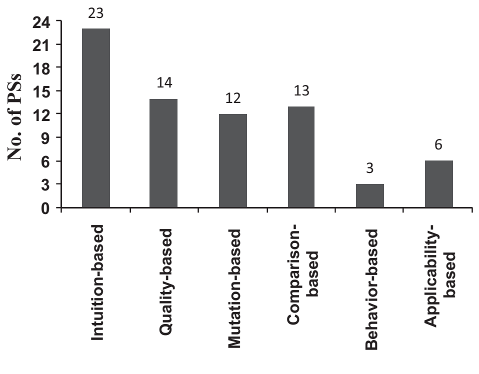

**Fig. 4.** Distribution of the PSs over the evaluation approaches.

cases that are identified by the technique and does not detect the cases that are undiscovered by the technique. The second level of intuition-based evaluation was applied in 18 PSs (S6, S7, S9, S10, S16, S17, S18, S24, S25, S30, S32, S33, S34, S35, S38, S42, S43, and S45). The total number of PSs that applied intuition-based evaluation (i.e., the union set of PSs that applied one or both of the intuition-based evaluation levels) is 23 (48.9%).

### 4.4.2. Quality-based evaluation

In 14 PSs (29.8%), including S4, S5, S9, S10, S12, S15, S29, S32, S33, S35, S36, S43, S45, and S46, the techniques were applied to one or more software systems to identify the refactoring candidates; the suggested refactoring activities were performed; and the quality of the code before and after applying the refactoring activities was assessed and compared. Cohesion and coupling are the two factors primarily considered in this evaluation. The improvement in the code quality was considered an indication for the correctness of the refactoring suggestions.

### 4.4.3. Mutation-based evaluation

Twelve PSs (25.5%) (S1, S2, S6, S7, S8, S11, S13, S14, S31, S37, S38, and S39) evaluated the proposed refactoring identification techniques by mutating the selected system so that the mutated systems needed refactoring. The identification technique was applied to the mutated system, and the technique was evaluated by checking whether the expected refactoring candidates were identified.

### 4.4.4. Comparison-based evaluation

Thirteen PSs (27.6%) considered comparison-based evaluation for their proposed techniques. In S13, S14, S22, S23, S27, S28, S32, S33, S39, and S40, the proposed techniques were applied to one or more systems, and the suggested refactoring candidates were compared to the ones obtained by applying other existing techniques. In addition, in S6, the proposed technique and an existing technique were applied to systems, and the code quality of the refactored systems was compared to explore which technique produced a greater improvement. In S11 and S37, the proposed technique and an existing technique were applied to systems; the results were evaluated; and the evaluation results were compared.

### 4.4.5. Behavior-based evaluation

In S22, S23, and S28, the proposed techniques were applied, and the suggested refactoring candidates were performed. The behavior of the system before and after applying the suggested refactoring activities was compared to ensure that the applied refactoring did not change the system's behavior.

### 4.4.6. Applicability-based evaluation

In S3, S20, S21, S26, S40, and S41, the proposed techniques were evaluated only by showing their applicability to one or more systems. However, the suggested refactoring candidates were not evaluated.

## 4.5. RQ4: What data sets were used to evaluate the identification techniques proposed in the PSs?

The PSs considered different data sets that had different sizes and programming languages, as described below.

### 4.5.1. The variety of the data sets

The total number of data sets used in the PSs was 138 (with an average of approximately three data sets per PS), including 79 distinct data sets. Fig. 5 shows the distribution of the number of

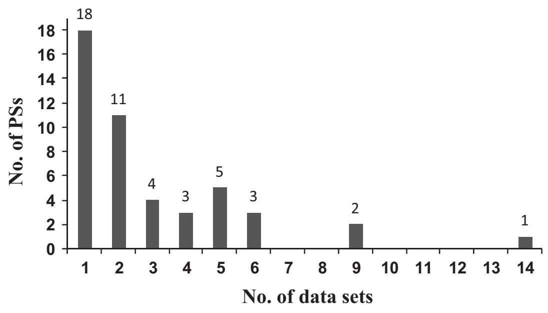

**Fig. 5.** Distribution of the data sets used in the PSs.

**Table 6**
Descriptions of the commonly used data sets.

| System | Programming language | Venue | No. of studies | Studies |
|---|---|---|---|---|
| JHotDraw | Java | Open-source | 16 | S1, S2, S6, S7, S9, S8, S11, S13, S14, S17, S18, S26, S34, S37, S38, S39 |
| Apache Ant | Java | Open-source | 6 | S20, S21, S22, S23, S37, S40 |
| ArgoUML | Java | Open-source | 6 | S6, S8, S11, S26, S31, S37 |
| GanttProject | Java | Open-source | 5 | S1, S6, S8, S9, S28 |
| JFreeChart | Java | Open-source | 5 | S10, S26, S40, S43, S45 |
| Jedit | Java | Open-source | 4 | S9, S10, S40, S43 |
| Trama | Java | Open-source | 3 | S5, S32, S33 |
| Front End | Java | Open-source | 3 | S5, S32, S33 |
| SelfPlanner | Java and C++ | Academic project | 3 | S33, S43, S45 |
| GESA | Not specified | Student project | 3 | S7, S9, S10 |
| SMOS | Not specified | Student project | 3 | S7, S9, S10 |
| CSGestionnaire | Java | Open-source | 2 | S3, S4 |
| Eclipse | Java | Open-source | 2 | S8, S11 |
| LAN-simulation | Java | Academic project | 2 | S19, S35 |
| Xerces | Java | Open-source | 2 | S6, S26 |
| Azureus | Java | Open-source | 2 | S26, S27 |
| eTour | Not specified | Student project | 2 | S7, S9 |
| SESA | Not specified | Student project | 2 | S7, S9 |
| eXVantage | Not specified | Industrial | 2 | S9, S10 |
| FreeCol | Not specified | Open-source | 2 | S12, S37 |
| Weka | Java | Open-source | 2 | S12, S37 |
| FreeMind | Java | Open-source | 2 | S1, S37 |
| Jruby | Java | Open-source | 2 | S37, S40 |
| Soul | SmallTalk | Academic project | 2 | S30, S41 |

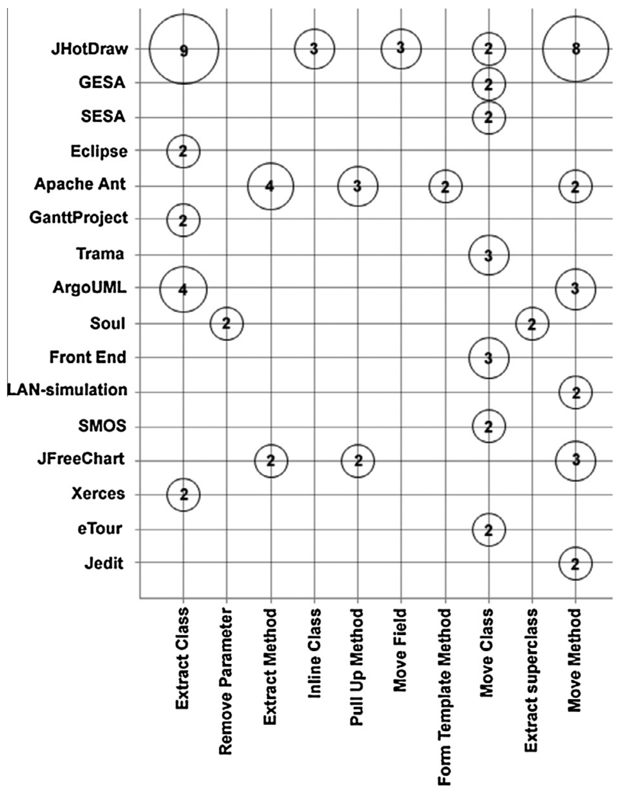

**Fig. 6.** Number of PSs using data sets by refactoring activities.

data sets considered in the PSs. The maximum number of data sets used in a PS was 14 (S37). Most of the PSs (61.7%) considered two or more data sets, and the remaining 18 PSs (38.3%) each considered a single data set.

