The MSR Cookbook
Mining a Decade of Research

Hadi Hemmati, Sarah Nadi, Olga Baysal, Oleksii Kononenko, Wei Wang, Reid Holmes, Michael W. Godfrey
Software Architecture Group
David R. Cheriton School of Computer Science
University of Waterloo, Canada
{hhemmati, snadi, obaysal, okononen, w65wang, rtholmes, migod}@uwaterloo.ca

***Abstract*—The Mining Software Repositories (MSR) research community has grown significantly since the first MSR workshop was held in 2004. As the community continues to broaden its scope and deepens its expertise, it is worthwhile to reflect on the best practices that our community has developed over the past decade of research. We identify these best practices by surveying past MSR conferences and workshops. To that end, we review all 117 full papers published in the MSR proceedings between 2004 and 2012. We extract 268 comments from these papers, and categorize them using a grounded theory methodology. From this evaluation, four high-level themes were identified: data acquisition and preparation, synthesis, analysis, and sharing/replication. Within each theme we identify several common recommendations, and also examine how these recommendations have evolved over the past decade. In an effort to make this survey a living artifact, we also provide a public forum that contains the extracted recommendations in the hopes that the MSR community can engage in a continuing discussion on our evolving best practices.**

I. Introduction

Mining Software Repositories (MSR) research focuses on analyzing various data sources that describe software systems and their development in order to provide actionable recommendations [27]. Since 2004, the MSR conference and workshop series has been a primary venue for publishing such studies, and the success of MSR has continued to attract new researchers, ideas, and applications. This year (2013) marks a milestone in MSR history as we come to the end of a decade of research in this area. In this paper, we take this opportunity to reflect on the territory we have covered, and some of the research best practices we have developed as a community.

Performing MSR research is often challenging as it involves dealing with a myriad of different development artifacts, tools, and analysis techniques, some of which have been adapted from other research areas of computer science. The broad scope and evolving technical landscape can be daunting not only to researchers who are new to the field, but also to experienced researchers who may have to work with new types of data or analysis techniques. In practice, guidelines for MSR research have often been communicated verbally during many of the community gatherings (e.g., PASED [2]) or are learned at the cost of blood, sweat, and tears. To help new researchers enter the MSR community, and to reflect on the experience of existing researchers, we codify a set of guidelines, tips, and recommendations based on previous MSR successes. In this paper, we aim to provide these set of best practices and recommendations in what we call *the MSR cookbook*.

To create this cookbook, we analyze the past decade of MSR publications; we summarize the experiences and accomplishments of research in this field, so that new researchers may learn from the past and avoid "re-inventing the wheel". In this paper, our goal is to be as unbiased as possible by developing the cookbook from recommendations extracted from the entire MSR research community, rather than a single researcher or group's ideas. To accomplish this, we review all 117 full papers published in MSR conferences and workshops between 2004–2012. During this process, we extract 268 *comments* from the papers. Comments describe information directly provided by the authors of the paper about a particular topic. The comments are subsequently categorized using an open coding approach [42], and are then grouped into four high-level themes that can be coincidentally mapped into the typical MSR process: data acquisition and preparation, synthesis, analysis, and sharing and replication. Finally, the top recommendations (i.e., those suggested in several papers) from each category are reported and discussed.

Recommendations in the *Data Acquisition and Preparation* theme (seven out of 16 recommendations) are mostly related to how and what to extract from software development artifacts including source code, source control management systems (SCMs), issue tracking systems, mailing lists, and textual data, in addition to how this data should be preprocessed before it can be used. The *Synthesis* theme includes four key recommendations regarding collinearity, text mining, clustering, classification, and prediction techniques. The *Analysis* theme contains four recommendations on performing statistical analysis and evaluation of the synthesis approaches. Finally, the *Sharing and Replication* theme only contains one recommendation for improving the reproducibility of the MSR studies.

Surveying past work in a research area to extract the state of the art is not something new. Such surveys typically provide a summary of what has been accompanied with some quantitative analysis of the different topics and trend analyses. Our work here takes a different approach by creating a working cookbook that can be continually used and updated as the MSR community matures and advances. To this end, we also provide an online version of the cookbook as a forum [1]. Our hope is that the MSR community continually contributes to the

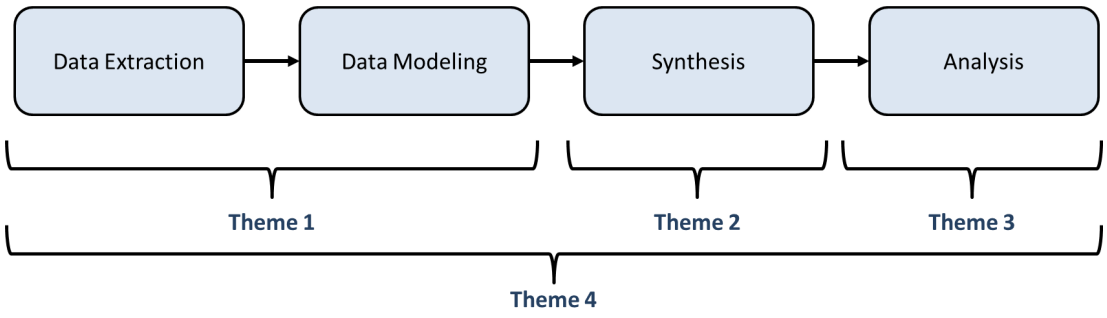

Fig. 1. Steps of a typical MSR process.

Cookbook forum to make it an up-to-date collaborative learning environment for all researchers in the field. We also provide the raw extracted data along with the original classifications [3], for the sake of reproducibility.

The rest of this paper is organized as follows. Section II describes our review process, and how we extract the data from the reviewed papers. We then explain the 16 recommendations in Sections III-VI according to the four themes we extracted. We describe the evolution of these themes in Section VII. Section VIII then provides some quantitative analysis of the tools used or developed by the MSR community. The Cookbook forum is then described in Section IX and related work is discussed in Section X. Finally, Section XI discusses limitations of this study and Section XII concludes the paper.

II. Our Review Process

Our ultimate goal is to develop a list of recommendations based on best practices of the MSR community that can help newcomers conduct rigorous research more effectively. In this section, we first describe the raw data we obtained from MSR proceedings, and how a preliminary set of recommendations were extracted from it. We then discuss our heuristics for selecting the recommendations that appear in our final list. Finally, we describe our categorization process.

