Software Mining Studies: Goals, Approaches, Artifacts, and Replicability

Sven Amann[1], Stefanie Beyer[2], Katja Kevic[3], and Harald Gall[3]

[1] TU Darmstadt, Germany
[2] Alpen-Adria University Klagenfurt, Austria
[3] University of Zurich, Switzerland

Abstract. The mining of software archives has enabled new ways for increasing the productivity in software development: Analyzing software quality, mining project evolution, investigating change patterns and evolution trends, mining models for development processes, developing methods of integrating mined data from various historical sources, or analyzing natural language artifacts in software repositories, are examples of research topics. Software repositories include various data, ranging from source control systems, issue tracking systems, artifact repositories such as requirements, design and architectural documentation, to archived communication between project members. Practitioners and researchers have recognized the potential of mining these sources to support the maintenance of software, to improve their design or architecture, and to empirically validate development techniques or processes. We revisited software mining studies that were published in recent years in the top venues of software engineering, such as ICSE, ESEC/FSE, and MSR. In analyzing these software mining studies, we highlight different viewpoints: pursued goals, state-of-the-art approaches, mined artifacts, and study replicability. To analyze the mining artifacts, we (lexically) analyzed research papers of more than a decade. In terms of replicability we looked at existing work in the field in mining approaches, tools, and platforms. We address issues of replicability and reproducibility to shed light onto challenges for large-scale mining studies that would enable a stronger conclusion stability.

1 Motivation

Software archives, such as source control systems, defect tracking systems, or archived communication among project members, are used to help managing the progress of software projects. Since about a decade, the software engineering community exploits the potential benefit of mining this information to support the evolution of software systems, improve software design and reuse, and empirically validate novel ideas and techniques. Research has now proceeded to uncover the ways in which mining these archives can help to understand software development, to support predictions about software properties, and to plan software projects. Researchers regularly exchange their results and present novel

tools at conferences and symposia, such as MSR[4], MSA 2010,[5] ASDS 2013,[6] or MSR Vision 2020.[7]

Mining software archives (MSA) is one kind of software analytics that deals with investigating repositories that are used during software development to store all kinds of information about the software. Examples are version control systems, issue trackers, task management, project management, software forges (such as BitBucket or GitHub), Q&A sites (such as StackOverflow), or communication archives (such as emails, instant messages, or social-media data).

MSA has evolved from applying data mining to all kinds of data about a software system to a discipline of data-driven analysis that today is known as software analytics [69]. Software analytics is more than just data mining software versions. It is about obtaining insights into the actual development and evolution of software systems. These insights shall enable the observer to take actions in terms of changing practices, tooling, or infrastructures to improve productivity of software developers.

One example for such insights into evolutionary aspects of software is defect prediction, i.e. discovering code components (modules, classes, methods, etc.) that are likely defect-prone. Actions following from that can be redesign, refactoring, or even reengineering. Another example are so-called recommender systems that provide help for code completion, suggest good code examples, or support understanding code. Other examples are software effort prediction [112] or test-code impact analysis [127].

The major conference for researchers to publish their latest mining results is the Working Conference on Mining Software Repositories (MSR).[8] Analyzing the proceedings of past MSR conferences revealed that the mined artifacts have become manifold. In the first editions of the MSR conference, about ten years ago, only data from CVS repositories was investigated, whereas today researchers mine data from a broad range of resources, such as Git repositories, Q&A sites, blogs, emails, tutorials, and Twitter. The prominence of version repositories has declined over the years, but augments towards "social" artifacts, focusing more on the individual developer.

Kagdi et al. [49] surveyed the field and provide a good overview of the areas of MSR that cover dimensions of information sources, representation (type and granularity, as well as context), purpose of studies, methodology, and evaluation. Besides a comprehensive discussion of approaches and their classification, it is remarkable that in 2007 (looking back for about a decade) only few threats to (internal and external) validity were discussed in MSR studies. We argue that, with the manifold techniques and mined artifacts, the dimensions of reproducibility and replicability have not been addressed adequately. This opens new avenues for systematic mining studies.

[4] `http://msrconf.org`
[5] `http://www.ifi.uzh.ch/seal/events/msa2010.html`
[6] `http://www.ifi.uzh.ch/seal/events/ASDS-2013.html`
[7] `http://msrcanada.org/msrvision2020/`
[8] `MSRconf.org`

In this chapter, we address the question of systematic mining studies by looking at two aspects in particular: reproducibility and replicability. A special focus will be given to the latter with investigating approaches, techniques, and platforms published at the major conferences in the field in the recent past—in particular ICSE, ESEC/FSE, and MSR—that claim to deal with this challenge to some extent. We look into what makes a good mining study, then describe techniques for mining, and further study replicability and systematic mining studies. We also take a look onto how these challenges are taken up in recent fields, such as green mining, sentiment analysis, and studies covering human aspects.

2 Mining Studies

This section introduces two approaches where to place the field of mining software repositories into data analytics and data science. This is followed by an overview what characterizes a mining study. The main focus is on the setup for the study, the process, and the interpretation of the results. Then, we discuss the threats of mining software repositories, in particular for bug prediction on biased datasets, sample size, stable rankings, and time variance. Approaches, languages and tools that support researchers in mining software repositories, for instance, to share data, to improve reproducibility and to avoid redundant preprocessing of data are presented. Finally, we discuss the reproducibility of mining studies.

2.1 Meta-Studies on Mining Software Repositories

Mining software repositories evolved a lot in the last decade and the question about the relation of MSR to data science and software analytics needs to be addressed.

Mining Software Repositories and Data Science. Mockus [74] discussed in his keynote at the MSR conference in 2014 the relationship between mining software repositories, operational data, and data science. Mining software repositories focuses on extracting knowledge from software data. Both use operational data, so he concludes that mining software repositories actually is data science.

Mining software repositories and data science have similar goals, namely to identify laws by extracting knowledge from data. The data used for data science is often experimental data, such as temperatures of sensors. Accordingly, for mining software repositories mainly software data is analyzed.

Operational data are digital traces which are not primarily created for analysis, such as logs of mobile phones that are not created to be measured, but may be used for data science. However, the use of operational data brings along multiple challenges, such as missing data or even wrong data. Therefore, the challenge is to identify data laws to segment information, impute missing information and correct the data. Traditional data may be taken into account to fill the gaps of operational data. Tools that create operational data are, for instance, version

control systems, such as SVN or git, as well as issue-trackers, such as BugZilla or JIRA. Mockus argues that it is worth to research on operational data because there is so much data that also map the human activities to the digital domain. Furthermore, data is *treacherous* [74] and may have multiple contexts, missing data, and faked data.

The aims for mining software repositories and data science are similar: approaches or tools to engineer operational data with a software system to ensure integrity of results, to get more effectively results, and simplify the building of software systems to analyze operational data in order to increase the quality of the data. Therefore, operational data are common in both, data science and mining software repositories. The conclusion is that mining software repositories is indeed data science.

Software Analytics. Menzies and Zimmermann [69] describe the changing goals of software analytics over time, as well as the the different methods of data analytics, and the principles to perform a good study. They point out that the main goal is to give 'relevant advice' to the audience. Who should benefit from the outcome of the analysis? Analytics for testers and developers may require different tools and techniques than analytics for managers or even researchers. The claims of data analytics are to share information, or more concrete, to share models, insights, data, and methods. The main goal about 40 years ago was to find 'the model of software engineering.' By the time, this goal has changed, since one model cannot fit all software projects. So the focus shifted to find methods of a particular system that may be transferred to other systems. The most important factors for data analysis are the choice of the right usage patterns for the data, as well as the user itself who should profit from this analysis. The right choice of tools, for instance to visualize data or draw conclusions automatically from data might influence the data analysis positively. However, tools, algorithms, as well as a suitable hardware for analysis are not the key components.

The Seven Principles for Software Analytics by Menzies and Zimmermann give advice on how to perform data analytics. To apply data analysis effectively, the users' goals and needs must be well known and understood. If possible, early and continuous feedback of the users should be considered. The system built for data analysis should be able to repeat the analysis several times. The possibility of growing datasets should be taken into account. If the approaches did not work out it is often helpful to be open-minded for other directions. Evaluation of the data plays a very important role. One of the main evaluation principles is to repeat the analysis various times with different percentage of the data and see if the results change. Furthermore, preprocessing of data is often required and should not be neglected. The last principle guides to use a wide range of technologies. Tools that are constantly updated with the implementation of new methods make this easier.

The analysis of data has widespread goals and different focus. Therefore, Menzies and Zimmermann differentiate between several kinds of analytics. First,

they distinguish *internal* and *external analytics*. For internal analytics the access to data is easier than for external analytics that requires more effort, since the data typically has to be anonymized to keep privacy. Second, they distinguish the *quantitative* and the *qualitative* method. The former is applied automatically by using several data mining tools and statistics; the latter is mainly applied by investigating the data manually. Lastly, they point out the necessity to distinguish *exploratory* and *deployment analytics*. In exploratory analytics, the goals often are not clear and the research might not result in prominent findings. However, if results are found and the goals are clear, it is possible to use deployment analytics to build tools and systems to use the findings.

2.2 What is a Mining Study?

Let us explain a mining study first by looking at a recent experiment in the field of defect prediction [28]: We train models to predict defect-prone source files of the next release of a software system. For that we use product, process, and organizational measures and then apply machine-learning techniques to training the models. The result is a model (basically a set of coefficients for a function or set of functions) that fits the data best. This model can then be used for predicting the defect-prone modules of the next release. Typical models are regressions or decision trees; however a huge variety of machine learners can be used for such an experiment.

While the setup and the process of such a mining study might be clear, it is less clear what the essential ingredients of a *good* study are. We might view that it is all about the data; it is of course data-driven, but at the same time it is as much about the research hypothesis and the underlying assumptions of the experiment: Typically, the starting point is a research question that tries to relate (or correlate) some property X (e.g. code churn) with another property Y (e.g. defects), to check whether there exists a statistically significant correlation. Other setups might investigate how some property Z (e.g. code ownership) has developed over time, by looking at the version history of a software system. Phenomenons, such as code ownership or networks of developers, often are related to quality aspects of the software system, such as defect proneness, code irregularity, or complexity.

A mining study typically starts with a research hypothesis, then prepares the data to be investigated, analyzes the data, and continues to interpreting the results. The granularity of studies can vary quite a bit, ranging from factors influencing some particular property (such as buggy commits) to comprehensive quality measures (such as proper design or architecture).

With any such mining study, it is important to check whether the process of analysis, mining, and interpretation can be reproduced or replicated. *Reproducibility* means that the study description provides all the data, tooling, and configuration settings to validate the experiment. *Replicability* means that the study can be performed with different data sets and projects to gain a broader conclusion. Basic criteria for studies are:

Stability: The algorithms and tools run stable without crashing.