Of the 79 data sets, 24 (30.4%) were used in more than one PS, and 55 were each used in a single PS. Table 6 provides details for the 24 multiply used data sets. The table shows that JHotDraw was used in 16 PSs (34%). The other data sets included in Table 6 were used in two to six PSs. Fig. 6 shows the mapping between the systems and the refactoring activities considered. The figure includes only the systems that were used at least in two PSs that consider the same refactoring activity, and therefore the figure depicts the number of PSs with comparable results. The figure shows that there are 27 instances (represented by the bubbles) of comparable studies. In addition, the figure shows that there are ten refactoring activities whose identification results can be compared across certain PSs because these PSs considered the same systems and the same refactoring activities. The largest matching (9 occurrences) was found for the identification of Extract Class refactoring opportunities in the JHotDraw system. Other comparisons can be performed across a number of PSs ranging from 2 to 8.

### 4.5.2. The sizes of the data sets

The PSs considered data sets of a variety of sizes. Based on the total number of classes and LOC in the systems considered by each PS, we found that 21 (44.7%) of the PSs either used a small data set or failed to report the size. Medium and large data sets were used by 11 (23.4%) and 15 (31.9%) of the PSs, respectively. At a finer grain level, we classified the individual data sets based on their sizes. Among the 138 systems, 58 (42%), 55 (39.9%), and 21 (15.2%) were small, medium, and large, respectively. The sizes of 4 systems (2.9%) were not specified.

### 4.5.3. The programming languages of the data sets

The most frequently used programming language for implementing the systems considered was Java (81.9% of the systems). Smalltalk and C++ were used in 2.2% and 1.4% of the systems, respectively; the programming languages for the remaining 20 systems (14.5%) were not specified.

### 4.5.4. The publicity of the data sets

Among the 79 distinct data sets, 55 (69.6%) data sets were open source; 9 (11.4%) were the results of academic projects; 12 (15.2%) were the results of student projects; and 3 (3.8%) were commercial/industrial. All the data sets in 29 (61.7%) PSs were open source, which potentially allows the studies to be repeatable. In addition, some of the data sets used in ten (21.3%) PSs were open source, so these studies are at least partially repeatable. The other systems used in these ten studies were results of academic projects (in 3 PSs), the results of student projects (in 6 PSs), or commercial/industrial (in 3 PSs). The rest of the PSs (17%) used systems that were the results of academic projects (in 5 PSs), student projects (in 2 PSs), or both (in one PS).

## 4.6. Study quality assessment

The results of the quality assessment study show that the scores of the PSs range widely from 33.3% to 94.4% with an average of 69.2%. More specifically, the scores of conference PSs range between 44.4% and 88.9% with an average of 64.2%, and the scores of journal PSs range from 33.3% to 94.4% with an average of 73.5%, considerably better than the conference PSs. We found that 14 journal PSs (52.4%) scored above 77.8% (higher than the second best conference PS score). These results were expected and confirm the results found in other literature reviews (e.g., [47] that, with some exceptions, journal papers are typically more complete and of better quality than conference papers for several reasons, including space limitations, number of pages, and the depth required of the reported empirical evaluation. For interested readers, we recommend the six PSs (S8, S10, S37, S6, S1, and S27, in order from the highest) that scored higher than 85%.

The results in Table 4 show that all of the PSs either partially or adequately described the techniques applied for identifying refactoring opportunities (QA1), stated the aims of the study (QA3), and answered all study questions (QA14). Most of the PSs adequately described the data sets (QA8), clearly stated the purpose of the analysis (QA12), and reported analysis results that sufficiently add to the literature (QA17). Regarding QA2, it was found that a considerable percentage of the PSs (36.2% + 4.3% = 40.5%) did not provide definitions of the refactoring activities considered, and two PSs (S27 and S47) did not specify the refactoring activities. In PS S27, the authors stated that their technique is applicable to *generalization* refactoring activities without stating or defining these activities. In addition, the technique proposed in S47 is applicable for Extract Class refactoring, although the authors never used this refactoring activity name. Question QA2 is not applicable to S28 because the proposed technique was claimed to be applicable to all refactoring activities.

For QA4, we considered the total sizes of the data sets used in a PS, as defined in Section 3.6.2, where large, medium, and small data sets were given values of 1, 0.5, and 0, respectively. Therefore, the results of QA4 match the results reported and discussed in Section 4.5.2. Regarding QA5, it was found that only 31.9% of the PSs fully defined the measures applied to evaluate the refactoring results. The rest of the PSs either did not define the evaluation measures (36.2%) or defined only some of them (31.9%). These PSs either assumed that the readers were familiar with the evaluation measures used or omitted the definitions because of space limitations. Most of the PSs (80.8%) failed to describe the scoring systems for the evaluation measures and thus left the conclusions subjective. We believe that the reason for the lack of scoring system descriptions was that the authors relied on existing evaluation measures, such as precision and recall, that do not have common standard score descriptions. For example, a score of 65 does not have a common interpretation (i.e., whether it is good, satisfactory, or poor). Describing the scoring systems (QA13) makes the conclusions objective and facilitates comparing the results with the results obtained in other studies. In addition, defining the evaluation measures is important to eliminate any ambiguity regarding the interpretation of the results.

More than half of the PSs adequately described the data collection methods (QA6), which helps to make the reported studies repeatable. The results of QA7 are related to RQ2, where the six PSs that considered applicability-based evaluation, the weakest evaluation approach, were given a value of zero. Regarding QA9, most of the PSs (70.2%) failed to provide sufficient information regarding the expertise of the study participants, and the evaluation results greatly depend on such experience. The absence of such information makes the evaluation results questionable. PSs that adequately described the applied tools but reported insufficient information about the study participants were given a value of 0.5 and made up 55.3% of the PSs. Among the PSs that applied statistical methods, a high percentage clearly described the statistical methods and at least partially provided justifications for applying them (related to QA10 and QA11). Regarding QA15, we observed that more than half of the PSs with negative findings represented and interpreted the negative findings. The rest of the PSs either listed the negative findings but failed to interpret them or did not present the negative findings at all. Presenting and interpreting negative findings is important to understand the limitations of the proposed technique and thus help to find ways to improve the technique. The results of QA16 are related to the results reported in Section 4.4.4 and show that most of the PSs lack comparison studies. Finally, most of the PSs did not sufficiently discuss validity threats, which makes the causal inference and generalization of the results questionable. We noted that only 13.6% of the conference studies sufficiently discussed validity threats, whereas 48% of the journal studies did. This difference may be due to the conference papers' space limitations.

Generally, the quality assessment study shows that most of the PSs scored well for questions QA1, QA3, QA7, QA8, QA10, QA11, QA12, QA14, and QA17. Researchers are advised to focus more on issues regarding questions QA2, QA5, QA9, QA13, QA16, and QA18, where we found that most of the PSs have weaknesses.

## 5. Discussion and open issues

This section discusses and interprets the results reported in Section 4 and discusses related open issues.