A. Data Description

Since its inception in 2004, the MSR conference and workshop series has published 270 papers, including 117 full papers, 26 position papers, 67 short/poster/lightning papers, and 60 challenge papers.[1] Four of the authors of this paper carefully read all 117 full papers from MSR 2005 to 2012, and extracted a total of 268 comments from them; comments were selected if they appeared to be generalizable observations or recommendations about doing work in the field, and if they appeared to be supported by evidence in the paper. We also extracted the names of any tools that were developed or used in the studies.

[1]To better foster discussions, only shorter papers were solicited for the first year of the conference; since we decided to study only full length research papers, we did not analyze any papers from MSR 2004.

B. Criteria for Selecting Recommendations

As mentioned above, *we decided to consider only full papers from the MSR canon* in our analysis; limiting ourselves to full papers allows us to focus on more established work, likely with more convincing evidence. Also, while there is much work in the field published at other venues, *we decided study only papers from the MSR conference series proceedings*; this defines a simple and obvious scope for the study, and at a tractable size.

Due to the large number of comments extracted from our review process, we devised a further heuristic to aid in deciding on recommendations were most important: *A key recommendation should be supported by evidence from at least five papers.* This heuristic is based on the assumption that if more papers are providing a specific recommendation then it is likely to be stronger and more useful to the research community. Extracting unbiased recommendations from the papers can be challenging. While each paper may offer several contributions to the MSR community, our approach focuses on extracting generalizable sound recommendations from it.

C. Open Coding Approach

We now move to categorizing the extracted recommendations. Two authors of this paper who were not heavily involved in the extraction process performed two sessions of *card sorting*, a grounded theory approach [42]. This analysis design was chosen to make the categorization and extraction as independent from each other as possible.

Our open card sort revealed 28 categories. We then formed high-level themes by merging related categories. Perhaps unsurprisingly, we found that the themes we formed coincided with the steps of a typical MSR process: 1) *Data acquisition and preparation*, 2) *Synthesis*, 3) *Analysis*, and 4) *Sharing and replication*. Figure 1 shows the steps of a typical MSR process. It starts with extracting raw data from several kinds of development artifact repositories, such as source code, SCM, bug tracking issues, mailing lists, etc. This is usually followed by a data modelling phase where some preparation of the raw data and linking data from different repositories may be required. The next step is applying a mining/learning technique such as clustering or regression. Finally, an analysis and interpretation of the results is required to make the conclusions.

Our first theme relates to the first two steps in Figure 1, while the second and third themes are a strong match for the third and fourth steps of MSR process, respectively. The last theme does not directly match with any steps in this process. However, it relates to the entire process when viewed as a whole since MSR researchers are encouraged to share their data and results.

D. Representation of Results

In the next four sections (Sections III-VI), we discuss the most common recommendations — those that have five or more supporting papers — from each theme, supported by examples from the original papers. For each theme, we discuss the comments we extracted for the top recommendations. For each recommendation, we indicate its theme by a letter: *D* for Data acquisition and preparation, *S* for Synthesis, *A* for Analysis, and *R* for sharing and Replication. We also created a synthetic quote that succinctly described the recommendation; we report the synthetic quote as well as the number of comments and papers for each recommendation in box format after each discussion. It is important to note that the comment and paper count is for the entire recommendation and any given paper might not map exactly to the synthetic quote.

III. Theme 1: Data acquisition and preparation

This theme gains the most attention from researchers. We extracted 180 related comments, which is 67% of the total extracted comments (268). In the rest of this section, we report the most common recommendations. Some concern only a single, specific repository kind — such as source code, issue tracking, or communication artifacts — while others are more general and address themes or techniques — such as processing text and modelling developers — that may apply to several repository kinds.

A. Source Code

Source code is king; it is the most important of all software development artifacts. Researchers mine source code repositories for different purposes such as source code evolution analysis [48], code search, fault prediction, etc. Source code is usually stored in a Source Control Management (SCM) system. Researchers first need to extract code artifacts from such SCMs. We extract 22 comments relating to extracting data from SCMs. The high number of related comments was expected since SCMs are one of the most mined repositories in the field. The comments range from tips on how to extract data from a specific SCM system such as CVS (e.g., using "sliding window" and the commit time for grouping CVS commits [18], [41]) and GIT (e.g., observing that "implicit" branches can create ordering confusion [12]) to general points that can be applied to all SCMs.

One common observation which is applicable to all SCMs is the importance of understanding the project, its domain, and the underlying development process when extracting commit data, and being wary when making general assumptions. For example, the person who commits a change is not necessarily the one who created it [5], or the fact that not all change-sets imply strong interdependences among the involved files [57]. In addition, determining the true time of a change — i.e., when it was made — is often subtle and difficult, and the assumption that "commit time is the same as change time" is often wrong [28], [51]. Another common mistake is assuming that all co-changes are symmetric. In reality, if `X` and `Y` are changed together, changing `X` may imply changing `Y` but not vice versa. Therefore, understanding the sequence or ordering of changed files is important [33]. Finally, one should be careful that commit comments are sometimes used for communication rather than the description of the change. Additionally, many projects have an initial period of transition in their SCM system, which may generate noise in the data [15].

D1: *SCM repositories contain a variety of noise; validation of heuristics and assumptions is essential.*
[22 comments from 17 papers]

Source code analysis often requires matching two pieces of code in the data extraction and preparation phase. The required precision of the matching algorithm may be loose (e.g., type 3 code clone detection) or tight (e.g., looking for a specific library version in a repository [20]), depending on the application. For simple cases, this can easily be done through the Unix `diff` command. However, for sophisticated analyses, researchers often suggest extracting Abstract Syntax Trees (ASTs) [45] or Control Flow Graphs (CFGs) instead [36]. Matching techniques based on AST or CFG produce matches at fine-grained levels but are applicable only to complete and parsable programs; additionally, such fine-grained analysis is computationally expensive, and hard to scale up. Therefore researchers must carefully consider the trade-offs between the desired granularity, the required application, and the cost and complexity of the extraction process [36].