Reliability: The same results can be achieved with same data over and over again.

Efficiency: The results can be produced in reasonable amount of time given the volume of the data.

Auxiliary tasks to be supported: Added value of the data can be provided, for example in the form of models or higher-level abstractions.

Essential for all mining studies is the data preparation and data cleansing, which includes analyzing spurious values in the data and eliminating outliers that would impact or even distort the results. Data filtering is the primary key for a successful study; it deals with selecting subsets of data based on defined selection criteria (dependent on the research question to be investigated). Data binning is one such technique that tries to reduce the effects of minor observation errors by using intervals. Original values that fall in a given interval, a bin, are used as representative for a central value. For example, this is done in defect prediction, where files (or classes etc.) are put into bins to represent defect prone or non-defect prone files.

More details on proper design of mining studies can be found, for example, in the Cross Industry Standard Process for Data Mining (CRISP-DM) documentation [105].

2.3 Threats to the Validity in Mining Software Repositories

Next, we discuss some threats of mining studies. In particular, we discuss four approaches to address threats in bug prediction concerning stable rankings of estimation methods, sample size and bias in datasets, and time variance. Any of these can result in distorted or even questionable predictions.

Stable Rankings for Different Effort-Estimation Models. Menzies *et al.* [68] investigated 158 software effort estimation methods concerning their stability across different evaluation cirteria on various datasets and randomly selected features of COCOMO. The goal was to find a ranking among the estimation methods, since previously conducted studies suffered from 'conclusion instability' [107]. They use the COSEEKMO effort-estimation workbench that combines preprocessors to prune rows or columns, learners, such as local calibration, model trees, and standard linear regression, as well as different nearest-neighbor algorithms. Menzies *et al.* evaluated the performance of the methods by applying the model to a training set and then to a test set, as well as by collecting performance statistics using AR (Absolute Residual), MRE (Magnitude of Relative Error), or MER (Magnitude of Error Relative to the estimate), and counting the number of times a method loses with the Mann-Withney U test. They found that Local Calibration (LC), COCOMIN + LC, COCOMIN + LOCOMO + LC, and LOCOMO + LC perform better than all the other combination of methods and conclude that the combination of nearest neighbors with other methods is quite powerful.

Sample Size vs. Bias in Defect Prediction. Rahman *et al.* [95] performed a study how bias and size influence the results of mining studies on defect prediction. They sampled a dataset of high quality to several small 'biased and polluted sub-datasets,' to see if there is an effect on the bias of the defect prediction. They considered five kinds of bias for defect prediction: experience of the defect-fixer, severity of the defect, proximity to the next release deadline, the time to fix a bug, and the cardinality in size of the commits for each defect. Meta-models are used to evaluate if there are differences between the types of bias and their effect on the results. They found, that the type of bias does not have a significant influence on the prediction results. Furthermore, they investigated how bias, pollution, and size effect the prediction results. Size is at least as important as bias and pollution. Considering the performance metrics AUC and F50 it is even significantly more important.

Bias in Bug-fix Datasets. Bird *et al.* [8] investigated how biased datasets influence the performance of bug prediction techniques. A biased dataset is a dataset where links between the code repository and the bugs tracker are missing. In their study they considered the severity of the bugs, as well as the experience of the developer who fixed the bug. They found that severe bugs are most likely fixed by experienced users, since there exist often links between the bug-fix and the issue-tracker. Bird *et al.* tested their hypotheses on BugCache, a bug prediction tool for biased datasets. By sampling the dataset, they found that if BugCache is trained on a certain level of severity, it performs well for this severity, but badly for other severities. The usage of a model, considering biased data and trained for trained for all kind of bugs, is reflected in the performance of the bug-prediction model.

Time Variance and Variability in Defect Prediction. Ekanayake *et al.* [18] investigated the problem of variability in the accuracy of a bug prediction-model over time. They looked into four large open source projects and empirically identified various project features that influence the defect-prediction quality. In particular, they observed that a change in the number of authors of a file and the number of defects fixed by these authors influence the prediction quality. As a major conclusion their experiments showed that there exist periods of stability and variability of prediction quality. As a result, one should use approaches such as the one proposed to assess the model's accuracy in advance. These findings have a major consequence in that prediction quality is highly dependent on the time interval one selects for training the data to then make predictions. The accuracy of the predictions, therefore, can range from poor to high just depending on the selected time slices. Still, it remains open how to pick time intervals that represent stable (versus variable) phases in the software development.

2.4 Approaches, Languages, and Platforms for Mining

There are many possibilities how to support the mining of software repositories. Here, we introduce approaches, languages, and platforms that support, for

instance, data sharing, the examination of mining software repositories from different aspects, and the use of domain-specific languages (DSLs). Extracting and preprocessing data from software archives is time intensive and, therefore, need to be assisted by tools. For that, several platforms and tools have been developed that we briefly introduce in this section.

SeCold, *TA-RE*, *iSPARQL*, and *EvoONT* address this problem by providing the possibility of data sharing and making the replication of studies easier. *SeCold*, implemented by Keivanloo *et al.* [52,53], is "an open and collaborative platform for sharing software datasets." It provides research data online, to avoid that researchers preprocess the same data several times. The data is collected from issue trackers, such as BugZilla, Issuezilla, or JIRA, as well as from version control systems, such as SVN, CVS, or Git. This data then is merged to an abstract representation that mirrors the main concepts of these approaches. SeCold can also be used to find code duplicates as well as source-code-license violations. Studies and experiments in data mining are often not replicable due to the lack of shared knowledge about how the data is extracted. The results also depend highly on the selected parameters and heuristics. The goal of *TA-RE* is to address this issue. The corpus of *TA-RE* consists of the extracted data of software repositories and of an exchange language to share additional data that influences the results of the studies, but is not contained in the data itself, such as heuristics or parameter settings. The data may be further used to benchmark experiments. Kiefer *et al.* [56] extended SPARQL to *iSPARQL* and added the possibility to query for similar software entities, such as classes or methods. Furthermore, they developed EvoONT, based on the Web Ontology Language (OWL) that includes software, releases and bug-related data. It is possible to extend EvoOnt and integrate existing tools. With the combination of iSPARQL and EvoONT it is possible to mine software repositories that are represented in OWL. This combination supports the visualization and counting of code changes between versions, the localization of bad code smells or orphan methods, and the recommending of refactorings, as well as the computation of design metrics, such as size and complexity.

Mining software repositories includes a variety of aspects, concerning the evolution, the granularity of data, and meta-data of the projects, such as development process or team information. Yamashita *et al.* developed *E-CUBE*, an analysis tool for mining software repositories [125]. E-CUBE addresses platform evolution, target evolution, and scale evolution. They use FODA (Feature Oriented Domain Analysis) to create a DSL for E-CUBE. To target platform evolution, abstract types for bug repositories or code repositories are defined, instead of using a concrete repository. To address target evolution E-CUBE structures the data in a way so that it may be observed on several levels of granularity, such as file-level or method-level. The DSL provides the functionality to link projects to deal with the massive amount of data and the time for analysis. Spacco *et al.* [111] used software-repository mining to find better ways to teach and learn programming. They proposed the tool Marmoset. Marmoset collects snapshots of code, that are committed on saving operations. These fine grained

code changes are collected in a database with a data schema that allows one to apply lots of queries to get information about fine-grained code evolution [111]. *CVSgrab* [120] provides the possibility to visualize the evolution of large software projects. It uses evolution similarity metrics to group files with similar evolution patterns. CVSgrab may be used to get information about the evolution of the team and development process, as well as for the localization of development issues.

In [45] Huang *et al.* describe their approach to use Alloy (a language and tool for relational models) to build a family of DSLs, similar to SQL, to address the various applications of mining software repositories. For this, they applied FODA (feature oriented domain analysis) to get the feature model of MSR. The feature model is then transformed to a logical formula using Alloy, which is used to derive automatically the language elements of the DSL.

In [122] Würsch *et al.* developed a pyramid of ontologies for software evolution analysis named *SE-ON*,[9] in particular to support mining studies. These ontologies model the domains of software versions, issues, developers, and the like. As such, they constitute a common vocabulary for tools to work on and exchange mining results. For software evolution analysis, Ghezzi and Gall devised a framework and platform for software analysis as a service, named *SOFAS* [26,27]. This approach enables systematic and reproducible software evolution analyses that exploit semantic descriptions of software, bugs, and versions using ontologies, semantic web services, and a RESTful architecture. This constitutes a major milestone for reproducibility in software mining studies [25]. The backbone for software analysis services is based on the pyramid of software evolution ontologies named SE-ON [122] and, for example, is used for developer support in Hawkshaw [123,124].

3 Revisiting a Decade of Software Mining Studies

Reasons to mine software repositories are manifold. Work related to mining software repositories spans from feature location, to better understanding development processes, to improve power consumption of software. We present a comprehensive overview of existing works in the field. In particular, we present a systematic literature review of research topics and methods applied from the past two years, followed by a lexical analysis of the research papers from eleven years of the Mining Software Repositories conference.