### 5.1. Refactoring activities (Related to RQ1)

The results presented in Section 4.2 show that researchers are more interested in *Move Method*, *Extract Class*, and *Extract Method* than in other refactoring activities. These refactoring activities have major impacts on software design and require careful consideration. The high interest in these activities may indicate their importance in the software industry and suggests that these activities are potentially more frequently applied in practice than other activities. However, this indication is not supported by the results of studies that investigated the frequency of applying refactoring activities in practice (e.g., [39,53]. These studies found that *Move Method* and *Extract Method* are among the activities that are frequently applied in practice. However, some refactoring activities that are frequently addressed by the PSs, such as *Extract Class* and *Extract Superclass*, are rarely applied in practice. In addition, we have noticed that most of the refactoring activities proposed by Fowler [23] (52 out of 72 refactoring activities) were not considered in any of the PSs. We are not aware of any reported discussion regarding why these activities might be ignored. Some of these activities such as *Rename Field*, *Rename Method*, *Inline Temp*, and *Add Parameter*, are among the activities that were found as the most activities applied in practice [39,53]. These observations give an indication that there is a gap between the refactoring practice and the research in the area of identifying refactoring opportunities. To reduce this gap, researchers are encouraged to make use of the studies that explore the refactoring practice and focus their research more on the refactoring activities that are frequently applied in practice. Researchers are encouraged to improve the results of refactoring activities that are more commonly used in practice. Instead of performing more studies on identifying opportunities of refactoring activities that have already been considered and are rarely applied in practice, researchers are advised to propose techniques to identify the opportunities of refactoring activities that are frequently applied in practice, especially those that have not been considered yet, such as *Rename Field*, *Rename Method*, *Inline Temp*, and *Add Parameter*. Murphy-Hill et al. [39] found that the *Rename* refactoring activities, which are among the most frequently applied refactoring activities, are mostly performed automatically. Automating the refactoring process includes automating the main two steps: (1) identifying the refactoring opportunities and (2) performing the refactoring activities. Therefore, to help automate the refactoring of *Rename* activities, researchers are advised to propose and automate effective techniques to identify opportunities for these and similar activities.

### 5.2. Approaches to identifying refactoring opportunities (Related to RQ2)

The results reported in Section 4.3 show that the most frequently applied refactoring identification approaches are quality metrics-oriented, precondition-oriented, and clustering-oriented. Code slicing has been shown to be an effective approach in several software engineering areas, such as code testing and quality measurement. However, we believe it has rarely been applied to identify refactoring opportunities because of scalability. Code slicing is suitable for relatively small pieces of code; therefore, it is used to identify opportunities for *Extract Method* refactoring, which requires analyzing the code within a method. All of the PSs except S21 used static identification approaches, which cause the analyses to ignore the effects of certain important OO features, such as dynamic binding, on the identification findings. The concentration on static analysis might be because most refactoring activities are related to design issues that can be statically determined.

Table 7 shows the mapping between the identification approaches followed by the PSs and the corresponding refactoring activities considered by these PSs. The table shows that *Extract Method*, *Move Method*, and *Extract Class* refactoring opportunities were identified in the literature using most of the approaches considered. The opportunities for certain refactoring activities, such as *Extract Interface* and *Inline Class*, are identified by only a single approach. Due to their usefulness, flexibility, and strong relationship to software design, we found the quality metrics-oriented and precondition-oriented approaches to be applicable to identifying opportunities for a wide variety of refactoring activities.

One of the key open issues in this area is comparing the results of applying different approaches for identifying refactoring opportunities of a certain activity to determine the best approach. We have noticed that none of the PSs compared the refactoring identification results of the same refactoring activities across the use of different approaches. Researchers are directed to compare different

**Table 7**
Mapping between the identification approaches and the refactoring activities considered.

| Identification approach | Refactoring activities |
|---|---|
| Quality metrics-oriented | Extract Method, Move Method, Pull Up Method, Pull Up Constructor, Form Template Method, Parameterize Method, Extract Class, Extract Subclass, Extract Superclass, Move Class |
| Precondition-oriented | Extract Method, Move Method, Replace Method with Method Object, Replace Data Value with Object, Extract Class, Remove Parameter, Eliminate Return Value, Separate Query from Modifier, Encapsulate Downcast, Replace Temp with Query, Form Template Method, Pull Up Method, Extract Class, Extract Superclass, Replace Type Code with State/Strategy, Replace Conditional with polymorphism |
| Clustering-oriented | Extract Method, Move Method, Move Field, Inline Class, Extract Class, Move Class |
| Graph-oriented | Extract Method, Move Method, Extract Interface, Extract Class, Move Class |
| Code-slicing-oriented | Extract Method |
| Dynamic analysis-oriented | Move Method |

approaches to identify the opportunities of each of the considered refactoring activities. Several factors have to be considered in such comparison studies, such as the simplicity of approach automation, the scalability of the approach, and the accuracy of the approach in detecting refactoring opportunities. Approach automation is one of the key difficulties in performing such comparison studies. Most of the researchers reported that they automated their proposed techniques. However, we note that only a few researchers made their tools publicly available. Consequently, researchers interested in comparing results across techniques of different approaches will need to exert extra effort for tool development. Researchers are strongly advised to make the refactoring opportunity identification tools they've developed publicly available and to support these tools with well-documented user manuals.

Another key issue in this area is exploring the applicability of the key approaches to each of the refactoring activities. For example, as shown in Table 7, the opportunities of *Pull Up Method* refactoring activity were only identified using the quality metrics-oriented approach. Studies are required to explore whether other approaches are applicable to identify opportunities for *Pull Up Method* refactoring and whether such application leads to better identification results. Otherwise, justifications are required to explain why the other identification approaches are unsuitable for identifying these refactoring activities.

Finally, one of the key challenges related to the proposed approaches and techniques is reducing the corresponding gap between the industrial and research domains. Most of the authors of the considered PSs are from the academic field. We are not aware of any evidence for the use of these techniques in practice. Several studies explored the application of the automated techniques used to perform certain refactoring activities in practice (e.g., [39,40]. It is recommended that researchers in the field perform similar studies to investigate the use of the proposed techniques for refactoring opportunity identification in practice. Such studies are important to determine the key limitations of the proposed techniques and to guide researchers to related areas that require more attention and careful addressing.

## 5.3. Empirical evaluation approaches (Related to RQ3)

The results reported in Section 4.4 show that a high percentage of PSs (87.2%) evaluated their proposed techniques based on intuition, quality factors, a mutation-based approach, or a comparison-based evaluation approach. These evaluation approaches are frequently applied because they are potentially more reliable than other approaches; therefore, researchers are encouraged to apply one or more of these approaches to evaluate their techniques. Four PSs (S6, S32, S33, and S45) considered three evaluation approaches; 18 PSs (S7, S9, S10, S11, S13, S14, S18, S22, S23, S27, S28, S35, S38, S39, S40, S43, S46, and S37) applied two approaches; and the remaining 25 PSs evaluated their techniques using only one approach. Evaluating the techniques using multiple evaluation approaches increases confidence in the reliability of the results. Each of the approaches has its own validity threats. The intuition-based approach is subjective and depends strongly on the experience of the evaluators. The quality-based approach relies on the debatable expectation that refactoring improves code quality and on the assumption that the quality measures accurately indicate code quality. The mutation-based approach depends on the variety of mutations applied and on the assumption that the original code is well structured and not in need of refactoring. The comparison-based approach provides no information regarding refactoring candidates that go undetected by both the proposed and the compared technique. The behavior-based approach is a weak evaluation technique because it does not indicate whether the suggested candidates really need refactoring or whether applying the suggested refactoring improves the code quality. That is, the suggested refactoring may cause the code to be messier or poorer quality, even though the code has the same behavior before and after refactoring. The applicability-based evaluation is the weakest approach because it provides no evidence regarding the candidate results. Consequently, the last two evaluation approaches are not recommended.