D2: *The choice of code extraction approach (simple `diff`, building AST or CFG, etc.) is often heavily influenced by analysis cost.*
[10 comments from 8 papers]

B. Issues and Patches

Bug prediction was one of the earliest research topics to be explored widely and in depth in MSR. However, surprisingly, we found relatively few explicit comments on mining bug and patch data; one possible explanation for this is that the topic and practices are well understood already, requiring relatively little advice for newcomers. A few comments (6) that we found interesting are again related to stating incorrect assumptions during data extraction. For example, several papers mentioned the importance of distinguishing between committers and submitters of the patch [61]. When determining the person who fixes an issue, researchers need to be aware that the `ASSIGNED TO` field in issue tracking systems might be set to a default email address [5]. When mining a bug repository, one should first understand how the system works. For example, in Bugzilla some fields such as

bug `Report time` or `History-related time` do not necessarily correspond to the actual time [64]. In addition, it is crucial for reproducibility to clearly define your heuristics and assumptions when describing your work. For example, when detecting if a patch has been accepted, what would you do if the directory of the patch was changed or renamed [61]? Different studies may define different heuristics, which may in turn affect the obtained results.

Another comment related to issues and patches suggests to carefully define and extract the data set you are planning to work with. For example, when predicting bug resolution time or when selecting bugs for validation from a bug repository, one should consider only issues whose resolution is set to `FIXED` or `CLOSED`, as this avoids issues that are under development or are duplicates [60], [43].

D3: *When working with issue trackers, it is often best to only consider closed and fixed issues.*
[6 comments from 5 papers]

C. Communication Artifacts

Developers often discuss different topics on mailing lists, and may belong to several social networks. This provides an opportunity for MSR researchers to mine social characteristics of developers and relate them to the characteristics of the software products and processes. Extracting such characteristics is a difficult task, and there are several comments from the papers on how and what to mine from these repositories. For instance, preprocessing email data (e.g., removing attachments, duplicate messages, and quoted replies, converting HTML emails to plain text, and extracting email header information) is suggested by several papers [31], [50]. It is also recommended that researchers study only emails with replies, since there is no guarantee that an email with no reply was actually read by any developers [46]. In addition, to avoid merging messages from two separate discussions that happen to have the same subject, using a sliding time window is recommended [31].

D4: *Social communication data requires extensive preprocessing before it can be used.*
[10 comments from 8 papers]

D. Text Mining

A large portion of software data is textual, and exploring such data requires significant preprocessing together with specialized analysis techniques usually borrowed from Natural Language Processing (NLP) research.

When dealing with mining textual data, it is often suggested to remove stop-words [40], [60], punctuation, and meaningless keywords. Additionally, it is advised to split compound words [49], [40], [63] and perform word stemming [8], [38]. For example, if your data set contains source code, characters related to the syntax of the programming language and non-alphabetic symbols (numbers, punctuation, special characters, etc.) are to be removed. Programming language keywords such as `IF` and `WHILE` are also to be removed, and identifier names need to be split into multiple parts based on common naming conventions. Removal of common English-language stop words and applying word stemming are highly recommended to remove noise in data. Finally, in some cases, the vocabulary of the resulting corpus is further pruned by removing words that occur in over 80% of the documents or under 20% of the documents [59], [62]. Alternatively, log weight functions are sometimes used to lessen the effect of a single term being repeated often in a corpus [24]. Depending on the context, you may then need to create various indices: a standard term index including all terms, a stop-index which excludes stop words, a stem-index with stemming applied, and a synonym-index [24] that aggregates synonyms for semantic analysis [24], [39]. Depending on the technique used and the analysis applied, normalization and smoothing may also be applied [49], [57].

D5: *Plain text requires splitting, stemming, normalization and smoothing before analysis.*
[24 comments from 16 papers]

In the context of text mining, one often needs to match words (e.g., for mapping identifiers of two versions of a file [22]). To reduce the number of false positives, researchers suggest different approaches. For example, using a normalized Levenshtein distance and an identifier type checking [49], or using a Vector Space Model with a Cosine similarity function [16]. Before looking for similar words (pairwise comparison), one could also put all documents into clusters, where they have at least one common word, and then perform a search within these clusters [63]. And no matter which search technique is used, one should always report possible bias (e.g., the choice of keywords when filtering bug reports) [6] and manually validate a subset of the outcome [19].

D6: *Text analyses should be manually verified (e.g., sampled) in addition to regular bias reporting.*
[9 comments from 7 papers]

E. Modelling Developers

One important aspect of many MSR studies — with 13 comments from 9 papers — is studying developers and their behaviour. This is usually accompanied by establishing the impact of their practices on code style and quality.

The first step in this type of analysis is identifying individual developers by mapping them to actual people. Before performing any analysis on the data set, it is necessary to merge multiple online identities that correspond to the same person [16], [53], [64]. This can be done, for example, by performing account aliasing when determining individual developers making commits to the source code repository [11] or by merging multiple email addresses that belong to the same person [9], [31]. Aliasing is even more challenging when people are in more than one repository under study [53]. For example, a user might have different authentication credentials for code and bug repositories [58], [64]. Manual validation is necessary for confirming such aliasing [10].

In some situations, an MSR researcher requires more detailed knowledge about developers than just knowing their identity;

for example, one might wish to know what organization the developers work in and where they are geographically located. Combining data from multiple repositories (e.g., email domain address, social networking sites, blogs, presentations, company sites, web articles, and SCM history [11]) is required for such analysis since relying on one source of data may lead to inaccurate results [54]. In addition, when locating a developer who contributes code modifications, the fact that the developer works for a particular organization does *not* mean that the organization itself is a formal contributor [11].

MSR researchers often try to model developers based on their experience and expertise. This cannot be accurately derived solely from expertise information in the developers' profiles, because they are rarely updated [43]. Their experience on a project can perhaps more accurately be defined as the time that has passed since their first commit to the project [23]. In addition, their expertise on a context, e.g., a bug report, can be modelled as the lexical similarity of their vocabulary (source code contributions) to the context, e.g., the content of the bug reports [40]. However, one should be aware that making commits to certain types of files does not necessarily mean that the developer has expertise in the corresponding area, since the commits could be deferred from other developers [34]. Furthermore, one needs to account for developers who made a few changes at one point in time, and are no longer active [43].

D7: *Multiple online personas can cause individuals to be represented as multiple people.*
[13 comments from 9 papers]

IV. Theme 2: Synthesis

This theme summarizes the recommendations around the synthesis theme of an MSR research process (33 comments). The synthesis usually involves clustering, classification, prediction, or other machine learning algorithms that is applied on the "cleaned-up" data that is the output of the data acquisition and preparation phase.