3.1 Why Researchers Mine Software Repositories

Over the past years, many research fields adopted MSR approaches as a new means to achieve their respective goals or to improve existing approaches. We reviewed the MSR research published at MSR 2014, ICSE 2014, ICSE 2013, and FSE 2013, to better understand what problems can be tackled through mining

[9] `http://www.se-on.org/`

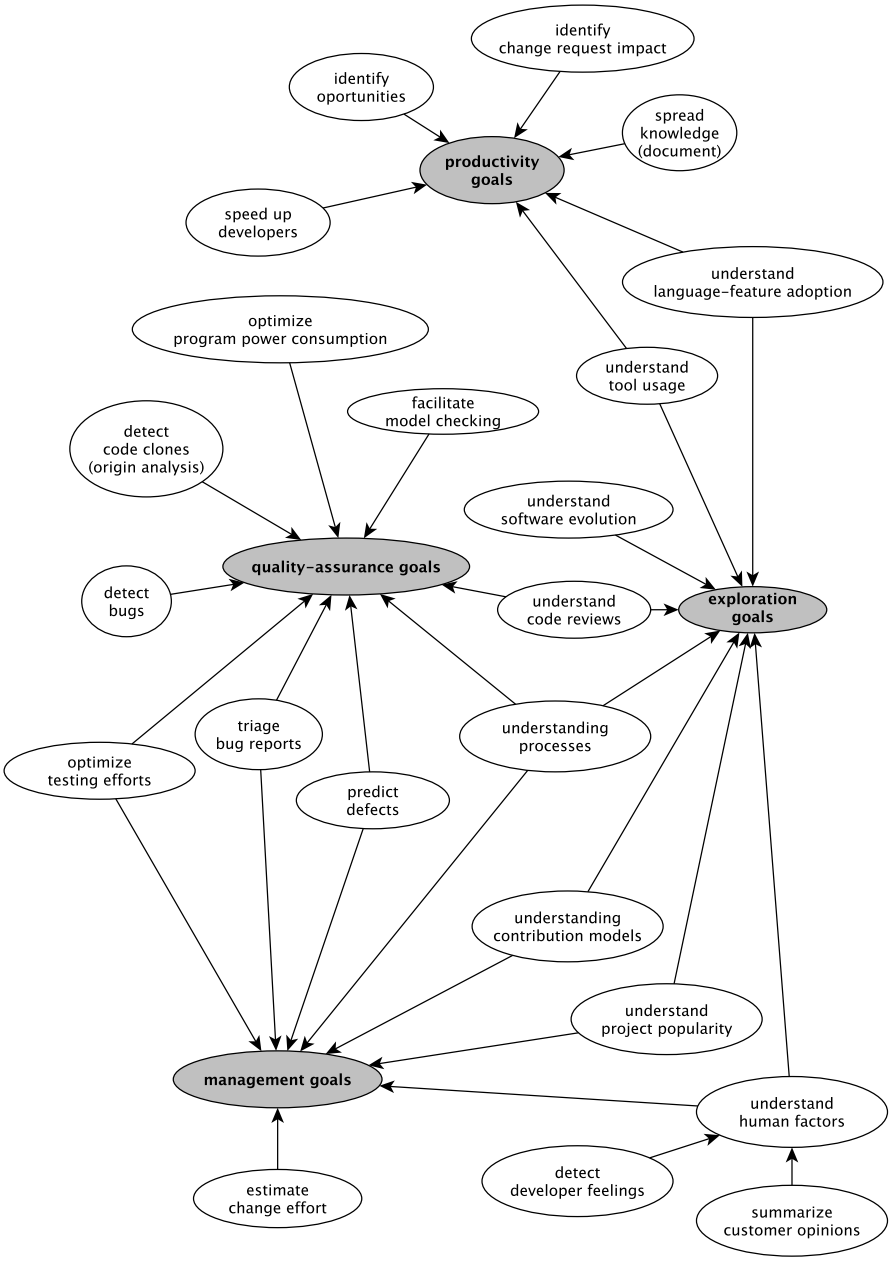

Fig. 1. Goals for Mining Software Repositories

software repositories and to gain an insight on the datasets used in the field. Figure 1 summarizes the goals that the reviewed work pursues. We identified three high-level goals: productivity goals, quality-assurance goals, and management goals. All three goal categories subsume concrete software engineering tasks that researchers try to support by developing targeted approaches and tools. In addition, the reviewed research includes exploratory studies to understand different aspects of software projects better, and meta-studies that aim at improving MSR methodologies.

Productivity Goals. Much past research aims at tools helping developers write better code faster, thus, pursuing the ultimate goal to make developers more productive. A category of such tools commonly referred to as *Recommender Systems for Software Engineering* [99] received much attention over the last decade. These recommender approaches typically mine code repositories [10,40,44,83], version-control systems [75], or even more fine-grained sequences of code changes to identify usage patterns [55,80]. Other productivity-enhancing approaches identify code locations that are likely to be affected by change requests from interaction- and version-control histories [129].

Another line of research focuses on the documentation of software systems. This research ranges from exploring the common forms of documentation [116], to enriching existing documentation with information about common pitfalls from bug trackers [51], examples from StackOverflow [115], or usage patterns from large code bases [81], to the automated creation of feature models for better software understanding [15,104].

In the light of the ever increasing amounts of data, e.g., due to the number and size of publicly available projects or additional data sources, such as Q&A-sites, it becomes even more difficult to find a specific piece of information. Lemos *et al.* [62] automatically expand code-search queries to increase the probability of finding the desired code snippets in the presence of potential vocabulary mismatch, i.e., when query and code use alternative terminology. Ponzanelli *et al.* [90] automatically look up relevant StackOverflow threads based on the developer's current coding context. Both approaches aim at faster knowledge accessibility and less need for context switching.

Exploratory work in the field includes the investigation of how developers use GitHub's pull requests to better understand change-management processes [33,97]. Other work focuses on the adoption of language features over time [17,98]. In the long run, such work will discover which kind of support developers need in learning and migrating to new language features. To improve widely used question-and-answer sites like StackOverflow, researchers investigated techniques to identify frequently asked questions [4] and reasons why questions remain unanswered [101].

Quality-Assurance Goals. Much research is dedicated to support developers in ensuring functional correctness of code, enhancing maintainability of code, and optimizing code. There are many different goals and approaches, which we discuss subsequently.

Bug detection is one of the most prominent areas in MSR. The goals are to detect previously unknown bugs [12,22,81], localize reported bugs in source code [34,82], and identify potential fixes for bugs [38]. Johnson *et al.* investigated on shortcomings of current bug-detection tools that keep developers from using them in practice [48]. They found that developers are dissatisfied by the additional effort required to use such tools and by the number of false positives they produce. They collected possible improvements to address the usability aspect.

Many evaluations of bug-detection approaches use FindBugs[10] as a test oracle. Therefore, researchers applied FindBugs to large sets of projects, to create benchmark datasets [72,102]. Other evaluations of bug detection approaches use issue trackers as oracles to test whether actually reported bugs would have been found by the respective approach. However, Chen *et al.* [14] identified a systematic bias in this evaluation technique, due to bugs reported only much later than introduced. Rahman *et al.* [94] compare static bug finders and statistical prediction methods to identify and discuss synergy potentials between these two fields.

Code clone detection and origin analysis are considered as further quality-assurance goals. Both identify code locations that are similar in terms of their structure or semantics. Mondal *et al.* [75] identify source code locations that are likely to require changes, based on their similarity to recently changed locations. Kevic *et al.* [55] identify locations that a bug report or change request most likely impacts, based on the impact of previous reports. Steidl *et al.* [113] present a framework for incremental origin analysis that scales even for very large code bases to make such approaches feasible in practice. Different oracles have been proposed to evaluate code clone detection approaches [59,76].

Tulsian *et al.* apply MSR methods to facilitate model checking in practical application [117]. Though model checking has improved significantly, it remains challenging to select the right checker for a given program and property. They prove that statistical evidence for correlations between checkers and program-property pairs can be mined. To the best of our knowledge, this is the first work to combine model checking and MSR.

Another rising new area of MSR research aims at the optimization of program power consumption on code level. Hindle [42] named this area Green Mining. Pinto *et al.* [88] explored which power-consumption-related questions matter to software developers. Other pioneering work investigates on frameworks for further research [43] and on evaluation benchmarks [130].

McIntosh *et al.* [67] inspected modern (lightweight) code-reviews in OSS projects. They find that reviews positively influence software quality, if review coverage is high and reviewers are involved in the development process. Beller *et al.* [6] find that the changes triggered from review processes are surprisingly similar between OSS and industrial projects. They analyze what kind of changes are triggered from reviews and what triggers them.

[10] `http://findbugs.sourceforge.net/`

Management Goals. Some research from the MSR community tackles management related goals. As management is a cross-cutting concern, some of these goals are also related to aspects such as the triaging of bug reports or quality assurance in general.

Bug triaging encompasses tasks such as finding report duplicates [2,57,60,61], automatic identification of non-reproducible bug reports [19], or predicting blocking bugs [119] and bugs that are eventually fixed [128]. Closely related research automatically infers bug-management processes from issue trackers [35], this exploratory work could help to detect differences between the intended and the actual process and identify potential for improvement.

Moreover, MSR techniques are used to estimate the effort (in terms of resources) required to realize an incoming change request based on historical change requests [100,132]. Other work investigates on how such effort models can be transferred between companies [71].

A different, quality-assurance-related management goal is the allocation of hardware resources for testing. Especially large industry projects face the problem that execution of all their regression tests takes too long for timely feedback. Therefore, Anderson *et al.* [3] and Shi *et al.* [108] propose approaches to rank tests according to their likelihood of identifying the next bugs.

Another closely related area is defect prediction. Its goal is to predict code modules (e.g. files, classes, or methods) that are likely to contain a bug, in order to optimize quality assurance efforts. While some defect-prediction research still explores new algorithms [47], most current effort is concerned with building cross-project defect-prediction models [24,79,131]. Lewis *et al.* [63] investigated the impact of defect-prediction tools on practitioners. They found that the predictors are rarely used, since, like the bug-prediction tools discussed above, they are to imprecise and investigating on their findings is much effort. Furthermore, the tools miss to present rationales for their findings to the users. In research on defect prediction, bug trackers are often used as oracles to evaluate the prediction quality. Herzig *et al.* analyze how misclassified bugs in such trackers impact evaluation results [41].

Recently, considering human factors becomes more and more important in software engineering research. The goal is to gain insight on the feelings of developers or other stakeholders involved in software development [37,77]. Such approaches are often referred to as Sentiment Analysis. Recent work has analyzed sentiments involved in discussions [89] and commit messages [36] on GitHub projects. Chen *et al.* [13] mine customers' opinions about software changes from reviews in mobile-app marketplaces.

Exploratory research aims at understanding how software and processes evolve over time, given changes in requirements, technologies, staff, and such. Past research has, for example, investigated feature churn on the basis of large source-code repositories [5,87] and changes of dependencies between modules in large software systems [9]. Other work focuses specifically on correlations between database-schema and code changes [93]. Brunet *et al.* [11] investigated whether

developers discuss design on GitHub, in commits messages, issues comments, or pull requests.

More exploratory work looked on reasons for project success and downfall [1,64]. Yamashita *et al.* [126] researched what makes developers contribute to OSS projects, while Matragkas *et al.* [66] explored indicators for a healthy OS community. Other research looked at how developers contribute to OSS projects on GitHub [86,106] and how they collaborate [118].

Meta Studies. A significant part of MSR research is to support the community itself and to bring it forward in terms of replicability and reproducibility [92]. Researchers have created large datasets of GitHub project (meta) data [32,121] and version histories [23] together with platforms to access them, as a basis for future research. Others investigated on scalable infrastructure and algorithms to perform analyses and searches on todays huge datasets [21,58].

To mitigate risks to the validity of evaluations, Kalliamvakou *et al.* [50] discuss common pitfalls and respective counter-measures of studies based on GitHub data. Linares-Vasquez *et al.* [65] looked at datasets from Google Play and discuss bias introduced by reusable app-modules. Merten *et al.* [70] discuss strategies to efficiently separate code from unstructured text in large datasets.