The techniques for identifying opportunities for several refactoring activities, including the *Parameterized Method* and the *Pull Up Constructor* were only evaluated using the applicability-based approach. In addition, the techniques for identifying opportunities for *Extract Superclass* and *Remove Parameter* activities are evaluated using either only applicability-based approach or small data sets. Therefore, the evaluation results of these techniques are questionable. Researchers are recommended to perform empirical studies to evaluate these techniques using more reliable evaluation approaches such as initiation and mutation-based approaches. In addition, researchers should apply the reliable evaluation approaches when proposing new identification techniques.

Researchers in some related areas, such as software quality measurement, used to make their obtained data and analysis results available in on-line repositories. We are not aware of similar repositories for refactoring opportunity identification data. Researchers are advised to make all analysis data available in public repositories, including the intuitions of the developers, the quality measurement results, the mutated versions of the code, and the refactoring opportunity identification results. Standards can be set to agree on corresponding "gold sets" of refactoring opportunity data for each refactoring activity. Instead of starting from scratch, researchers can make use of available gold sets of data in corresponding repositories and consider them as benchmarks for comparison with their own obtained results.

### 5.4. Data sets employed (Related to RQ4)

The findings presented in Section 4.5 show that the literature in the area of refactoring identification is relatively rich in terms of the number of systems considered. This wide range of systems provides researchers with a wider selection of previously considered systems to which they can apply the newly proposed refactoring identification techniques and compare results with existing ones. The PSs differ widely in terms of the number of systems considered (from one to 14) and their sizes. The conclusions drawn by the PSs with more and larger data sets are more trustworthy than the conclusions obtained by the PSs with less and smaller data sets. In terms of the total size of considered systems, the results indicate that there is a danger in generalizing the results of almost half of the PSs because their data sets are either small or not reported. However, in the last three years (2011–2013) we have noted a decrease in the percentage of PSs that rely on small data sets (i.e., 34.6%, compared to 57% for the older PSs). The reliance on large data sets has greatly increased, from 9.5% for the PSs published before 2011 to 50%, for the PSs published since 2011. In general, these observations indicate that more recent research has used larger data sets, making the conclusions of recent PSs more trustworthy.

Another factor that has affected the size of the considered systems is the granularity of the required analysis. That is, identifying opportunity of performing some refactoring activities, such as the *Extract Method*, require extracting data from each code statement and analyzing the relationships among the extracted data. Other refactoring activities, such as *Move Class*, require extracting and analyzing data at the class level. The amount of data per class to be extracted and analyzed in the former case is much higher than in the latter case. We identified three main analysis granularity levels, ordered from the finest to the most coarse: (1) analyzing relationships between the statements within each method (e.g., to identify *Extract Method* refactoring opportunities), (2) analyzing relationships between the elements within a class (e.g., to identify *Extract Class* refactoring opportunities), and (3) analyzing relationships between classes (e.g., to identify *Move Class* refactoring opportunities). We classified the PSs into these categories, although some PSs fall within more than one category because they consider several refactoring activities. Based on this classification, we found that 11, 16, and 30 PSs fall within the three categories, respectively. We found that 9.1%, 18.7%, and 36.7% of the PSs that use large data sets are in the first, second, and third categories, respectively, and 63.6%, 50%, and 36.7% of the PSs that use small data sets are in the first, second, and third categories, respectively. These observations show that to reduce the amount of data to be extracted and analyzed, researchers tend to use smaller data sets for refactoring activities that require fine-grained code analysis than for refactoring activities that require coarse-grained analysis.

We have noticed that the studies of opportunity for six of the considered refactoring activities, including Replace Method with Method Object, Replace Data Value with Object, Separate Query from Modifier, Encapsulate Downcast, Replace Temp with Query, and Eliminate Return Value, were evaluated using small size data sets. Replicated studies are required to prove or disprove the obtained results. In addition, when considering other refactoring activities, researchers are directed to use numerous and relatively large data sets.

Java was found to be the dominant programming language for the data sets considered, whereas other object-oriented programming languages, such as C++ and Smalltalk, were rarely used. This popularity of Java systems may be because the problem of refactoring is relatively new (the oldest PS was published in 2001), and Java has been the most popular programming language since the problem began to be studied. In addition, all the examples in Fowler's refactoring book are in Java. We have noted that most of the refactoring tools reported in the PSs support Java refactoring systems, although older systems written in older programming languages, such as C++ and Smalltalk, are potentially in greater need of maintenance and refactoring. We encourage future research in OO programming languages other than Java.

JHotDraw was found to be the most frequently used data set (34% of the PSs). This high usage is because it was built using design patterns and is well known for its good design. Certain other data sets, such as Apache Ant, ArgoUML, and GanttProject, were also considered in more than one study; some of the these studies are by the same authors, so the related data were already available to them for their more recent studies.

Although a considerable number of PSs evaluated their techniques using common data sets and considered the identification of common refactoring activities, only two (S14 and S39) reported a comparison between their results and the results of other studies. This lack of comparison results can be due to the unsuitability of including detailed refactoring results within the paper. For example, due to space limitations, some researchers tend to report only the numbers of refactoring opportunities identified instead of listing the details of each. This shortage of details does not allow researchers to directly compare results across studies. Instead, some researchers apply their own techniques and previously existing ones, obtain the refactoring results, and compare them. This approach was followed by 11 (23.4%) PSs.

To enable others to perform comparison studies, researchers are encouraged to evaluate their techniques using open-source systems, publicize the details of their results, and make the refactoring identification tools they develop publicly available. As indicated in Section 5.3, special repositories can be established for this purpose, and corresponding gold sets can be maintained. Whenever they propose a new technique to identify opportunities for refactoring activities that are already considered in the literature, researchers can apply their new technique to the data sets available in the corresponding repositories and compare their results with the corresponding gold sets.

The results show that 74.6% of the data sets used were open source and that 83% of the PSs used partially or fully public data sets. These high percentages allow the studies to be repeatable. However, the unbalanced percentages of data set sources (i.e., the high percentage of open source data sets and the low percentage of industrial-based data sets [4.3%]) raises questions of generality for the results and conclusions. Researchers are encouraged to use more industrial-based data sets to validate or invalidate their results.

## 6. Conclusions and future work

This paper reports a SLR that identifies and analyzes proposed and empirically evaluated techniques to identify opportunities for refactoring object-oriented code. A total of 2883 potential articles were identified using automatic searching in seven scientific digital libraries. After screening the articles, performing a pilot search, and contacting the corresponding authors, we selected and analyzed 47 PSs.

The results of the systematic review indicate that the identification of refactoring opportunities is a highly active area of research. Although most researchers in this area are from academia, some of the participants in the reported empirical studies are from industry, indicating that researchers are somewhat in contact with people in industry. The results of the SLR show that the *Move Method*, *Extract Class*, and *Extract Method* refactoring activities are of greater interest to researchers than other refactoring activities, giving the impression that these activities are more frequently applied in practice during the refactoring process than other refactoring activities. We have identified six main approaches followed by researchers to identify refactoring opportunities and found that the quality metrics-oriented, precondition-oriented, and clustering-oriented approaches are the most frequently followed. Most of the researchers (87.2%) applied well known and commonly applied software engineering-related empirical evaluation approaches, including intuition, using quality factors, applying a mutation-based approach, or using a comparison-based evaluation approach, which increases confidence in the findings of the studies. Frequently, the use of these approaches indicates that they are more reliable than other approaches. In the reported empirical studies, a relatively large number of data sets were considered, providing researchers with a rich pool of data sets to select and use in future related studies. The present results can be then compared to the new results. Some data sets were used in several studies; therefore, the related results can be compared to each other. We have noted that most of the recent PSs (published since 2011) used relatively large data sets, making the results and conclusions of these PSs more trustworthy than the results and conclusions of older studies. Finally, we observed that most of the data sets used are open source, which helps to make the studies repeatable.