The first step, which must be done with caution, is designing a proper synthesis model. For example, one popular comment about designing prediction models concerns removing collinearity among the predictors [10], [55], [11]. Another important comment is including time decay into the prediction models (e.g., old modifications to a file and old bugs are likely to have no effect on new futures) [19], [57]. In addition, when building a regression model, one must ensure that the independent and control variables are not skewed. If they are, one should perform a logarithm transformation on these variables before using them in the regression model [11].

S1: *Be careful to remove collinearities and deal with skewness when synthesizing models from data.*
[22 comments from 17 papers]

Another design related common recommendation concerns parameter settings. Many synthesis techniques have several parameters that need to be set. For example, to find the optimal number of topics in topic modelling synthesis technique, researchers have suggested a fixed number, a range, or a number based on the input data size, but they also suggest sensitivity analysis [17], [29], [59].

S2: *Carefully tuning parameters and performing sensitivity analysis can improve the modelling process.*
[6 comments from 5 papers]

As expected, many papers suggest using different machine learning algorithms that they have found effective in their experiments and contexts. However, when choosing a classifier, it should be noted that complex ones are not necessarily better; one should experiment with several classifiers for a given problem and context. Sometime the readability and interpretability of the approach becomes more important since the difference in performance of different complex classifiers is minor [25]. For example, the simple Decision Tree classifier is suggested as a machine learning algorithm as it explicitly shows the major factors that affect a developer's decision to contribute, while other machine learning approaches produce black box models that do not explain their classification decisions [31].

S3: *Simple analyses often outperform their complex counterparts.*
[6 comments from 5 papers]

Regression analysis is one of the most famous prediction models that are used in the synthesis phase of MSR research. When applying a regression model, one should be aware of its assumptions, namely: 1) constant variance, 2) zero mean error, and 3) independence of independent parameters. When using regression strictly for descriptive and prediction purposes, these assumptions are less important since the regression will still result in an unbiased estimate between the dependent and independent variables. However, when inference based on the obtained regression models is made (e.g., conclusions about the slope or the significance of the entire model itself), these assumptions must be verified [25].

S4: *Be aware of the assumptions made by regression analyses.*
[10 comments from 8 papers]

V. Theme 3: Analysis

Analyzing the results of the synthesis phase and interpreting them (with 40 comments) may be seen as the most important phase of the MSR research, since the research conclusions are the direct output of this phase. Using manual verification, rigorous statistics, and visualizations are key to success.

Automated approaches and heuristics can go wrong. If we base our entire research on such heuristics, the outcome might be very inaccurate and misleading. MSR researchers suggest manually inspecting the outcome of automated algorithms. For example, a keyword-based approach for bug report identification [6] and clustering of bug types [65] requires manual verification of the outcome. Otherwise, if the keywords or the search are not suitable for a project, the entire analysis and conclusions are doubtful. Mapping different repositories, also,

requires manual inspection, since it is mostly done through the use of heuristics [9], [11], [56].

However, manual inspection of every single output is not feasible. Therefore, sampling and verification approaches are suggested by researchers [30]. For example, one can validate a mapping or classification algorithm by reporting its precision and recall on a randomly sampled data set that is manually verified [23]. The best way of manual verification is performing it by the actual user (e.g., developers), who are expert in the field [24], [40]. If this is not possible, then at least somebody outside the research group should perform the manual verification [14]. It is also suggested that rigorous inspection and testing is applied to the source code and scripts that implement the analysis approach since wrong output might not necessarily be due to poor heuristics, but simply due to defective implementations [44]. Finally, researchers suggest normalization in many scenarios. For example, when studying some phenomena concerning browsers or operating systems, you may normalize the data per browser/operating system by its average market share [7].

A1: *Manual verification of all outputs is required.*
[20 comments from 16 papers]

Before using sophisticated evaluation approaches and statistical tests, there are many cases where one can simply look at the correlations between the output values and the best (or actual) values. For example, one can examine the correlation between the number of predicted bugs per file vs. the actual number of bugs per file. A correlation function that does not make any assumptions about the distribution of the data is usually suggested. For example, Spearman rank correlation may be preferred over Pearson correlation when performing rank correlation.

A2: *Correlation analysis can be used to quickly check initial hypotheses about a set of data.*
[12 comments from 9 papers]

Prediction models and classifiers are among the most used synthesis techniques in MSR. Therefore, researchers often need to evaluate the effectiveness of these techniques in their studies. Reporting precision and recall is probably the most common approach in this type of research, which is also suggested by many MSR papers [23], [38]. However, there are also other ways that may provide better performance indicators or compliments in some contexts. For example, reporting the "average absolute residual" (the difference between the predicted and the actual effort for a task) in the context of effort prediction [60]. Sometimes, a full confusion matrix is more informative than reporting only the precision and recall [47]. Area under curve (AUC) is also a robust measure to assess and compare the performance of classifiers [26]. For example, using "accuracy" as a measure of prediction is problematic in heavily skewed distributions since it does not relate the prediction to the probability distribution of the classes, but AUC is insensitive to probability distributions [21].

A3: *While precision and recall are valuable measures, be aware that other indicators may be more appropriate in a given context.*
[12 comments from 10 papers]

In many situations, a proposed approach (e.g., a classifier) may be superior to the baselines for some input data but not for others. In addition, an MSR technique may be randomized in the nature that results in potentially different outcome per run of the technique on the same input data. Therefore, when comparing various techniques or data groups, it is essential to report statistical significance of the differences [17], [64]. There are many tests that are suggested for different scenarios. The most suggested case is using non-parametric tests when the normality of the data is unknown [13], [35]. In addition, when the data is skewed, researchers suggest reporting other stats (e.g., median) than the average (mean) of the data, which can be very misleading [26]. In many situation (e.g., when working with extracted metrics), one needs to deal with missing data, especially when applying statistical tests [54]. Researchers suggested different techniques, such as list-wise removal or mean data replacement [55], that can be applicable in different situations.

Looking at statistical significance is necessary to show that the difference between two techniques or two data sets is not due to chance. However, the statistical significance does not say anything about how much the better technique is improving the baseline. In other words, the practical difference is unknown. To report the practical difference, researchers suggest reporting the effect size, e.g., a Cohen's d effect size measure [16]. The more common recommendation is simply visualizing the results. Researchers use several types of plots to visualize the practical differences between the tested techniques. In addition, visualization is a useful tool when the study is not simply comparing the effectiveness of two approaches. For example, when studying software evolution [33] or developers' behaviour [23], visualizing SCMs, bug tracking systems, and mailing list data may be more helpful.