3.2 Characteristics of the Data Sources Used

MSR research has long passed the point of mining only software repositories. Alongside traditional sources, such as code repositories and version-control systems, many other knowledge bases, such as issue trackers, Q&A sites, and developers itself, are the target of mining approaches. Researchers collect data from a multitude of companies and projects, from the very small to the very large. They creat datasets of various sizes and with regard to different criteria. Many of these datasets are tailored to answer specific research questions, others to reproduce previous results, and others again to enable reproducibility and comparability of future work.

In the majority, researchers evaluate their approaches using one or multiple software projects as exemplary subjects. Depending on the respective approach, they retrieve different types of data from these projects, e.g., sources of test and production code, execution traces, change histories, bug reports, developer or user discussions, and even energy-consumption traces. The data sources from which this data can be retrieved and the effort required to do so varies greatly.

This section first gives an overview of which data sources have been exploited and how and why they were selected. Second, it presents some filtering strategies applied to extract data from these sources and the properties of the resulting datasets. Last, it discusses the limitations of these datasets and the experiments performed on them as well as the issue of reusability of those datasets.

Data Sources. Datasets have a huge impact on the validity of MSR experiments. Oftentimes, datasets qualify for the generalizability of the findings. Thus, to reduce the threats on external validity, researchers constitute big datasets that

include diverse data [78]. The datasets' diversity can stem, for example, from small and large change sets or from a lot of different developers.

To create big datasets, researchers often mine the source-code repositories of large OSS projects, because this data is publicly accessible and contains many data points. Popular examples of such projects are Eclipse [19,24,60,61,119,128], Firefox [19,128,130], and the Linux kernel [34,70,87]. However, all data points in such datasets originate from the same project and might not be representative for other projects.

To increase the diversity within datasets, researchers mine different projects from meta-repositories, such as Apache Projects [12,14,77], the Eclipse Marketplace [121], the Gentoo Repository [9], or Google Play [51,57]. Many of these contain more diverse projects, ranging over multiple sizes and maturity levels. Since all projects in meta-repositories are accessible in the same way, the effort to extend datasets is manageable. However, diversity may still be limited, since projects in such repositories often either belong to the same domain or are developed by the same organization. This issue was discussed by Nagappan *et al.* [78] who presented an approach to select diverse sets of projects for evaluations, in order to increase external validity. Proksch *et al.* [92] further discussed how to use this idea for a standardized platform of evaluation datasets.

Recently, the emergence of mega-repositories, such as SourceForge [62,98,131], GitHub [32,121], or Google Code [131], helped researchers more easily access large quantities of projects. Mega-repositories contain a huge variety of projects, targeting all kinds of platforms. However, researchers showed that the variety of projects in such repositories can bias datasets. Kalliamvakou *et al.* [50] show, for example, that the majority of the projects on GitHub are personal and inactive; that GitHub is also used for free storage and as a Web hosting service; and that almost 40% of all pull requests do not appear as merged, even though they were.

In addition, researchers have investigated on closed-source, commercial products [19,22,96,82,63,132]. Some research could show similarities, other differences between commercial and OS software. A general problem with evaluations on commercial products is the availability of the datasets, which is mostly limited by legal restrictions. Therefore, such evaluations are typically not reproducible.

Besides repositories, benchmark datasets, like the Nasa PROMISE repository,[11] are valuable data sources. These are specialized datasets that contain precomputed metadata used in the evaluations of respective tools. In contrast to the non-standardized way of retrieving data from repositories, benchmark datasets enable the comparability of results.

To further investigate on software projects, MSR was complemented with the mining of issue trackers, such as JIRA [19,77] or BugZilla [19,24,34], Q&A sites, such as StackOverflow [4,88,90], as well as email discussions [37,114], discussion threads [11,89], documentation sites [1,116], code reviews [6,67], change requests [129], and customer reviews [13]. To fully exploit these as data sources, the linking between issues and source code changes became a research target in its own right [113,129].

[11] `http://openscience.us/repo/`

A further data source constitutes observational studies, which capture, for instance, interaction traces of developers within the source code [55] or interactions with specific tools [63]. The effort to obtain such datasets is especially high, since it requires the contribution of large numbers of developers and the legal and privacy matters are particularly present.

Data Sampling. Much of the reviewed work does not specify why one data source was selected over the other. Mostly, the selection seems guided by the specific requirements of the respective approach. The generalizability of the results is pursued by increasing the datasets' size. Large project sets (up to 140k projects) are often sampled from mega-repositories, e.g., in [11,13,77,86,98], while smaller sets (1-50 projects) are also collected manually, e.g., in [19,24,38,57,61,119]. The number of projects is limited by the manual effort and time required to include further projects.

The primary filtering criteria is the availability of required (amounts of) data about the projects and the format this data is available in. For example, Erfani Joorabchi *et al.* [19] filter for projects that use either BugZilla or JIRA as their issue trackers, because the prototype implementation of their approach supports these two platforms; Kechagia *et al.* [51] select clients of the BugSense SDK, as crash reports are available for these projects; Brunet *et al.* [11] select projects with more than 50 discussion threads on GitHub, as they want to detect design discussions; Aggarwal *et al.* [1] select popular projects from GitHub that have documentation, as they want to investigate on relations between popularity and documentation.

Some work considers the diversity of their sample, with respect to dimensions such as the programming languages [3,131], the project domain [13,24,46,63], project size [13,46,63], project maturity (age, size, quality measures) [41,46], and open-source vs. industrial software [116].

After the selection of a project set follows the extraction of data from those projects, e.g., by analyzing source code [59], change history [80,87], discussions [77,88], or bug reports [2,57]. In this process, again, different sampling strategies are applied. For example, researchers often limit datasets to datapoints with certain properties, like closed bug reports [2,14,41,57,61,128], answered questions [4] or discussions with certain keywords [88], code entities with online documentation [51], or issue reports with links to code changes [67,129]. Work that uses historical data, like change history or discussions, often limits the considered time period [3,13,77,128,129].

A special case in data sampling is the interaction with developers. For surveys, the most common strategy is to just take all received answers [59,90,113,118]. Researchers also include validation questions, to filter participants, especially when calling out to the general public. The driving factor seems the need for a sufficiently large number of participants.

Reusability. When looking at the availability of datasets, we found that only 27 publications (about 29%) make the respective datasets available for reuse. We counted only those papers that provided an explicit link or instructions on how

to obtain the dataset. Another 2 publications (about 2%) name legal issues in the context of industry cooperations as the reason for not publishing the dataset.

Only 10 publications (less than 11%) actually reuse a dataset from previous studies. Unfortunately, it is difficult to understand why this is the case, as we did not encounter any work that discusses problems or shortcomings of available datasets as the reason for coming up with a new one. Future work should identify reasons for this low reuse rate, e.g. insufficiencies of the datasets, and respective mitigation strategies.

The remaining 58% of the work does not mention availability at all. This shows that reusability is not generally considered by the community.

3.3 A History of Artifacts in Mining Research

The artifact selection has a great impact on the replicability and reliability of an experiment's result. To better understand which factors influence the selection of artifacts within previous experiments, we investigated the proceedings of the past MSR conferences. Specifically, we conducted a lexical analysis on all accepted papers of the past eleven MSR conferences to answer the following questions:

Q1: At which point in time were particular artifacts more popular? How do technical developments influence MSR research?
Q2: Which artifacts will be used most likely in future?

Procedure. In our effort investigating these questions, we created for each past MSR conference a list including the most popular terms of each paper. To elicit each year's list of most popular terms, we first collected the proceedings of the past eleven years. To establish our dataset, we only considered full- and short-papers, disregarding other paper types, such as papers related to mining challenges. Parsing the remaining 297 papers into strings,[12] enabled to analyze the content of each paper. First, we eliminated the text, which follows the last occurrence of the term "'references"'. Then, we split these strings into tokens, according to the whitespaces in the text. To improve the accuracy of our analysis, we performed well-known text-preprocessing steps, which included stop word removal and stemming. Specifically, we removed stop words included in an English stop word list of the Journal of Machine Learning Research.[13] For stemming the tokens, we used the Porter stemming algorithm [91], which strips suffixes from terms. To find the most popular terms within each paper, we counted the occurrences of each remaining token, producing a set of the top-ten terms per paper. Finally, we combined for each year the top-ten lists of each paper to a map, which includes pairs of terms associated to the count of appearance in a top-ten list.

[12] We used the java PDF library Apache PDFBox, https://pdfbox.apache.org/
[13] http://jmlr.csail.mit.edu/papers/volume5/lewis04a/a11-smart-stop-list/english.stop

Q1: At which point in time were particular artifacts more popular? How do technical developments influence MSR research? Overall, we identified 15 distinct sources of artifacts which are used for mining. These resources include *cvs, git, mercurial, github, svn, jazz, bug, commit, patch, message, stackoverflow, email, twitter, blog*, and *tutorial*. There are resources which are closely related to the source code, such as source code repositories. In contrast, other resources are more generic, not targeting particularly towards software engineering. To examine the appearance and popularity of the different kinds of artifacts, we plotted the artifacts' popularity metric along a timeline, see Figure 3.3. The diagram depicts in the upper part technologies for versioning source code, while the remaining artifacts and sources for artifacts are represented in the lower part of the diagram.
When looking at the popularity of different version control systems over the years, the data indicates that in first experiments of the MSR conference *cvs* was used predominantly. Then, in 2009, several version control systems, were mentioned a lot in papers: *jazz*, *svn*, and *git*. However, from 2009 onwards version control systems were not as prominent as before. Our dataset indicates that terms like *github* and *mercurial* gained popularity.
Various terms related to communication channels started to appear more frequently from 2009 onwards. While terms like *email* and *message* started to be mentioned already in 2006, another category of artifact sources, namely social media, started to appear predominantly in 2011. From 2011 a conglomerate of various artifacts from different sources were included in MSR experiments.
Our data indicates that more and more diverse artifacts are considered in a mining study. The consideration of more and diverse artifacts highlights different aspects within the programming tasks of developers. It potentially converges more and more to the actual environment in which a developer works. However, the introduction of new artifact sources reduces the reproducibility and replicability as each experimenter then selects a particular combination of artifacts to be mined out of the set of available artifacts.