We have identified several points that should receive more attention in researchers' future studies, and we have noted several related areas that require more research and investigation. Researchers are advised to invite experts from industry to participate and share their expertise. We found that 72.2% of the refactoring activities proposed by Fowler were not considered by any study. Therefore, researchers are encouraged to expand the coverage of their work to include more refactoring activities. Researchers typically compare the results of their techniques with the results of other techniques that follow the same identification approach. Comparing results across techniques that follow different approaches is open for future investigation.

To increase confidence in empirical evaluation results, researchers are advised to use relatively large data sets implemented with different programming languages and to consider a mixture of open source and industrial systems. Based on the quality assessment study, researchers are advised to pay greater attention to (1) clearly stating and defining the refactoring activities considered, (2) fully defining the evaluation measures, (3) adequately describing the study participants, (4) clearly describing the scoring systems used, (4) comparing the results with previous findings, and (5) discussing validity threats. Most of the existing studies suffered from weaknesses related to at least one of these research points.

## Acknowledgments

The author would like to acknowledge the support of this work by Kuwait University Research Grant QI03/13. In addition, the author would like to thank Anas Abdin for assisting in collecting and analyzing the required data.

## Appendix A. Search strings

### A.1. Library: ACM

*Search String*: (((((((Title:"object-oriented" or Title:"object oriented" or Title:class or Title:classes or Title:method or Title:methods or Title:software or Title:code or Abstract:"object-oriented" or Abstract:"object oriented" or Abstract:class or Abstract:classes or Abstract:method or Abstract:methods or Abstract:software or Abstract:code))) and (Title:refactor* or Abstract:refactor*))) and (Title:predict* or Title:determin* or Title:identif* or Title:indicat* or Title:detect* or Title:track* or Abstract:predict* or Abstract:determin* or Abstract:identif* or Abstract:indicat* or Abstract:detect* or Abstract:track*))) and (PublishedAs:journal OR PublishedAs:proceeding OR PublishedAs:transaction).

### A.2. Library: Springer

*Search String*: ("object-oriented" OR "object oriented" OR class OR classes OR method OR methods OR attribute OR attributes) AND refactor* AND (estimat* OR predict* OR determin* OR identif* OR indicat* OR detect* OR track*).

### A.3. Library: Scopus

*Search String*: (ABS(object-oriented OR "object oriented" OR class OR classes OR method OR methods OR attribute OR attributes) OR TITLE(object-oriented OR "object oriented" OR class OR classes OR method OR methods OR attribute OR attributes)) AND (ABS(refactor*) OR TITLE(refactor*))AND (ABS(estimat* OR predict* OR determin* OR identif* OR indicat* OR detect* OR track*) OR TITLE(estimat* OR predict* OR determin* OR identif* OR indicat* OR detect* OR track*)) AND (LIMIT-TO(LANGUAGE,"English")) AND (LIMIT-TO(SRCTYPE,"p") OR LIMIT-TO(SRCTYPE,"j")).

### A.4. Library: IEEE

*Search String*: ((("Publication Title":"object-oriented" OR "Publication Title":"object oriented" OR "Publication Title":class OR "Publication Title":classes OR "Publication Title":method OR "Publication Title":methods OR "Publication Title":code OR "Publication Title":software OR "Abstract":"object-oriented" OR "Abstract": "object oriented" OR "Abstract":class OR "Abstract":classes OR "Abstract":method OR "Abstract":methods OR "Abstract":software OR "Abstract":code) AND (p_Publication_Title:refactor OR "Publication Title":refactoring OR "Abstract":refactor OR "Abstract":refactoring) AND ("Publicatio Title":estimate OR "Publication Title":estimating OR "Publication Title":estimation OR "Publication Title":predict OR "Publication Title":predicting OR "Publication Title":prediction OR "Publication Title":determine OR "Publication Title":determining OR "Publication Title":identify OR "Publication Title":identifying OR "Publication Title":identification OR "Publication Title":indicat* OR "Publication Title":detect* OR "Publication Title":track OR "Publication Title":tracking OR "Abstract":estimate OR "Abstract":estimating OR "Abstract":estimation OR "Abstract":predict OR "Abstract":predicting OR "A bstract":prediction OR "Abstract":determine OR "Abstract":determining OR "Abstract":identify OR "Abstract":identifying OR "Abstract":identification OR "Abstract":indicat* OR "Abstract": detect* OR "Abstract":track OR "Abstract":tracking))).

### A.5. Library: ISI

*Search String*: (TS=((object-oriented OR "object oriented" OR class OR classes OR method OR methods OR attribute OR attributes) AND refactor* AND (estimat* OR predict* OR determin* OR identif* OR indicat* OR detect* OR track*))) AND Language=(English).

### A.6. Library: ScienceDirect

*Search String*: (abs(object-oriented OR object oriented OR class OR classes OR method OR methods OR software OR code) OR ttl(object-oriented OR object oriented OR class OR classes OR method OR methods OR software OR code)) AND (abs(refactor*) OR ttl(refactor*))AND (abs(estimat* OR predict* OR determin* OR