A4: *While applying rigorous statistical analyses is important, consider practical differences when drawing conclusions.*
[16 comments from 15 papers]

VI. Theme 4: Sharing and Replication

The final — and often ignored — phase (with only 15 comments) of rigorous MSR research is sharing the data and tools to allow for external validation and replication studies. By nature, most MSR research consists of empirical studies that report on applying some techniques on certain data sources. Such a single study, alone, will not provide convincing evidence that the results are generalizable to any other, even similar, data sources.

Researchers suggest making replication studies more common in MSR by providing as much as information about the described analysis as possible. For example, it is suggested that the MSR community should have a SourceForge-like

Year	Theme 1	Theme 2	Theme 3	Theme 4	Total per year
2005	8	0	0	1	9
2006	14	1	4	0	19
2007	23	5	3	0	31
2008	10	2	1	2	15
2009	28	1	4	1	34
2010	27	7	10	7	51
2011	45	9	8	3	65
2012	25	8	10	1	44
Total	180	33	40	15	268

TABLE I
NUMBER OF COMMENTS EXTRACTED FROM MSR FULL PAPERS OVER THE EIGHT YEARS EXAMINED, CATEGORIZED BY THEME.

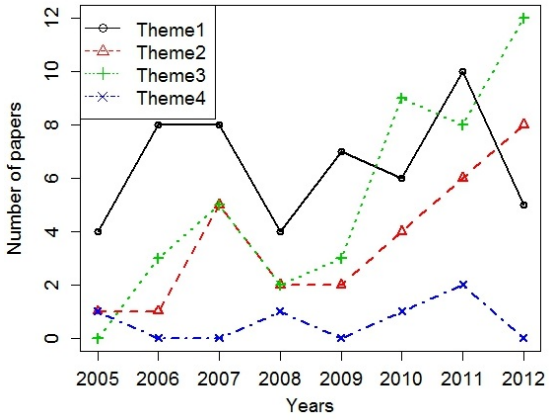

Fig. 2. Evolution of themes. *Theme 1*: Data Acquisition and Preparation, *Theme 2*: Synthesis, *Theme 3*: Analysis, *Theme 4*: Sharing and Replication.

website that keeps track of each paper's data and tools to avoid broken URLs since researchers may not regularly maintain their websites [52]. Regarding data, it is suggested that one should provide both raw and processed data since replication may be needed on any of the two sources [52]. Even if research tools are made available for external use, in practice they can be immature, buggy, hard-to-install, and hard-to-learn for external users. In the early stages of MSR, researchers tend to develop more in-house tools even for basic tasks, but nowadays the goal is using more standard tools for basic tasks, at least. In section VIII, we report the tools that are provided or used in the MSR community and provide more discussion around this issue.

R1. *Sharing data, tools, and techniques helps push the community forward and is just Good Science™.*
[11 comments from 5 papers]

VII. THE EVOLUTION OF RECOMMENDATIONS

Table I shows the number of comments extracted for each theme per year. The most visible observation is the dominance of *Theme 1* (Data Acquisition and Preparation) comments when compared to other themes. However, if we look at their evolution, we can see that the relative percentage of the comments in Theme 1 has steadily dropped, from 8/9=89% of the entire comments in 2005 to only 25/44=57% in 2012.

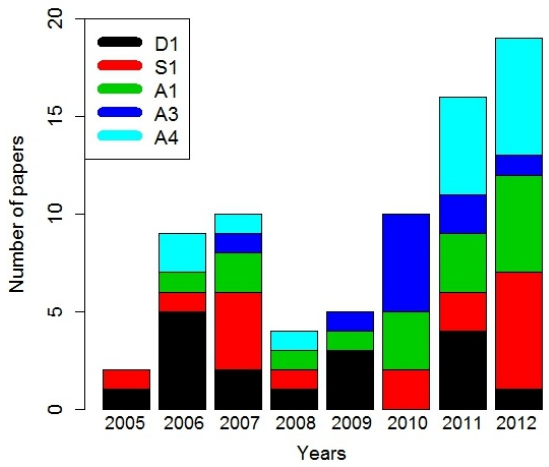

Fig. 3. Evolution of top recommendations. Top recommendations are those with at least five supporting papers in a singe year.

In contrast, *Theme 2* (Synthesis) and *Theme 3* (Analysis) are receiving more attention in recent years. In Figure 2, we study the same evolution but with respect to the the number of papers suggesting a recommendation from a theme, rather than the comment-level analysis in Table I. Interestingly, we can see that the increase in *Theme 2* (Synthesis) and *Theme 3* (Analysis) is more apparent when we look at distinct papers. For the first time, in 2012, the number of papers providing suggestions related to *Theme 1* (Data Acquisition and Preparation) drops to only five papers, which is fewer than the number of papers providing comments related to *Theme 2* (8) and *Theme 3* (12). This suggests that the extraction and preparation of data, which was a primary challenge in the past, is perhaps becoming less of a roadblock as researchers have developed mature tools and techniques for these tasks. This allows researchers to focus their energy on more interesting synthesis and analysis techniques and approaches. Finally, we note that *Theme 4* (Sharing and Replication) has always been under-represented relative to the others.

To get more insight on the distribution and evolution of recommendations over the years, we look at the top recommendations from our list more carefully in Figure 3. The top recommendations are defined as those with at least five distinct supporting papers that occur in the same year. These top recommendations are: 1) *D1*: SCM repositories contain a variety of noise; validation of heuristics and assumptions is essential, 2) *S1*: Be careful to remove collinearities and deal with skewness when synthesizing models from data), *A1*: Manual verification of all outputs is required, 3) *A3*: While precision and recall are valuable measures, be aware that other indicators may be more appropriate in a given context, and 4) *A4*: While applying rigorous statistical analyses is important, consider practical differences when drawing conclusions.

The figure shows the distribution of the number of supporting papers per each of the top five recommendations, over the years. The first observation is the dominance of the *Theme 2* (Analysis), with three recommendations among top five.

Theme	Groups	# Distinct Tools	# Papers
Data Acquisition and Preparation	Source Code Analysis	21	25
	Cloning	2	6
	Text XML Data	2	2
	Bug Repositories	1	1
	Mail Repositories	4	4
	Code Repositories	13	16
	Linking Diff. Repositories	8	9
	Bug Analysis	1	1
Synthesis	Recommenders/Decision Trees	6	9
	Classification	4	6
Analysis	Statistical Analysis & Visualization	5	9
Sharing	–	1	1
Total		68	89

TABLE II
SUMMARY OF TOOLS USED IN EACH THEME.