Q2: Which artifacts will be used most likely in future? To make a qualified guess which artifacts will become even more prominent in future, we analyzed which subtopics of MSR are currently emerging and would potentially involve new artifacts. Hence, we filtered our dataset for terms which appeared for the first time at most three years ago. The terms revealed by our filtering scheme can be categorized into three major topics:

- **Green Mining**, indicated through terms, such as *energy*, *consumption*, *green*, *power*, *watt*, and *energy-greedy*
- **Mobile Software Engineering**, indicated through terms, such as *mobile*, *chrome*, and *browser*
- **Human Aspects in Software Engineering/ Social Mining**, indicated through terms, such as *emotion*, *behavior*, *twitter*, and *stackoverflow*

Interestingly, the term *nonisolated* appears as well in this filtered list, further indicating that the examination of several integrated artifacts bears further poten-

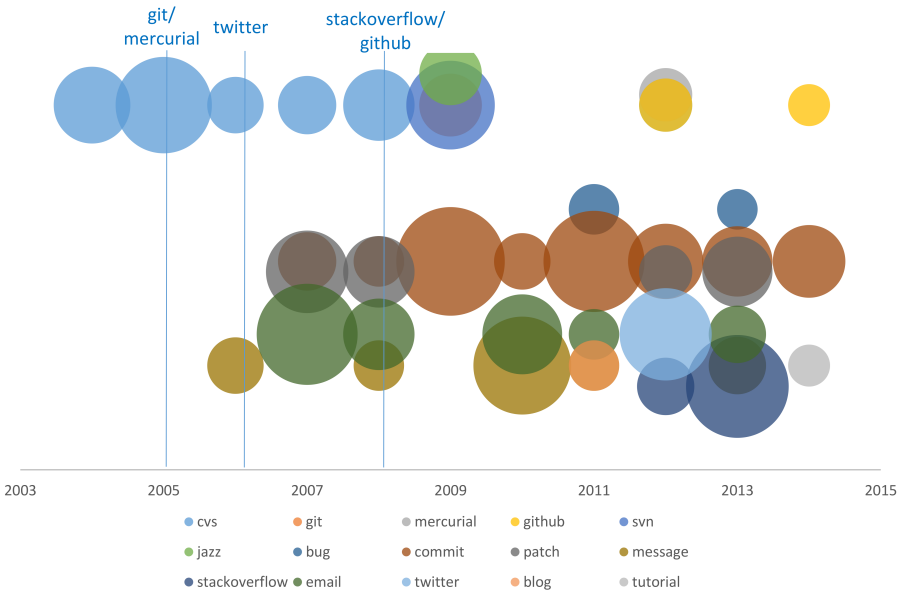

Fig. 2. Popularity of artifacts and artifact-sources in MSR since 2004.

tial. These three research areas uncover artifacts which could potentially become more prominent. As example, related to green mining, artifacts about CPU, I/O, and memory traces are particularly interesting. Advanced technologies, which necessarily require energy-aware applications, such as the Google Glass, might unravel even further artifacts. Considering mobile software engineering, uncovers that, for example, data gathered through Web IDEs could become relevant for further experiments. To better understand human behavior within the software development process, a variety of data sources can be mined. Data sources to better understand developers capture either data about the developer itself, as example through psycho-physiological measurements in particular situation while coding, or capture data about social interactions of developers. Devices, such as eye trackers, electrodermal activity, electrocardiodiagrams, or electroencephalograms, allow detailed insights about individual behaviors during a programming task. Communication threads in *emails* or *messages* were part of early experiments in the MSR community. However, in recent years the range of these artifacts has increased. Recent experiments mine Twitter feeds, so it is conceivable that social networks, such as Facebook, will eventually become a mining artifact for software engineering as well.

3.4 Discussion

We performed a survey of the MSR research published at MSR 2014, ICSE 2014, ICSE 2013, and FSE 2013 and performed a lexical analysis on the proceedings of the last decade (2004-2014)of the MSR proceedings. Comparing the used approaches and artifacts supports our hypothesis that MSR research questions change over time. We hypothesize that there is a constant development of ongoing research, driven by two main factors: First, MSR research solves software engineering problems and respective tools emerge and spread. Second, new and evolving tools present new features, which pose new possibilities and challenges to MSR research.

We observed that today's artifacts hold more and more data about individuals in software engineering processes. Researchers have shown that modern mining techniques are able to extract valuable information from such data. This enables more personalized investigation approaches and tools in the future. However, it also gives rise to new problems, such as the major privacy concerns that come with the mining of data from social media.

The survey shows that MSR research already came a long way towards representativity of evaluation datasets and generalizability of respective results. However, more work is required to further increase the diversity and to improve reusability of evaluation datasets which can increase the comparability of results.

4 Replication of Mining Studies

The replication of studies in mining software repositories is essential to compare different mining techniques and their results across many projects. The study of Ghezzi and Gall [25,27] reported that the replication of these studies is still at a rather early stage. However, the replication of mining studies is just as fundamental as the studies themselves.

Very few studies can be reproduced because of the lack of availability of the tools or the data used [30] for the study: The tools used in the studies are accessible only for approximately 20% of all the studies and for another 20% they are only partially accessible. Even when publicly available, they are difficult to set up and use. As a matter of fact, they are mostly prototypes (or a collection of scripts) and work only under rather specific operating systems and settings.

Data can be divided into raw and processed data. Raw data can be directly retrieved from publicly available sources such as version control systems, issue trackers, plain source code, mailing lists, etc. Preprocessed data, which is what is actually used to perform the mining, is the result of the retrieval and processing of raw data. While raw data is usually widely available (at least in the case of OSS projects), processed data is not.

Different approaches have been proposed to address this problem. But these efforts are mainly aimed at creating large, internet accessible, data repositories, such as PROMISE [103]. Some of these internet repositories offer a query-able

static collection of data for specific projects fetched from single [73] or multiple sources [84,85]; other online repositories allow the user to interactively run specific analyses on her own projects of interest [29,31].

Large static software data repositories such as PROMISE[14], Krugle[15], or Open Hub (formerly known as Ohloh)[16], provide third party applications with a common body of knowledge to build analyses upon. They could also be useful to provide benchmark data to test and compare similar tools/analysis that use such data. However, they do not target the *replication* of analyses and are based on static data of a multitude of software projects. The interactive features of these online repositories limit the user to only the pre-defined analyses the platform offers by design. Replicability is thus still limited to very few and specific cases. While these online repositories are certainly a step into the right direction, a more systematic approach to replicability is required [25].

4.1 Platform Support for Mining Studies

Platforms to support mining studies have been developed, although their number is still very low, given that the effort to spend on providing a mature platform is pretty high and that scientifically such an achievement is hardly rewarding.

SOFAS (Software Analysis as a Service) is a platform developed at the University of Zurich that enables a systematic and replicable analysis of software projects by providing extensible and composable analysis workflows [26]. These analysis workflows can be applied repeatedly and in the same manner on a multitude of software projects, facilitating the replication and scaling of mining studies.

Using SOFAS, Ghezzi *et al.* investigated the mining studies of the MSR conference from 2004 to 2011 [27] and found that from 88 studies published in the MSR proceedings in that time frame, they could fully replicate 25 empirical studies using their platform. Additional 27 studies could be replicated to a large extent. The remainder of 36 studies could not be replicated due to lack of tool support or automation of the models or that were proposed in the studies. A platform such as SOFAS that focuses on analyses services up to the level of statistical analysis can support (and automate) close to 60% of the published studies. This shows that there is a high potential for such platforms to support the automation and replication of mining studies.

Dyer *et al.* developed Boa [16], which is a mining tool for large code repositories, which translates queries formulated in a domain specific language into parallelized code that runs on a Hadoop cluster. It is one of the few tools that address a systematic extraction of data from code repositories. For that, it offers a domain-specific language and infrastructure that supports the testing of hypotheses and the re-running of mining experiments. It can be used to mine repository metadata as well as source code across thousands of software projects.

[14] www.promisedata.org
[15] www.krugle.org
[16] www.openhub.net

Formulating queries in the Boa DSL enable to look for the existence of particular code fragments (e.g., assert statements or specific class names), but not to perform more elaborate investigations, such as complexity computations, code clone or code smell detection or other more complex structural analyses. However, it is certainly a major contribution to the field of replicating software mining studies as it provides a web-based interface to its infrastructure and a DSL as query language.

Kenyon developed by Bevan *et al.* [7] is a platform designed to facilitate the fact extraction from code archives and configuration management systems. Its features enable a multi-project analysis of repositories by providing a common set of importers from various kinds of archives. As such it can be seen as one of the early platforms to deal with the peculiarities of different archiving systems in version control, issue tracking, or configuration management. It is mainly a toolbox to build one's analyses on top, but by itself does not provide specific mining features. It can be seen as a middleware between the specifics of software archives and the applications that actually perform the mining parts.

4.2 Replication of Software-Mining Studies

The importance of replication has long been recognized in other fields, such as statistics, field research, or psychology. We highlight the following quote about replication taken from [39]:

> "Replication is the key to the support of any worthwhile theory. Replication involves the process of repeating a study using the same methods, different subjects, and different experimenters. It can also involve applying the theory to new situations in an attempt to determine the generalizability to different age groups, locations, races, or cultures.
> [..]
> Replication, therefore, is important for a number of reasons, including (1) assurance that results are valid and reliable; (2) determination of generalizability or the role of extraneous variables; (3) application of results to real world situations; and (4) inspiration of new research combining previous findings from related studies."[39]

According to [109], replication can be divided in two main categories: exact and conceptual replication. *Exact replication* is when the procedures of the experiment are followed as closely as possible. *Conceptual replication* is when the experimental procedure is not followed strictly, but the same research questions or hypotheses are evaluated, e.g. different tools or algorithms are used or some of the variables are changed.

In [25], a mining study was considered replicable whenever it could be replicated, either conceptually or exactly, using mining and analysis services available in the mining platform *SOFAS*. Table 1 describes how many of the analyzed studies published in the MSR conference 2004–2011 could be replicated and to which extent.

Study category	Number of studies (%)	Replicable	Partially replicable	Not replicable
Version History Mining	8 (9%)	4	0	4
History Mining	17 (20%)	0	8	9
Change Analysis	13 (15%)	5	6	2
Social Networks and People	19 (22%)	6	5	8
Defect Analysis	19 (22%)	8	6	5
Bug Prediction	8 (9%)	2	2	4
	88 (100%)	25 (30%)	27 (32%)	32 (38%)

Table 1. Replicability of MSR studies from 2004–2011; source: [25]

As a result, 52 out of the analyzed 88 mining studies (i.e. 59%) could be fully or at least partially replicated with mining services offered by *SOFAS*. The replication of these studies typically requires basic services such as import from various version control or issue tracking systems; it further requires composite services such as change coupling analysis or linking issues to fixes in the code, all of which can be supported by a platform such as *SOFAS*. The details of how each study category can be replicated are given in the paper [25].

In terms of case studies that have been investigated in the MSR conferences, a study by Gonzalez-Barahona *et al.* [30] reported the following most often analyzed projects until 2010: PostgreSQL (18), ArgoUML (16), Eclipse (15), Apache Web Server (10), Gnome and Linux (7). The study shows that there are rather few studies that have been frequently analyzed, but it also shows that some of them could be used as reference projects for further replication studies.