**Table B.1**
PS primary extracted data.

| PS reference | Refactoring activities | Refactoring opportunity identification approach | Evaluation approaches | Names of data sets | No. of small data sets | No. of medium data sets | No. of large datasets | Programming languages of the data sets | Venues of the data sets |
|---|---|---|---|---|---|---|---|---|---|
| S1: Al Dallal [1] | Extract Subclass | Quality metrics-oriented | Mutation-based | Art of Illusion, FreeMind, GanttProject, JabRef, Openbravo, JHotDraw | 1 | 5 | 0 | Java | Open source |
| S2: Al Dallal and Briand [2] | Move Method, Extract Class | Quality metrics-oriented | Mutation-based | JHotDraw | 0 | 1 | 0 | Java | open source |
| S3: Alkhalid et al. [3] | Extract Method | Clustering-oriented | Applicability-based | PDF Split and Merge, JLOC, CSGestionnaire | 3 | 0 | 0 | Java | open source |
| S4: Alkhalid et al. [4] | Move Method | Clustering-oriented | Quality-based | CSGestionnaire | 1 | 0 | 0 | Java | Open source |
| S5: Alkhalid et al. [5] | Move Class | Clustering-oriented | Quality-based | Trama, Front End for MySQL Domain | 2 | 0 | 0 | Java | Open source |
| S6: Bavota et al. [6] | Extract Class | Quality metrics-oriented | Intuition-based, mutation-based, comparison-based | ArgoUML, Eclipse, GanttProject, JHotDraw, Xerces | 0 | 3 | 2 | Java | Open source |
| S7: Bavota et al. [7] | Move Class | Quality metrics-oriented | Intuition-based, mutation-based | JHotDraw, eTour, GESA, SESA, SMOS | 3 | 2 | 0 | Java | Open source, student project |
| S8: Bavota et al. [8] | Extract Class | Graph-oriented | Mutation-based | ArgoUML, JHotDraw, Eclipse, GanttProject | 0 | 2 | 2 | Java | Open source |
| S9: Bavota et al. [9] | Move Class | Quality metrics-oriented | Intuition-based, quality-based | GanttProject, jEdit, JHotDraw, jVLT, eXVantage, GESA, eTour, SESA, SMOS | 3 | 3 | 3 | Java | Open source, student project, commercial project |
| S10: Bavota et al. [10] | Move Method | Quality metrics-oriented | Intuition-based, quality-based | jEdit, JFreeChart, AgilePlanner, eXVantage, GESA, SMOS | 1 | 0 | 5 | Java | Open source, student project, commercial project |
| S11: Bavota et al. [11] | Extract Class | Quality metrics-oriented | Mutation-based, comparison-based | ArgoUML, JHotDraw | 0 | 1 | 1 | Java | Open source |
| S12: Cassell et al. [13] | Extract Class | Clustering-oriented | Quality-based | FreeCol, Heritrix, Jena, Weka | 4 | 0 | 0 | Java | Open source |
| S13: Czibula and Czibula [15] | Move Method, Move Attribute, Inline Class, Extract Class | Clustering-oriented | Mutation-based, comparison-based | JHotDraw, ‘A’ | 1 | 1 | 0 | Java | Open source, commercial project |
| S14: Czibula and Serban [16] | Move Method, Move Attribute, Inline Class | Clustering-oriented | Mutation-based, comparison-based | JHotDraw | 1 | 0 | 0 | Java | Open source |
| S15: Du Bois et al. [18] | Extract Method, Move Method, Replace Method with Method Object, Replace Data Value with Object, Extract Class | Precondition-oriented | Quality-based | Apache Tomcat | 1 | 0 | 0 | Java | Open source |
| S16: Fokaefs et al. [19] | Extract Class | Clustering-oriented | Intuition-based | eRisk, SelfPlanner | 0 | 0 | 0 | C++ | Academic project, student project |
| S17: Fokaefs et al. [20] | Extract Class | Clustering-oriented | Intuition-based | JHotDraw | 1 | 0 | 0 | Java | Open source |
| S18: Fokaefs et al. [21] | Extract Class | Clustering-oriented | Intuition-based | JHotDraw, CLRServerPack, TPMSim, CoverFlow | 4 | 0 | 0 | Java | Open source, student project |
| S19: Fokaefs et al. [22] | Move Method | Quality metrics-oriented | Intuition-based | Video Store, LAN-simulation | 2 | 0 | 0 | Java | Academic project |
| S20: Higo et al. [24] | Extract Method, Pull Up Method | Quality metrics-oriented | Applicability-based | Ant | 0 | 1 | 0 | Java | Open source |
| S21: Higo et al. [25] | Extract Class, Extract Superclass, Extract Method, Pull Up Method, Form Template Method, Move Method, Parameterize Method, Pull Up Constructor | Quality metrics-oriented | Applicability-based | Ant | 0 | 1 | 0 | Java | Open source |
| S22: Hotta et al. [26] | Form Template Method | Precondition-oriented | Comparison-based, behavior-based | Apache-Ant, Apache-Synapse | 0 | 2 | 0 | Java | Open source |
| S23: Kanemitsu et al. [27] | Extract Method | Graph-oriented | Comparison-based, behavior-based | Students Old Code, Ant | 2 | | | Java | Open source, student project |
| S24: Kataoka et al. [28] | Remove Parameter, Eliminate Return Value, Separate Query from Modifier, Encapsulate Downcast, Replace Temp with Query | Precondition-oriented | Intuition-based | Nebulous | 1 | 0 | 0 | Java | Open source |

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| S25: Kimura et al. [29] | Move Method | Dynamic analysis-oriented | Intuition-based | FRISC, MASU | 1 | 0 | 1 | Java | Open source, academic project |
| S26: Lee et al. [32] | Pull Up Method, Form Template Method, Move Method, Extract Class, Extract Superclass | Precondition-oriented | Applicability-based | JFreeChart, ANTLR, JHotDraw, BCEL | 2 | 2 | 0 | Java | Open source |
| S27: Liu et al. [33] | Generalization refactorings | Precondition-oriented | Intuition-based, comparison-based | GEF, JSM, Jface, Thout Reader, AutoMed | 2 | 3 | 0 | Java | Open source, student project |
| S28: Mahouachi et al. [34] | open, any refactoring activity | quality metrics-oriented | comparison-based, behavior-based | GanttProject, Xerces-J, ArgoUML, Quick UML, LOG4 J, AZUREUS | 2 | 2 | 2 | Java | Open source |
| S29: Melton and Tempero [35] | Extract Interface | Graph-oriented | Quality-based | Azureus | 0 | 0 | 1 | Java | Open source |
| S30: Ments et al. [37] | Remove Parameter, Extract Superclass, Pull Up Method | Precondition-oriented | Intuition-based | Soul, Smalltalk Collection hierarchy, HotDraw | 3 | 0 | 0 | Smalltalk | Academic project |
| S31: Oliveto et al. [41] | Move Method | Quality metrics-oriented | Mutation-based | ArgoUML | 0 | 0 | 1 | Java | Open source |
| S32: Pan et al. [42] | Move Class | Graph-oriented | Intuition-based, quality-based, comparison-based | Trama, Front End | 2 | 0 | 0 | Java | Open source |
| S33: Pan et al. [43] | Move Class | Graph-oriented | Intuition-based, quality-based, comparison-based | Trama, Front End | 2 | 0 | 0 | Java | Open source |
| S34: Pan et al. [44] | Move Method | Graph-oriented | Intuition-based | JHotDraw | 1 | 0 | 0 | Java | Open source |
| S35: Pan et al. [45] | Move Method | Graph-oriented | Intuition-based, quality-based | LAN-Simulation | 1 | 0 | 0 | Java | Academic project |
| S36: Rao and Reddy [48] | Extract Class | Clustering-oriented | Quality-based | Bank Application | 1 | 0 | 0 | Java | Student project |
| S37: Sales et al. [49] | Move Method | Quality metrics-oriented | Mutation-based, comparison-based | Ant, ArgoUML, Cayenne, DrJava, FreeCol, FreeMind, Jmeter, Jruby, JTOpen, Maven, Megamek, WCT, Weka, JHotDraw | 1 | 10 | 3 | Java | Open source |
| S38: Seng et al. [50] | Move Method | Precondition-oriented | Intuition-based, mutation-based | JHotDraw, InnerClasses | 1 | 1 | 0 | Java | Open source |
| S39: Serban and Czibula [51] | Move Method, Move Attribute, Inline Class, Extract Class | Clustering-oriented | Mutation-based, comparison-based | JHotDraw | 1 | 0 | 0 | Java | Open source |
| S40: Tairas and Gray [54] | Extract Method, Pull Up Method (not discussed in the evaluation) | Precondition-oriented | Comparison-based, applicability-based | Apache Ant, Columba, EMF, Hibernate, Jakarta-Jmeter, Jedit, JFreeChart, Jruby, Squirrel-SQL | 0 | 9 | 0 | Java | Open source |
| S41: Tourwé and Mens [56] | Remove Parameter, Extract super class, extract method | Precondition-oriented | Applicability-based | SOUL | 1 | 0 | 0 | SmallTalk | Academic project |
| S42: Tsantalis and Chatzigeogiou [57] | Extract Method | Code slicing-oriented | Intuition-based | Telephone Exchange Emulator | 1 | 0 | 0 | Java | Academic project |
| S43: Tsantalis and Chatzigeogiou [58] | Move Method | Precondition-oriented | Intuition-based, quality-based | JFreeChart, Jedit, Jmol, Diagram, SelfPlanner | 1 | 4 | 0 | Java, C++ | Open source, academic project |
| S44: Tsantalis and Chatzigeogiou [59] | Replace Type Code with State/Strategy, Replace Conditional with polymorphism | Precondition-oriented | Intuition-based | Violet, IHM, Nutch | 2 | 1 | 0 | Java | Open source |
| S45: Tsantalis and Chatzigeogiou [60] | Extract Method | Code slicing-oriented | Intuition-based, quality-based | JFreeChart, WikiDev, SelfPlanner | 1 | 0 | 0 | Java | Open source, academic project |
| S46: Yang et al. [62] | Extract Method | Quality metrics-oriented | Intuition-based, quality-based | ThroutReader | 0 | 1 | 0 | Java | Open source |
| S47: Zhao and Hayes [64] | Extract Class | Quality metrics-oriented | Intuition-based | Student Project | 1 | 0 | 0 | Java | Student project |

identif* OR indicat* OR detect* OR track*) OR ttl(estimat* OR predict* OR determin* OR identif* OR indicat* OR detect* OR track*)).