Looking at each recommendation, we realize the constant increase in the number of papers related to A1 and A4, both of which are related to the fact that we need stronger evidence to support MSR research by making sure the automated techniques are valid and the improvements are practically significant.

In summary, both theme-level and recommendation-level trend analysis suggests that MSR research is maturing as a field; it appears to be shifting from considering only data extraction with limited automated analysis toward deeper analysis of the results and seeking practical use.

VIII. The tools of MSR

To improve the practicality of our cookbook, we extracted the names and descriptions of the tools used by researchers from the canon of MSR papers we studied. We found a mix of "homebrew" tools specially created and tuned for the research at hand together with well known tools from other areas of computer science that were adapted for use within an MSR context. We believe reporting and summarizing this technical dimension of the work can complement the top recommendations list and help newcomers make used of the collective experiences of the MSR community.

Tools, however, can be hard to extract and categorize. Researchers do not always report all the tools they have used and/or developed. For example, we know from experience that the GNU R language and environment is very widely used by the community, yet only some of the papers (5) explicitly mention the use of this tool. Table II summarizes the tools mentioned in the 117 full papers. The number of distinct tools used in each group is reported in addition to the number of papers that use such a tool. A total of 68 tools are reported in the table, with 66 of them being distinct. This is as a result of the tool Weka appearing in both the "recommenders/decision trees" group as well as the "classification" group. The majority of tools are developed to support tasks associated with data acquisition and preparation. Among these tools, 21 of them aim to assist with source code extraction. The full version of the table which includes the name of the tool, as well as the reference to the original paper is available online [3].

IX. The MSR Cookbook: An online forum for MSR recommendations

Surveying published literature is an effective approach for extracting recommendations based on best practices. However, it is limited to a static document (the survey article), which becomes outdated over time. Surveys are also subjective: even if the data collection process is systematic and the criteria are well documented, the choices are still made by the authors based on their expertise, preferences, and even biases. To help overcome these limitations, we provide an online version of the recommendations in the format of a forum: the MSR Cookbook [1]. The forum is initially populated with the top recommendation list provided in this paper, and is open to the public. Researchers can not only contribute by commenting on the existing posts, but may also add new recommendations based on their own findings. The ultimate goal is having the forum as a live version of this paper, where people can discuss recommendations to reduce any potential biases and provide a collaborative learning environment.

X. Related Work

There are several articles that provide guidelines in software engineering, some of which are applicable to a range of research studies. For example, Kitchenham et al. [37] provide a set of guidelines on empirical research in software engineering. Although these types of guidelines are very useful, they are often too generic to be immediately useful and may require tailoring to a specific context. For example, Ali et al. [4] have provided a systematic review and a set of guidelines inspired from generic guidelines and best practices in the specific field of search-based test case generation.

Another type of guideline is purely driven by authors' expertise. For example, in the field of MSR, Bird et al. [12] propose a set of guidelines on using the `GIT` source code management system based on their experience. In a broader sense, researchers sometimes provide the guidelines in the format of a road ahead [27] based on their personal experience and studying other literature. In a slightly different setting, Robles [52], provided a set of guidelines on the replicability of the MSR papers, which is driven from a survey of MSR paper (2004-2009).

Surveys and reviews are also popular but are mainly used for categorization and trend analysis. For example, Zannier et al. [66] have done a trend analysis by quantitative analysis of empirical studies in the International Conference on Software Engineering (ICSE). In the context of MSR, Kagdi et al. [32] have published a survey of all MSR related research before 2006. In this paper, however, we take a different path. We are more interested in providing guidelines and best practices but with the approach of surveying, which is less subjective than the common form of reporting own experience. Therefore, we review published MSR full papers and derive our top recommendations based on such raw data, which basically formed our cookbook.

XI. Threats to Validity

The main threats to the validity of this study arise from the nature of our survey where we go beyond counting papers and tools, seeking to provide a set of best practices. There is some subjectivity that goes into such a process. However, to avoid any bias arising from that, we document the criteria we use to select the recommendations in Section II. We also reduce potential bias in categorization by separating the data extraction team from the card sort team. Finally, we provide the MSR Cookbook forum as an interface for improving the top recommendation list by the community itself. Since any survey runs the risk of not properly reporting the original authors' point of view, having an online forum allows these authors to easily point out such issues and correct any misinterpreted information. We also share our raw data (extracted comments and tools) to provide a means for replication and verification.

XII. Conclusion

After a decade of the MSR conference and workshop series, we believe that taking a moment to reflect on the best practices of our community can help both newcomers and experienced researchers alike. We have extracted 268 comments from the 117 full papers published in MSR since 2004. Applying an open card sort for categorization of the comments resulted in four themes of recommendations, namely: *data acquisition and preparation*, *synthesis*, *analysis*, and *sharing/replication*. In this paper we reported the most common recommendations among these 117 papers. The evolution analysis of the recommendations showed that although data acquisition and preparation is clearly an important part of most MSR projects, their prevalence is decreasing while recommendations about synthesis and analysis are increasing. We consider that this is a sign of the relative maturity of the data acquisition and preparation, which has shifted the attention of researchers toward more thorough analysis and interpretations. Finally, we provide an online version of this cookbook as a forum to make the MSR community engaged and active on providing explicit recommendations, and contributing to the follow-up discussions.

References

[1] Msr cookbook forum. /urlswag.cs.uwaterloo.ca/ okononen/msr, year = 2013.

[2] Pased - canadian summer school on practical analyses of software engineering data. /url/http://pased.soccerlab.polymtl.ca/, 2011.

[3] Msr cookbook raw data. /urlswag.cs.uwaterloo.ca/ okononen/msr, 2013.

[4] Shaukat Ali, Lionel C. Briand, Hadi Hemmati, and Rajwinder Kaur Panesar-Walawege. A systematic review of the application and empirical investigation of search-based test case generation. *IEEE Trans. Software Eng.*, 36(6):742–762, 2010.

[5] John Anvik and Gail C. Murphy. Determining implementation expertise from bug reports. In *International Workshop on Mining Software Repositories (MSR)*, page 2, 2007.

[6] Cyrille Artho, Kuniyasu Suzaki, Roberto Di Cosmo, Ralf Treinen, and Stefano Zacchiroli. Why do software packages conflict? In *International Working Conference on Mining Software Repositories (MSR)*, pages 141–150, 2012.