4.3 Performance of Prediction Studies

With any mining study, its performance is essential. For that, we briefly look into some of their performance aspects, in particular for prediction studies.

Time Variance. It depends on the time interval chosen for training whether a (defect) prediction study has better or worse performance. Studies such as [18] investigate time variance dependencies by taking different time intervals (such as 1 or 2 months etc.) and computing the prediction model.

Calibration of Learners. It also depends on the calibration of the (machine) learners used and the coefficients computed for coming up with a highly accurate prediction. This means that data preparation (data cleansing, binning, filtering, etc.) in combination with the proper configuration of learners is essential for reproducible and replicable studies. Keung *et al.* analyze aspects of learner calibration for selecting the best effort predictor in software effort estimation [54].

Data Preparation. Data distribution analysis, outlier elimination, and binning (failure-prone, non failure prone) are essential. As for binning there are quasi standards in the MSR community that are widespread and accepted, for example, failure-proneness classification is based on the median of failure distributions; this however is a model that could be more fine-tuned to the data and less binary. Learning about the data (including its visualization) are key practices in data mining (see CRISP-DM [105]) and need to be part of any mining study.

Benchmarking. Results of a mining study are typically not benchmarked, but at most compared to some "baseline technique". This however has a bias in terms of what is considered such a baseline technique and whether this is representative for the kind of data, the research question, and the case studies to compare with. As there is lots of data sources available (such as PROMISE, or the MSR Mining Challenge datasets), unfortunately, there is no benchmark data (results of mining studies) out there. This asks for an intensive investigation, as being performed, for example in our most recent SNF project named "Whiteboard."

4.4 Replicating Mining Studies with SOFAS

For replicating a mining study one has to take the dataset of the experiment, prepare the data according to the published data preparation mechanisms (e.g., distribution analysis, filtering outliers, binning, etc.) and then start with the same dataset the modeling; one would typically use functions for importing data, preprocessing it, and delivering models to start with for data mining and machine learning. The latter would be outside a mining platform, but be embedded features of machine learning software. As such, the machine learning parts are outside a platform, such as SOFAS, but the platform would provide interfaces to the machine learner.

Fully replicable with SOFAS means that the published study can fully be computed inside the platform including the presentation of the results. 25 out of 84 studies (i.e. 30%) in our set were fully replicable.

Partially replicable means that platforms such as SOFAS would provide all functionality until it gets to the machine learning or statistics parts. In the replication study, 27 out of 84 (i.e. 32%) fell in that category.

This left 32 out of 84 (i.e. 38%) in the residual of non-replicable studies. Summing up the fully and partly replicable studies this amounts to 52 out of 88 (i.e. 59%) of all the published studies by then. This clearly shows the potential for such platforms as they can be considered major contributors to the replicating software mining studies.

Given a mining platform such as *SOFAS*, the replication of an already published study is just one aspect. A further substantial benefit is that the original study can be extended rather easily in at least two ways:

- Extending a study by adding more software systems to the dataset
- Extending a study by refining or adding research questions to the analysis

Given the goals of replication (assuring that results are valid and reliable; determining the generalizability of extraneous variables; applying results to other (real-world) situations; and inspiring new research (questions) combining previous findings) the two dimensions of extensibility are essential for the field of mining studies. We need more studies of the same kind to assure our findings are the same and that they generalize beyond the typical small body of systems (a handful to a dozen).

Next, we look into one particular replication study that extended an original software mining studies by adding more systems to be analyzed and by extending its research question.

4.5 Replicating the Study on "Do time of day and developer experience affect commit bugginess?"

The original study, performed by Eyolfson *et al.* [20], investigates the correlation between the bugginess of a commit and a series of factors: the time of day of the commit, the day of week of the commit, the experience and commit frequency of the committer. Such a mining study is based on the history of a project extracted from its version control system combined with data from issue tracking. The authors consider as a bug-introducing commit any commit for which there exists another commit explicitly fixing the former at a later point in time. To identify them, the authors first detect all the bug fixing commits using a standard heuristic used in the MSR field: finding the ones that have specific keywords (e.g. "fix", "fixed", etc.) in their commit message. Buggy commits are commits that changed files that were involved in such fixes.

In their investigation, the Eyolfson *et al.* studied the two projects, the Linux kernel and PostgreSQL, and discovered four major results: (1) about a quarter of the commits in a project history introduce bugs; (2) the time of the day does actually influence the introduction of bugs, as late night commits (between midnight and 4 AM) are significantly buggier and morning commits (between 7 AM and noon) are less buggy; (3) regularly committing developers (daily-committers) and more experienced committers introduce fewer bugs; and (4) the influence of the day of the week on the commit bugginess is project-dependent.

In the replication study of this paper published in [25], Ghezzi *et al.* verified these four findings by fully replicating the original study. Moreover, they also tested if the findings also hold for three additional OSS projects: Apache HTTP, Subversion, and VLC. They extended the original study by adding more software systems as subjects to the study. And they also extended the study by refining and adding more research questions. The goal of the replication study was to show the potential of a systematic mining platform such as *SOFAS* to draw broader (in number of systems investigated) and deeper (in number of questions addressed) conclusions with little additional effort.

To replicate this study, the following steps had to be performed:

1. Extracting the full version history of the project: This can be accomplished by using a version history extractor.

2. Identifying the bug-introducing and bug-fixing commits (i.e. revisions) from the version history. This can be accomplished by a bug-revision linker, which would find the bug-fixing commits. To accomplish that, the replication study encoded the bug-fixing identification algorithm for Git and Mercurial and provided those in their *SOFAS* platform. Actually, the heuristics was adapted to support a larger vocabulary (*fixes*, *fixed*, *bug(s)* in addition to *fix*).
3. Extracting the commit frequency and experience of the all the developers who introduced bugs (calculated from the bug introducing date). This is achieved by querying the data extracted in the first step with specific (SPARQL) queries, as *SOFAS* works with RDF and ontologies (for data representation) and SPARQL (for querying).
4. Aggregating the buggy commits by time of the day, day of the week, developers experience, and commit frequency. This is also achieved with SPARQL queries.
5. Interpreting the results. *SOFAS* simply supports the extraction and combination of analyses and data. The conclusions still have to be drawn manually by the users of such analyses, depending on their specific needs.

The replication study analyzed the projects in the time frame of July 1-10, 2012. The results were lined up with respect to the original study:

Percentage of Buggy Commits: The replication study confirmed the results of the original study for both Linux and PostgreSQL. Some slightly different values were explained by the different heuristics used to detect bug fixes and the different analysis date (the projects were analyzed a year later than the original study). Moreover, all the other analyzed projects exhibited similar values (22-28%), as shown in Table 4.5. These results even indicate a trend worth investigating in more detail and with a larger body of projects.

	commits	bug-introducing commits	bug-fixing commits
Linux	268'820	68'010 (25%)	68'450
PostgreSQL	38'978	9'354 (24%)	8'410
Apache Http Server	30'701	8'596 (28%)	7'802
Subversion	47'724	12'408 (26%)	10'605
VLC	47'355	10'418 (22%)	10'608

Table 2. Commit characteristics of the analyzed projects - source: [25]

Influence of Time of the Day on Commit Bugginess: The replication study confirmed the results of the original study for both original projects Linux and PostgreSQL. Moreover, the analysis of the additional projects substantiates the finding of late night commits (midnight until 4 AM) versus morning and afternoon commits. However, the replication study showed that these 'windows' of

below average bugginess greatly vary across projects. Furthermore, the individual commit bugginess of projects follows different patterns which do not allow any further generalization on the influence of the time of the day on the commit bugginess.

Influence of Developer on Commit Bugginess: The replication study confirms the original results that bugginess decreases with greater author experience for all the projects analyzed. In all projects, a drop in commit bugginess is evident as the time a developer has spent on a project increases. In four of the projects such drops happen between 32 and 40 months of experience, while for the remaining one, PostgreSQL, such a drop takes place at 104 months of experience.

Influence of Day of the Week on Commit Bugginess: The replication study confirms also that the day of week can have some influence on the commit bugginess. However, the added projects and their commit bugginess present quite different patterns. Apache HTTP server and Subversion tend to have two commit bugginess 'phases': a higher than average one from Tuesday to Friday and a lower than average from Saturday to Monday. The bug introduction in VLC is almost the opposite, as it is lower in the middle of the week (Wednesday to Friday). The analysis of these additional projects shows that the finding of the original project that commits on different days of week have about the same bugginess is not generalizable. Moreover, it also shows that the results of a previous study by Sliwersky *et al.* [110], which showed that Friday was the day with the most buggy commits (based on the analysis of Mozilla and Eclipse), cannot be generalized.

4.6 A Plea for Conclusion Stability

Given the need for replication to achieve mining goals and the potential support of analysis platforms, we need to scale and extend studies, and come up with benchmarks based on a multitude of projects analyzed. To advance the field of software mining studies and enable better conclusion stability across studies, at least two things have to be provided:

Infrastructures and Mining Platforms. Analysts should be able to run software mining studies on a large corpus of software systems with only little effort. It is essential to guide them through the process of designing and carrying out empirically sound studies based on good patterns for software data analysis and point them to potential pitfalls based on anti-patterns. Replicability of the studies is key and should be fostered by an adequately formal description of data, data-processing, study design, and study results.

Benchmarking. Software forges store vast amounts of artifacts and data related to the software process. This information can potentially serve as a baseline to assess whether a given software system follows a "healthy" evolution path or whether its underlying development process needs adjustment.

5 Conclusion and Outlook

Mining software repositories is a research area that gained a lot of attention over the last decade. In particular with the open and free access to software archives, such as version control systems, issue trackers, or various other kinds of data about a software project, mining version history has shown great potential for advancing the state-of-the-art in software engineering. Many studies have been published so far, with quite varying benefit to the field. It is, therefore, important to take a fresh look onto the field and discuss the goals, approaches, artifacts, and replicability of these mining studies.

We revisited a decade of software mining studies and highlighted mining goals, study replicability, and trends in mined artifacts. Since the artifacts used for mining software repositories are highly diverse, we visualized changes in artifacts, and thereby indicated some future trends. We also discussed how the replicability of studies is influenced by the evolution of the artifacts.

Our systematic literature review of the research *topics and methods* en vogue in the last two years showed that the main goals to mine software repositories are mostly productivity goals, such as the identification of change impacts, as well as making the development more effective. Other goals are to support quality assurance, for instance by finding and predicting bugs, or the detection of code clones and the calculation of test effort. Management-relevant goals, such as the estimation of change effort, the understanding of human factors, or the understanding of processes, are pursued as well, but by a much smaller number of studies.