### A.7. Library: Compendex and Inspec (using Engineering Village web-based discovery platform)

*Search String*: ((object-oriented OR "object oriented" OR class OR classes OR method OR methods OR attribute OR attributes) wn TI OR (object- oriented OR "object oriented" OR class OR classes OR method OR methods OR attribute OR attributes) wn AB) AND ((refactor*) wn TI or (refactor*) wn AB) AND ((estimat* OR predict* OR determin* OR identif* OR indicat* OR detect* OR track*) wn TI OR (estimat* OR predict* OR determin* OR identif* OR indicat* OR detect* OR track*) wn AB).

## Appendix B:. PS primary extracted data

See Table B.1.

## References

[1] J. Al Dallal, Constructing models for predicting extract subclass refactoring opportunities using object-oriented quality metrics, J. Inform. Softw. Technol. Arch. 54 (10) (2012) 1125–1141.

[2] J. Al Dallal, L.C. Briand, A Precise Method-Method Interaction-Based Cohesion Metric for Object-Oriented Classes, ACM Transact. Softw. Eng. Methodol. (TOSEM) TOSEM 21 (2) (2012). Article No. 8.

[3] A. Alkhalid, M. Alshayeb, S. Mahmoud, Software refactoring at the function level using new Adaptive K-Nearest Neighbor algorithm, Adv. Eng. Softw. 41 (10–11) (2010) 1160–1178.

[4] A. Alkhalid, M. Alshayeb, S.A. Mahmoud, Software refactoring at the class level using clustering techniques, J. Res. Practice Inform. Technol. 43 (4) (2011) 285–306.

[5] A. Alkhalid, M. Alshayeb, S.A. Mahmoud, Software refactoring at the package level using clustering techniques, Software IET 5 (3) (2011) 276–284.

[6] G. Bavota, A. De Lucia, A. Marcus, R. Oliveto, Automating extract class refactoring: an improved method and its evaluation, Empir. Softw. Eng. (2013) 1–48.

[7] G. Bavota, A. De Lucia, A. Marcus, R. Oliveto, Using structural and semantic measures to improve software modularization, Empir. Softw. Eng. 18 (5) (2013) 901–932.

[8] G. Bavota, A. De Lucia, R. Oliveto, Identifying Extract Class refactoring opportunities using structural and semantic cohesion measures, J. Syst. Softw. 84 (3) (2011) 397–414.

[9] G. Bavota, M. Gethers, R. Oliveto, D. Poshyvanyk, A. De Lucia, Improving software modularization via automated analysis of latent topics and dependencies, ACM Transact. Softw. Eng. Methodol. 23 (1) (2014). Article No. 4.

[10] G. Bavota, R. Oliveto, M. Gethers, D. Poshyvanyk, A. De Lucia, Methodbook: recommending move method refactorings via relational topic models, IEEE Trans. Software Eng. (2013), http://dx.doi.org/10.1109/TSE.2013.60.

[11] G. Bavota, R. Oliveto, A. De Lucia, G. Antoniol, Y. Gueheneuc, Playing with refactoring: identifying extract class opportunities through game theory, in: IEEE International Conference on Software Maintenance (ICSM), 2010, pp. 1–5.

[12] P. Brereton, B. Kitchenhama, D. Budgenb, M. Turnera, M. Khalil, Lessons from applying the systematic literature review process within the software engineering domain, J. Syst. Softw. 80 (4) (2007) 571–583.

[13] K. Cassell, P. Andreae, L. Groves, A Dual clustering approach to the extract class refactoring, in: Proceedings on the 23rd International Conference on Software Engineering and Knowledge Engineering (SEKE'11) Miami, FL, USA, 2011.

[14] J. Cohen, A coefficient of agreement for nominal scales, Educ. Psychol. Measur. 20 (1960) 37–46.

[15] I.G. Czibula, G. Czibula, Hierarchical clustering based automatic refactorings detection, WSEAS Transact. Electron. 5 (7) (2008) 291–302.

[16] I.G. Czibula, G. Serban, Improving systems design using a clustering approach, IJCSNS Int. J. Comput. Sci. Network Security 6 (12) (2006) 40–49.

[17] D. Dig, R.E. Johnson, The role of refactorings in API evolution, in: Proceedings of the 21st IEEE International Conference on Software Maintenance, 2005, pp. 389–398.

[18] B. Du Bois, S. Demeyer, J. Verelst, Refactoring – improving coupling and cohesion of existing code, in: Proceedings of the 11th Working Conference on Reverse Engineering, 2004, pp. 144–151.

[19] M. Fokaefs, N. Tsantalis, A. Chatzigeorgiou, J. Sander, Decomposing object-oriented class modules using an agglomerative clustering technique, in: IEEE International Conference on Software Maintenance, Canada, 2009, pp. 93–101.

[20] M. Fokaefs, N. Tsantalis, E. Stroulia, A. Chatzigeorgiou, JDeodorant: identification and application of extract class refactorings, in: Proceedings of the 33rd International Conference on Software Engineering, 2011, pp. 1037–1039.

[21] M. Fokaefs, N. Tsantalis, E. Stroulia, A. Chatzigeorgiou, Identification and application of Extract Class refactorings in object-oriented systems, J. Syst. Softw. 85 (10) (2012) 2241–2260.

[22] M. Fokaefs, N. Tsantalis, E. Stroulia, A. Chatzigeorgiou, JDeodorant: Identification and removal of Feature Envy bad smells, in: Proceedings of IEEE International Conference on Software Maintenance, 2007, pp. 467–468.

[23] M. Fowler, Refactoring: Improving the Design of Existing Code, Addison-Wesley Longman Publishing Co. Inc., Boston, MA, 1999.

[24] Y. Higo, T. Kamiya, S. Kusumoto, K. Inoue, Aries: Refactoring support environment based on code clone analysis, in: Proceedings of the 8th IASTED International Conference on Software Engineering and Applications, Article No. 436–084, 2004, pp. 222–229.

[25] Y. Higo, S. Kusumoto, K. Inoue, A metric-based approach to identifying refactoring opportunities for merging code clones in a Java software system, J. Softw. Maintenance Evolut.: Res. Practice 20 (6) (2008) 435–461.

[26] K. Hotta, Y. Higo, S. Kusumoto, Identifying, Tailoring, and Suggesting Form Template Method Refactoring Opportunities with Program Dependence Graph, in: Proceedings of the 16th European Conference on Software Maintenance and Reengineering, 2012, pp. 53–62.