[7] Olga Baysal, Reid Holmes, and Michael W. Godfrey. Mining usage data and development artifacts. In *International Working Conference on Mining Software Repositories (MSR)*, pages 98–107, 2012.

[8] Olga Baysal and Andrew J. Malton. Correlating social interactions to release history during software evolution. In *International Workshop on Mining Software Repositories (MSR)*, page 7, 2007.

[9] Christian Bird, Alex Gourley, Premkumar T. Devanbu, Michael Gertz, and Anand Swaminathan. Mining email social networks. In *International Workshop on Mining Software Repositories (MSR)*, pages 137–143, 2006.

[10] Christian Bird, Alex Gourley, Premkumar T. Devanbu, Anand Swaminathan, and Greta Hsu. Open borders? immigration in open source projects. In *International Workshop on Mining Software Repositories (MSR)*, page 6, 2007.

[11] Christian Bird and Nachiappan Nagappan. Who? where? what? examining distributed development in two large open source projects. In *International Working Conference on Mining Software Repositories (MSR)*, pages 237–246, 2012.

[12] Christian Bird, Peter C. Rigby, Earl T. Barr, David J. Hamilton, Daniel M. Germán, and Premkumar T. Devanbu. The promises and perils of mining git. In *International Working Conference on Mining Software Repositories (MSR)*, pages 1–10, 2009.

[13] Alexander W. J. Bradley and Gail C. Murphy. Supporting software history exploration. In *International Working Conference on Mining Software Repositories (MSR)*, pages 193–202, 2011.

[14] Marcel Bruch, Mira Mezini, and Martin Monperrus. Mining subclassing directives to improve framework reuse. In *International Working Conference on Mining Software Repositories (MSR)*, pages 141–150, 2010.

[15] Gerardo Canfora and Luigi Cerulo. Fine grained indexing of software repositories to support impact analysis. In *International Workshop on Mining Software Repositories (MSR)*, pages 105–111, 2006.

[16] Gerardo Canfora, Luigi Cerulo, Marta Cimitile, and Massimiliano Di Penta. Social interactions around cross-system bug fixings: the case of freebsd and openbsd. In *International Working Conference on Mining Software Repositories (MSR)*, pages 143–152, 2011.

[17] Tse-Hsun Chen, Stephen W. Thomas, Meiyappan Nagappan, and Ahmed E. Hassan. Explaining software defects using topic models. In *International Working Conference on Mining Software Repositories (MSR)*, pages 189–198, 2012.

[18] Marco D'Ambros, Michele Lanza, and Mircea Lungu. The evolution radar: visualizing integrated logical coupling information. In *International Workshop on Mining Software Repositories (MSR)*, pages 26–32, 2006.

[19] Marco D'Ambros, Michele Lanza, and Romain Robbes. An extensive comparison of bug prediction approaches. In *International Working Conference on Mining Software Repositories (MSR)*, pages 31–41, 2010.

[20] Julius Davies, Daniel M. German, Michael W. Godfrey, and Abram Hindle. Software Bertillonage: Finding the provenance of an entity. In *Proc. of the 8th Working Conference on Mining Software Repositories*, MSR '11, pages 183–192, New York, NY, USA, 2011. ACM.

[21] Jayalath Ekanayake, Jonas Tappolet, Harald Gall, and Abraham Bernstein. Tracking concept drift of software projects using defect prediction quality. In *International Working Conference on Mining Software Repositories (MSR)*, pages 51–60, 2009.

[22] Laleh Mousavi Eshkevari, Venera Arnaoudova, Massimiliano Di Penta, Rocco Oliveto, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. An exploratory study of identifier renamings. In *International Working Conference on Mining Software Repositories (MSR)*, pages 33–42, 2011.

[23] Jon Eyolfson, Lin Tan, and Patrick Lam. Do time of day and developer experience affect commit bugginess. In *International Working Conference on Mining Software Repositories (MSR)*, pages 153–162, 2011.

[24] Michael Gegick, Pete Rotella, and Tao Xie. Identifying security bug reports via text mining: An industrial case study. In *International Working Conference on Mining Software Repositories (MSR)*, pages 11–20, 2010.

[25] Emanuel Giger, Martin Pinzger, and Harald Gall. Comparing fine-grained source code changes and code churn for bug prediction. In *International Working Conference on Mining Software Repositories (MSR)*, pages 83–92, 2011.

[26] Emanuel Giger, Martin Pinzger, and Harald C. Gall. Can we predict types of code changes? an empirical analysis. In *International Working Conference on Mining Software Repositories (MSR)*, pages 217–226, 2012.

[27] Ahmed E. Hassan. The road ahead for mining software repositories (msr), 2008.

[28] Lile Hattori and Michele Lanza. Mining the history of synchronous changes to refine code ownership. In *International Working Conference on Mining Software Repositories (MSR)*, pages 141–150, 2009.

[29] Abram Hindle, Neil A. Ernst, Michael W. Godfrey, and John Mylopoulos. Automated topic naming to support cross-project analysis of software maintenance activities. In *International Working Conference on Mining Software Repositories (MSR)*, pages 163–172, 2011.

[30] Abram Hindle, Daniel M. Germán, and Richard C. Holt. What do large commits tell us?: a taxonomical study of large commits. In *International Working Conference on Mining Software Repositories (MSR)*, pages 99–108, 2008.

[31] Walid M. Ibrahim, Nicolas Bettenburg, Emad Shihab, Bram Adams, and Ahmed E. Hassan. Should i contribute to this discussion? In *International Working Conference on Mining Software Repositories (MSR)*, pages 181–190, 2010.

[32] Huzefa H. Kagdi, Michael L. Collard, and Jonathan I. Maletic. A survey and taxonomy of approaches for mining software repositories (msr) in the context of software evolution. *Journal of Software Maintenance*, 19(2):77–131, 2007.

[33] Huzefa H. Kagdi, Shehnaaz Yusuf, and Jonathan I. Maletic. Mining sequences of changed-files from version histories. In *International Workshop on Mining Software Repositories (MSR)*, pages 47–53, 2006.

[34] Siim Karus and Harald Gall. A study of language usage evolution in open source software. In *International Working Conference on Mining Software Repositories (MSR)*, pages 13–22, 2011.

[35] Foutse Khomh, Tejinder Dhaliwal, Ying Zou, and Bram Adams. Do faster releases improve software quality? an empirical case study of mozilla firefox. In *International Working Conference on Mining Software Repositories (MSR)*, pages 179–188, 2012.