Additionally, after investigating the *reusability* of studies, we found that still very few studies are replicable due to the lack of replication information including data and tools. Only 40% of the studies provided their datasets for reuse, for only about 20% of the studies the tools are available. Only 2% of the studies mentioned that data could not be provided due to legal issues. However, if data is available and accessible (e.g. in OSS repositories), mining platforms such as *SOFAS* or the like can replicate a substantial amount of studies (currently up to 60%) by providing automation support for the analysis and mining.

Software data repositories, such as PROMISE, Krugle, or Open Hub provide the possibility to apply analyses of the data they already preprocessed. However, this does not solve the problem of replicability. Mining platforms do address this problem by supporting the systematic and repeatable analysis of software projects. Still, for conclusion stability, many more systems have to be analyzed and studies have to be replicated on a large scale to enable deep conclusions and benchmarking of systems.

To analyze the mining trends, we investigated 297 papers of the past eleven years of the MSR conference lexically to analyze the *artifacts* used for mining. We found that the popularity of different version control systems changed quite substantially over the years. For the first experiments merely CVS was used. In 2009, the mining of the version control systems Jazz, SVN and git was predominant. From 2009 onwards also GitHub and Mercurial are used for min-

ing. Emails and messages are investigated since 2006 and social media gained popularity since 2011.

We investigated the terms that appeared the first time in the last three years to make an educated guess which artifacts will get more popular in the near *future*. We identified three main topics that could gain popularity in future: green mining, mobile software engineering, as well as human aspects in software engineering and social mining. Artifacts for green mining could be, for instance, CPU or I/O traces. Artifacts for mobile software engineering may include data from Web IDEs and for human aspects, for instance, psycho-physiological measurements may be conducted, using eye trackers while coding.

As software-project data continues to grow fast, the plethora of mining studies will grow along with the potential to gain more and better insights into aspects of (more) productive software development. However, a clear focus will have to be on conclusion stability of these studies, provided by systematic experiments and studies combined with their proper replicability.

Acknowledgements

This work was partially funded by the German Federal Ministry of Education and Research (BMBF) within the Software Campus projects *KaVE* and *Eko*, both grant no. 01IS12054. The views and opinions expressed in this article are those of the authors and do not necessarily reflect the official policy or position of the funding agency.

References

1. Karan Aggarwal, Abram Hindle, and Eleni Stroulia. Co-evolution of project documentation and popularity within github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 360–363, New York, NY, USA, 2014. ACM.
2. Anahita Alipour, Abram Hindle, and Eleni Stroulia. A contextual approach towards more accurate duplicate bug report detection. In *Proceedings of the 10th Working Conference on Mining Software Repositories*, MSR '13, pages 183–192, Piscataway, NJ, USA, 2013. IEEE Press.
3. Jeff Anderson, Saeed Salem, and Hyunsook Do. Improving the effectiveness of test suite through mining historical data. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 142–151, New York, NY, USA, 2014. ACM.
4. Kartik Bajaj, Karthik Pattabiraman, and Ali Mesbah. Mining questions asked by web developers. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 112–121, New York, NY, USA, 2014. ACM.
5. Boris Baldassari and Philippe Preux. Understanding software evolution: The maisqual ant data set. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 424–427, New York, NY, USA, 2014. ACM.

6. Moritz Beller, Alberto Bacchelli, Andy Zaidman, and Elmar Juergens. Modern code reviews in open-source projects: Which problems do they fix? In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 202–211, New York, NY, USA, 2014. ACM.
7. Jennifer Bevan, E. James Whitehead, Jr., Sunghun Kim, and Michael Godfrey. Facilitating software evolution research with kenyon. In *Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ESEC/FSE-13, pages 177–186, New York, NY, USA, 2005. ACM.
8. Christian Bird, Adrian Bachmann, Eirik Aune, John Duffy, Abraham Bernstein, Vladimir Filkov, and Premkumar Devanbu. Fair and balanced?: Bias in bug-fix datasets. In *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC/FSE '09, pages 121–130, New York, NY, USA, 2009. ACM.
9. Remco Bloemen, Chintan Amrit, Stefan Kuhlmann, and Gonzalo Ordóñez Matamoros. Gentoo package dependencies over time. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 404–407, New York, NY, USA, 2014. ACM.
10. Marcel Bruch, Martin Monperrus, and Mira Mezini. Learning from examples to improve code completion systems. In *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC/FSE '09, pages 213–222, New York, NY, USA, 2009. ACM.
11. João Brunet, Gail C. Murphy, Ricardo Terra, Jorge Figueiredo, and Dalton Serey. Do developers discuss design? In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 340–343, New York, NY, USA, 2014. ACM.
12. Joshua Charles Campbell, Abram Hindle, and José Nelson Amaral. Syntax errors just aren't natural: Improving error reporting with language models. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 252–261, New York, NY, USA, 2014. ACM.
13. Ning Chen, Jialiu Lin, Steven C. H. Hoi, Xiaokui Xiao, and Boshen Zhang. Arminer: Mining informative reviews for developers from mobile app marketplace. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 767–778, New York, NY, USA, 2014. ACM.
14. Tse-Hsun Chen, Meiyappan Nagappan, Emad Shihab, and Ahmed E. Hassan. An empirical study of dormant bugs. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 82–91, New York, NY, USA, 2014. ACM.
15. Jean-Marc Davril, Edouard Delfosse, Negar Hariri, Mathieu Acher, Jane Cleland-Huang, and Patrick Heymans. Feature model extraction from large collections of informal product descriptions. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 290–300, New York, NY, USA, 2013. ACM.
16. Robert Dyer, Hoan Anh Nguyen, Hridesh Rajan, and Tien N. Nguyen. Boa: A language and infrastructure for analyzing ultra-large-scale software repositories. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 422–431, Piscataway, NJ, USA, 2013. IEEE Press.

17. Robert Dyer, Hridesh Rajan, Hoan Anh Nguyen, and Tien N. Nguyen. Mining billions of ast nodes to study actual and potential usage of java language features. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 779–790, New York, NY, USA, 2014. ACM.
18. Jayalath Ekanayake, Jonas Tappolet, Harald C. Gall, and Abraham Bernstein. Time variance and defect prediction in software projects. *Empirical Software Engineering*, 17(4-5):348–389, August 2012.
19. Mona Erfani Joorabchi, Mehdi Mirzaaghaei, and Ali Mesbah. Works for me! characterizing non-reproducible bug reports. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 62–71, New York, NY, USA, 2014. ACM.
20. Jon Eyolfson, Lin Tan, and Patrick Lam. Do time of day and developer experience affect commit bugginess? In *Proceedings of the 8th Working Conference on Mining Software Repositories*, MSR '11, pages 153–162, New York, NY, USA, 2011. ACM.
21. Gabriel Farah, Juan Sebastian Tejada, and Dario Correal. Openhub: A scalable architecture for the analysis of software quality attributes. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 420–423, New York, NY, USA, 2014. ACM.
22. Henning Femmer, Dharmalingam Ganesan, Mikael Lindvall, and David McComas. Detecting inconsistencies in wrappers: A case study. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 1022–1031, Piscataway, NJ, USA, 2013. IEEE Press.
23. Kenji Fujiwara, Hideaki Hata, Erina Makihara, Yusuke Fujihara, Naoki Nakayama, Hajimu Iida, and Kenichi Matsumoto. Kataribe: A hosting service of historage repositories. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 380–383, New York, NY, USA, 2014. ACM.
24. Takafumi Fukushima, Yasutaka Kamei, Shane McIntosh, Kazuhiro Yamashita, and Naoyasu Ubayashi. An empirical study of just-in-time defect prediction using cross-project models. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 172–181, New York, NY, USA, 2014. ACM.
25. Giacomo Ghezzi and Harald Gall. Replicating mining studies with SOFAS. In *10th Working Conference on Mining Software Repositories*, Washington, DC, 2013. IEEE Computer Society.
26. Giacomo Ghezzi and Harald C. Gall. SOFAS: A Lightweight Architecture for Software Analysis as a Service. In *Working IEEE/IFIP Conference on Software Architecture (WICSA 2011), 20-24 June 2011, Boulder, Colorado, USA*. IEEE Computer Society, 2011.
27. Giacomo Ghezzi and Harald C. Gall. A framework for semi-automated software evolution analysis composition. *International Journal on Automated Software Engineering*, 20(3):463–496, Sept 2013.
28. Emanuel Giger, Marco D'Ambros, Martin Pinzger, and Harald C. Gall. Method-level bug prediction. In *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '12, pages 171–180, New York, NY, USA, 2012. ACM.
29. Robert Gobeille. The fossology project. In *Proceedings of the 2008 International Working Conference on Mining Software Repositories, MSR 2008 (Co-located with ICSE), Leipzig, Germany, May 10-11, 2008, Proceedings*, pages 47–50, 2008.

30. Jesús M. González-Barahona and Gregorio Robles. On the reproducibility of empirical software engineering studies based on data retrieved from development repositories. *Empirical Software Engineering*, 17(1-2):75–89, 2012.
31. Georgios Gousios and Diomidis Spinellis. A platform for software engineering research. In *Proceedings of the 6th International Working Conference on Mining Software Repositories, MSR 2009 (Co-located with ICSE), Vancouver, BC, Canada, May 16-17, 2009, Proceedings*, pages 31–40, 2009.
32. Georgios Gousios, Bogdan Vasilescu, Alexander Serebrenik, and Andy Zaidman. Lean ghtorrent: Github data on demand. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 384–387, New York, NY, USA, 2014. ACM.
33. Georgios Gousios and Andy Zaidman. A dataset for pull-based development research. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 368–371, New York, NY, USA, 2014. ACM.
34. Lisong Guo, Julia Lawall, and Gilles Muller. Oops! where did that code snippet come from? In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 52–61, New York, NY, USA, 2014. ACM.
35. Monika Gupta, Ashish Sureka, and Srinivas Padmanabhuni. Process mining multiple repositories for software defect resolution from control and organizational perspective. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 122–131, New York, NY, USA, 2014. ACM.
36. Emitza Guzman, David Azócar, and Yang Li. Sentiment analysis of commit comments in github: An empirical study. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 352–355, New York, NY, USA, 2014. ACM.
37. Emitza Guzman and Bernd Bruegge. Towards emotional awareness in software development teams. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 671–674, New York, NY, USA, 2013. ACM.
38. Quinn Hanam, Lin Tan, Reid Holmes, and Patrick Lam. Finding patterns in static analysis alerts: Improving actionable alert ranking. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 152–161, New York, NY, USA, 2014. ACM.
39. Christopher L. Heffner. AllPsych: Research Methods, Chapter 1.11 Replication. `http://allpsych.com/researchmethods/replication.html`.
40. Lars Heinemann, Veronika Bauer, Markus Herrmannsdoerfer, and Benjamin Hummel. Identifier-based context-dependent api method recommendation. In *CSMR'12*, pages 31–40, 2012.
41. Kim Herzig, Sascha Just, and Andreas Zeller. It's not a bug, it's a feature: How misclassification impacts bug prediction. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 392–401, Piscataway, NJ, USA, 2013. IEEE Press.
42. Abram Hindle. Green mining: A methodology of relating software change to power consumption. In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pages 78–87, June 2012.
43. Abram Hindle, Alex Wilson, Kent Rasmussen, E. Jed Barlow, Joshua Charles Campbell, and Stephen Romansky. Greenminer: A hardware based mining software repositories software energy consumption framework. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 12–21, New York, NY, USA, 2014. ACM.