[27] T. Kanemitsu, Y. Higo, S. Kusumoto, A visualization method of program dependency graph for identifying extract method opportunity, in: Proceedings of the 4th Workshop on Refactoring Tools, 2011, pp. 8–14.

[28] Y. Kataoka, D. Notkin, M. D. Ernst, W.G. Griswold, automated support for program refactoring using invariants, in: Proceedings of the IEEE International Conference on Software Maintenance, 2001, pp. 736.

[29] S. Kimura, Y. Higo, H. Igaki, S. Kusumoto, Move code refactoring with dynamic analysis, in: Proceedings of the IEEE International Conference on Software Maintenance (ICSM), 2012, pp. 575–578.

[30] B. Kitchenham, S. Charters, Guidelines for Performing Systematic Literature Reviews in Software Engineering, Technical Report EBSE 2007, Keele University, UK, 2007.

[31] J. Landis, G. Koch, Measurement of observer agreement for categorical data, Biometrics 33 (1977) 159–174.

[32] S. Lee, G. Bae, H.S. Chae, D. Bae, Y.R. Kwon, Automated scheduling for clone-based refactoring using a competent GA, Software—Practice Exper. 41 (5) (2011) 521–550.

[33] H. Liu, Z. Niu, Z. Ma, W. Shao, Identification of generalization refactoring opportunities, Autom. Softw. Eng. 20 (1) (2013) 81–110.

[34] R. Mahouachi, M. Kessentini, K. Ghedira, A new design defects classification: marrying detection and correction, in: Proceedings of the 15th International Conference on Fundamental Approaches to Software Engineering, 2012, pp. 455–470.

[35] H. Melton, E. Tempero, The CRSS metric for package design quality, in: Proceedings of the 30th Australasian Conference on Computer Science, vol. 62, 2007, pp. 201–210.

[36] T. Mens, T. Tourwé, A survey of software refactoring, IEEE Trans. Software Eng. 30 (2) (2004) 126–139.

[37] T. Mens, T. Tourwé, F. Muñoz, Beyond the Refactoring Browser: Advanced Tool Support for Software Refactoring, in: Proceedings of the 6th International Workshop on Principles of Software Evolution, 2003, pp. 39.

[38] M. Misbhauddin, M. Alshayeb, UML model refactoring: a systematic literature review, Empir. Softw. Eng. (2013) 1–46.

[39] E.R. Murphy-Hill, C. Parnin, A.P. Black, How we refactor, and how we know it, IEEE Trans. Software Eng. 38 (1) (2012) 5–18.

[40] S. Negara, N. Chen, M. Vakilian, R.E. Johnson, D. Dig, A comparative study of manual and automated refactorings, in: Proceedings of the European Conference on Object-Oriented Programming (ECOOP), 2013, pp. 552–576.

[41] R. Oliveto, M. Gethers, G. Bavota, D. Poshyvanyk, A. De Lucia, Identifying method friendships to remove the feature envy bad smell (NIER track), In: Proceedings of the 33rd International Conference on Software Engineering, 2011, pp. 820–823.

[42] W.F. Pan, B. Jiang, B. Li, Refactoring software packages via community detection in complex software networks, Int. J. Autom. Comput. 10 (2) (2013) 157–166.

[43] W. Pan, WB. Jiang, Y. Xu, Refactoring packages of object-oriented software using genetic algorithm based community detection technique, Int. J. Comput. Appl. Technol. 48 (3) (2013) 185–194.

[44] W. Pan, B. Li, Y. Ma, J. Liu, Y. Qin, Class structure refactoring of object-oriented softwares using community detection in dependency networks, Frontiers Comput. Sci. China 3 (3) (2009) 396–404.

[45] W.F. Pan, J. Wang, M.C. Wang, Identifying the move method refactoring opportunities based on evolutionary algorithm, Int. J. Model. Ident. Control 18 (2) (2013) 182–189.

[46] E. Piveta, Improving the search for refactoring opportunities on object-oriented and aspect-oriented software, Ph.D. thesis, Univeridade Federal Do Rio Grande Do Sul, Porto Alegre, 2009.

[47] D. Radjenović, M. Heričko, R. Torkar, A. Živkovič, Software fault prediction metrics: a systematic literature review, Inf. Softw. Technol. 55 (8) (2013) 1397–1418.

[48] AA. Rao, Kn. Reddy, Identifying clusters of concepts in a low cohesive class for extract class refactoring using metrics supplemented agglomerative clustering technique, Int. J. Comput. Sci. Issues 8 (5-2) (2011) 185–194.

[49] V. Sales, R. Terra, L.F. Miranda, M.T. Valente, Recommending move method refactorings using dependency sets, in: IEEE 20th Working Conference on Reverse Engineering (WCRE), 2013, pp. 232–241.

[50] O. Seng, J. Stammel, D. Burkhart, Search-based determination of refactorings for improving the class structure of object-oriented systems, in: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, 2006, pp. 1909–1916.

[51] G. Serban, I.G. Czibula, Restructuring software systems using clustering, in: 22nd International Symposium on Computer and Information Sciences, 2007, pp. 1–6.

[52] D. Sjoeberg, J. Hannay, O. Hansen, V. Kampenes, A. Karahasanovic, N. Liborg, A. Rekdal, A survey of controlled experiments in software engineering, IEEE Trans. Software Eng. 31 (9) (2005) 733–753.

[53] G. Soares, R. Gheyi, E.R. Murphy-Hill, B. Johnson, Comparing approaches to analyze refactoring activity on software repositories, J. Syst. Softw. 86 (4) (2013) 1006–1022.

[54] R. Tairas, J. Gray, Increasing clone maintenance support by unifying clone detection and refactoring activities, Inf. Softw. Technol. 54 (12) (2012) 1297–1307.

[55] K. Taneja, D. Dig, T. Xie, Automated detection of API refactorings in libraries, in: Proceedings of the 22 IEEE/ACM International Conference on Automated Software Engineering, 2007, pp. 377–380.

[56] T. Tourwé, T. Mens, Identifying refactoring opportunities using logic meta programming, in: Proceedings of the 7th European Conference on Software Maintenance and Reengineering, 2003, pp. 91–100.

[57] N. Tsantalis, A. Chatzigeorgiou, Identification of extract method refactoring opportunities, in: Proceedings of the 13th European Conference on Software Maintenance and Reengineering, 2009, pp. 119–128.

[58] N. Tsantalis, A. Chatzigeorgiou, Identification of move method refactoring opportunities, IEEE Trans. Software Eng. 35 (3) (2009) 347–367.

[59] N. Tsantalis, A. Chatzigeorgiou, Identification of refactoring opportunities introducing polymorphism, J. Syst. Softw. 83 (3) (2010) 391–404.

[60] N. Tsantalis, A. Chatzigeorgiou, Identification of extract method refactoring opportunities for the decomposition of methods, J. Syst. Softw. 84 (10) (2011) 1757–1782.

[61] R.D. Wangberg, A literature review on code smells and refactoring, Master Thesis, Department of Informatics, University of Oslo, 2010.

[62] L. Yang, H. Liu, Z. Niu, Identifying Fragments to be Extracted from Long Methods, in: Proceedings of the 16th Asia-Pacific Software Engineering Conference, 2009, pp. 43–49.

[63] M. Zhang, T. Hall, N. Baddoo, Code Bad Smells: a review of current knowledge, J. Software Maintenance Evolut.: Res. Practice 23 (3) (2011) 179–202.

[64] L. Zhao, J. Hayes, Predicting classes in need of refactoring: an application of static metrics, in: Proceedings of the 2nd International PROMISE Workshop, Philadelphia, Pennsylvania USA, 2006.