[36] Miryung Kim and David Notkin. Program element matching for multi-version program analyses. In *International Workshop on Mining Software Repositories (MSR)*, pages 58–64, 2006.

[37] Barbara Kitchenham, Shari Lawrence Pfleeger, Lesley Pickard, Peter Jones, David C. Hoaglin, Khaled El Emam, and Jarrett Rosenberg. Preliminary guidelines for empirical research in software engineering. *IEEE Trans. Software Eng.*, 28(8):721–734, 2002.

[38] Ahmed Lamkanfi, Serge Demeyer, Emanuel Giger, and Bart Goethals. Predicting the severity of a reported bug. In *International Working Conference on Mining Software Repositories (MSR)*, pages 1–10, 2010.

[39] Walid Maalej and Hans-Jörg Happel. From work to word: How do software developers describe their work? In *International Working Conference on Mining Software Repositories (MSR)*, pages 121–130, 2009.

[40] Dominique Matter, Adrian Kuhn, and Oscar Nierstrasz. Assigning bug reports using a vocabulary-based expertise model of developers. In *International Working Conference on Mining Software Repositories (MSR)*, pages 131–140, 2009.

[41] Keir Mierle, Kevin Laven, Sam T. Roweis, and Greg Wilson. Mining student cvs repositories for performance indicators. In *International Workshop on Mining Software Repositories (MSR)*, 2005.

[42] M.B. Miles and A.M. Huberman. *Qualitative Data Analysis: An Expanded Sourcebook*. SAGE Publications, 1994.

[43] Shawn Minto and Gail C. Murphy. Recommending emergent teams. In *International Workshop on Mining Software Repositories (MSR)*, page 5, 2007.

[44] Shuuji Morisaki, Akito Monden, Tomoko Matsumura, Haruaki Tamada, and Ken ichi Matsumoto. Defect data analysis based on extended association rule mining. In *International Workshop on Mining Software Repositories (MSR)*, page 3, 2007.

[45] Iulian Neamtiu, Jeffrey S. Foster, and Michael W. Hicks. Understanding source code evolution using abstract syntax tree matching. In *International Workshop on Mining Software Repositories (MSR)*, 2005.

[46] Roozbeh Nia, Christian Bird, Premkumar T. Devanbu, and Vladimir Filkov. Validity of network analyses in open source projects. In *International Working Conference on Mining Software Repositories (MSR)*, pages 201–209, 2010.

[47] Ariadi Nugroho, Michel R. V. Chaudron, and Erik Arisholm. Assessing uml design metrics for predicting fault-prone classes in a java system. In *International Working Conference on Mining Software Repositories (MSR)*, pages 21–30, 2010.

[48] Chris Parnin and Carsten Görg. Improving change descriptions with change contexts. In *International Working Conference on Mining Software Repositories (MSR)*, pages 51–60, 2008.

[49] Shivani Rao and Avinash C. Kak. Retrieval from software libraries for bug localization: a comparative study of generic and composite text models. In *International Working Conference on Mining Software Repositories (MSR)*, pages 43–52, 2011.

[50] Peter C. Rigby and Ahmed E. Hassan. What can oss mailing lists tell us? a preliminary psychometric text analysis of the apache developer mailing list. In *International Workshop on Mining Software Repositories (MSR)*, page 23, 2007.

[51] Romain Robbes. Mining a change-based software repository. In *International Workshop on Mining Software Repositories (MSR)*, page 15, 2007.

[52] Gregorio Robles. Replicating international working conference on mining software repositories (msr): A study of the potential replicability of papers published in the mining software repositories (msr) proceedings. In *International Working Conference on Mining Software Repositories (MSR)*, pages 171–180, 2010.

[53] Gregorio Robles and Jesús M. González-Barahona. Developer identification methods for integrated data from various sources. In *International Workshop on Mining Software Repositories (MSR)*, 2005.

[54] Gregorio Robles and Jesús M. González-Barahona. Geographic location of developers at sourceforge. In *International Workshop on Mining Software Repositories (MSR)*, pages 144–150, 2006.

[55] Pete Rotella and Sunita Chulani. Analysis of customer satisfaction survey data. In *International Working Conference on Mining Software Repositories (MSR)*, pages 88–97, 2012.

[56] Vibha Singhal Sinha, Senthil Mani, and Saurabh Sinha. Entering the circle of trust: developer initiation as committers in open-source projects. In *International Working Conference on Mining Software Repositories (MSR)*, pages 133–142, 2011.

[57] Bunyamin Sisman and Avinash C. Kak. Incorporating version histories in information retrieval based bug localization. In *International Working Conference on Mining Software Repositories (MSR)*, pages 50–59, 2012.

[58] Jacek Sliwerski, Thomas Zimmermann, and Andreas Zeller. When do changes induce fixes? In *International Workshop on Mining Software Repositories (MSR)*, 2005.

[59] Stephen W. Thomas, Bram Adams, Ahmed E. Hassan, and Dorothea Blostein. Modeling the evolution of topics in source code histories. In *International Working Conference on Mining Software Repositories (MSR)*, pages 173–182, 2011.

[60] Cathrin Weiß, Rahul Premraj, Thomas Zimmermann, and Andreas Zeller. How long will it take to fix this bug? In *International Workshop on Mining Software Repositories (MSR)*, page 1, 2007.

[61] Peter Weißgerber, Daniel Neu, and Stephan Diehl. Small patches get in! In *International Working Conference on Mining Software Repositories (MSR)*, pages 67–76, 2008.

[62] Peter Weißgerber, Mathias Pohl, and Michael Burch. Visual data mining in software archives to detect how developers work together. In *International Workshop on Mining Software Repositories (MSR)*, page 9, 2007.

[63] Jinqiu Yang and Lin Tan. Inferring semantically related words from software context. In *International Working Conference on Mining Software Repositories (MSR)*, pages 161–170, 2012.

[64] Shahed Zaman, Bram Adams, and Ahmed E. Hassan. Security versus performance bugs: a case study on firefox. In *International Working Conference on Mining Software Repositories (MSR)*, pages 93–102, 2011.

[65] Shahed Zaman, Bram Adams, and Ahmed E. Hassan. A qualitative study on performance bugs. In *International Working Conference on Mining Software Repositories (MSR)*, pages 199–208, 2012.

[66] Carmen Zannier, Grigori Melnik, and Frank Maurer. On the success of empirical studies in the international conference on software engineering. In *ICSE*, pages 341–350, 2006.