44. Reid Holmes and Gail C. Murphy. Using structural context to recommend source code examples. In *Proceedings of the 27th International Conference on Software Engineering*, ICSE '05, pages 117–125, New York, NY, USA, 2005. ACM.
45. Changyun Huang, Yasutaka Kamei, Kazuhiro Yamashita, and Naoyasu Ubayashi. Using alloy to support feature-based dsl construction for mining software repositories. In *Proceedings of the 17th International Software Product Line Conference Co-located Workshops*, SPLC '13 Workshops, pages 86–89, New York, NY, USA, 2013. ACM.
46. Laura Inozemtseva and Reid Holmes. Coverage is not strongly correlated with test suite effectiveness. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 435–445, New York, NY, USA, 2014. ACM.
47. Xiao-Yuan Jing, Shi Ying, Zhi-Wu Zhang, Shan-Shan Wu, and Jin Liu. Dictionary learning based software defect prediction. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 414–423, New York, NY, USA, 2014. ACM.
48. Brittany Johnson, Yoonki Song, Emerson Murphy-Hill, and Robert Bowdidge. Why don't software developers use static analysis tools to find bugs? In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 672–681, Piscataway, NJ, USA, 2013. IEEE Press.
49. Huzefa Kagdi, Michael L. Collard, and Jonathan I. Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of Software Maintenance and Evolution*, 19:77–131, 2007.
50. Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. The promises and perils of mining github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 92–101, New York, NY, USA, 2014. ACM.
51. Maria Kechagia and Diomidis Spinellis. Undocumented and unchecked: Exceptions that spell trouble. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 312–315, New York, NY, USA, 2014. ACM.
52. Iman Keivanloo. Online sharing and integration of results from mining software repositories. In *Software Engineering (ICSE), 2012 34th International Conference on*, pages 1644–1646, June 2012.
53. Iman Keivanloo, Christopher Forbes, Aseel Hmood, Mostafa Erfani, Christopher Neal, George Peristerakis, and Juergen Rilling. A linked data platform for mining software repositories. In *Mining Software Repositories (MSR), 2012 9th IEEE Working Conference on*, pages 32–35, June 2012.
54. Jacky Keung, Ekrem Kocaguneli, and Tim Menzies. Finding conclusion stability for selecting the best effort predictor in software effort estimation. *Automated Software Engg.*, 20(4):543–567, December 2013.
55. Katja Kevic and Thomas Fritz. A dictionary to translate change tasks to source code. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 320–323, New York, NY, USA, 2014. ACM.
56. Christoph Kiefer, Abraham Bernstein, and Jonas Tappolet. Mining software repositories with isparol and a software evolution ontology. In *Proceedings of the Fourth International Workshop on Mining Software Repositories*, MSR '07, pages 10–, Washington, DC, USA, 2007. IEEE Computer Society.
57. Nathan Klein, Christopher S. Corley, and Nicholas A. Kraft. New features for duplicate bug detection. In *Proceedings of the 11th Working Conference on Mining*

Software Repositories, MSR 2014, pages 324–327, New York, NY, USA, 2014. ACM.

58. Oleksii Kononenko, Olga Baysal, Reid Holmes, and Michael W. Godfrey. Mining modern repositories with elasticsearch. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 328–331, New York, NY, USA, 2014. ACM.
59. Daniel E. Krutz and Wei Le. A code clone oracle. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 388–391, New York, NY, USA, 2014. ACM.
60. Alina Lazar, Sarah Ritchey, and Bonita Sharif. Generating duplicate bug datasets. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 392–395, New York, NY, USA, 2014. ACM.
61. Alina Lazar, Sarah Ritchey, and Bonita Sharif. Improving the accuracy of duplicate bug report detection using textual similarity measures. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 308–311, New York, NY, USA, 2014. ACM.
62. Otávio A. L. Lemos, Adriano C. de Paula, Felipe C. Zanichelli, and Cristina V. Lopes. Thesaurus-based automatic query expansion for interface-driven code search. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 212–221, New York, NY, USA, 2014. ACM.
63. Chris Lewis, Zhongpeng Lin, Caitlin Sadowski, Xiaoyan Zhu, Rong Ou, and E. James Whitehead Jr. Does bug prediction support human developers? findings from a google case study. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 372–381, Piscataway, NJ, USA, 2013. IEEE Press.
64. Mario Linares-Vásquez, Gabriele Bavota, Carlos Bernal-Cárdenas, Massimiliano Di Penta, Rocco Oliveto, and Denys Poshyvanyk. Api change and fault proneness: A threat to the success of android apps. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 477–487, New York, NY, USA, 2013. ACM.
65. Mario Linares-Vásquez, Andrew Holtzhauer, Carlos Bernal-Cárdenas, and Denys Poshyvanyk. Revisiting android reuse studies in the context of code obfuscation and library usages. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 242–251, New York, NY, USA, 2014. ACM.
66. Nicholas Matragkas, James R. Williams, Dimitris S. Kolovos, and Richard F. Paige. Analysing the 'biodiversity' of open source ecosystems: The github case. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 356–359, New York, NY, USA, 2014. ACM.
67. Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E. Hassan. The impact of code review coverage and code review participation on software quality: A case study of the qt, vtk, and itk projects. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 192–201, New York, NY, USA, 2014. ACM.
68. Tim Menzies, Omid Jalali, Jairus Hihn, Dan Baker, and Karen Lum. Stable rankings for different effort models. *Automated Software Engineering*, 17(4):409–437, 2010.
69. Tim Menzies and Thomas Zimmermann. Software analytics: So what? *Software, IEEE*, 30(4):31–37, July 2013.

70. Thorsten Merten, Bastian Mager, Simone Bürsner, and Barbara Paech. Classifying unstructured data into natural language text and technical information. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 300–303, New York, NY, USA, 2014. ACM.
71. Leandro L. Minku and Xin Yao. How to make best use of cross-company data in software effort estimation? In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 446–456, New York, NY, USA, 2014. ACM.
72. Dimitris Mitropoulos, Vassilios Karakoidas, Panos Louridas, Georgios Gousios, and Diomidis Spinellis. The bug catalog of the maven ecosystem. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 372–375, New York, NY, USA, 2014. ACM.
73. Audris Mockus. Amassing and indexing a large sample of version control systems: Towards the census of public source code history. In *Proceedings of the 2009 6th IEEE International Working Conference on Mining Software Repositories*, MSR '09, pages 11–20, Washington, DC, USA, 2009. IEEE Computer Society.
74. Audris Mockus. Is mining software repositories data science? (keynote). In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 1–1, New York, NY, USA, 2014. ACM.
75. Manishankar Mondal, Chanchal K. Roy, and Kevin A. Schneider. Prediction and ranking of co-change candidates for clones. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 32–41, New York, NY, USA, 2014. ACM.
76. Hiroaki Murakami, Yoshiki Higo, and Shinji Kusumoto. A dataset of clone references with gaps. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 412–415, New York, NY, USA, 2014. ACM.
77. Alessandro Murgia, Parastou Tourani, Bram Adams, and Marco Ortu. Do developers feel emotions? an exploratory analysis of emotions in software artifacts. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 262–271, New York, NY, USA, 2014. ACM.
78. Meiyappan Nagappan, Thomas Zimmermann, and Christian Bird. Diversity in software engineering research. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 466–476, New York, NY, USA, 2013. ACM.
79. Jaechang Nam, Sinno Jialin Pan, and Sunghun Kim. Transfer defect learning. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 382–391, Piscataway, NJ, USA, 2013. IEEE Press.
80. Stas Negara, Mihai Codoban, Danny Dig, and Ralph E. Johnson. Mining fine-grained code changes to detect unknown change patterns. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 803–813, New York, NY, USA, 2014. ACM.
81. Hung Viet Nguyen, Hoan Anh Nguyen, Anh Tuan Nguyen, and Tien N. Nguyen. Mining interprocedural, data-oriented usage patterns in javascript web applications. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 791–802, New York, NY, USA, 2014. ACM.
82. Thanh H. D. Nguyen, Meiyappan Nagappan, Ahmed E. Hassan, Mohamed Nasser, and Parminder Flora. An industrial case study of automatically identifying performance regression-causes. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 232–241, New York, NY, USA, 2014. ACM.

83. Tung Thanh Nguyen, Hoan Anh Nguyen, Nam H. Pham, Jafar M. Al-Kofahi, and Tien N. Nguyen. Graph-based mining of multiple object usage patterns. In *Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ESEC/FSE '09, pages 383–392, New York, NY, USA, 2009. ACM.
84. Lucas Nussbaum and Stefano Zacchiroli. The ultimate debian database: Consolidating bazaar metadata for quality assurance and data mining. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*, pages 52–61, May 2010.
85. Joel Ossher, Sushil Krishna Bajracharya, and Cristina Videira Lopes. Automated dependency resolution for open source software. In *Proceedings of the 7th International Working Conference on Mining Software Repositories, MSR 2010 (Co-located with ICSE), Cape Town, South Africa, May 2-3, 2010, Proceedings*, pages 130–140, 2010.
86. Rohan Padhye, Senthil Mani, and Vibha Singhal Sinha. A study of external community contribution to open-source projects on github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 332–335, New York, NY, USA, 2014. ACM.
87. Leonardo Passos and Krzysztof Czarnecki. A dataset of feature additions and feature removals from the linux kernel. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 376–379, New York, NY, USA, 2014. ACM.
88. Gustavo Pinto, Fernando Castor, and Yu David Liu. Mining questions about software energy consumption. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 22–31, New York, NY, USA, 2014. ACM.
89. Daniel Pletea, Bogdan Vasilescu, and Alexander Serebrenik. Security and emotion: Sentiment analysis of security discussions on github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 348–351, New York, NY, USA, 2014. ACM.
90. Luca Ponzanelli, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Michele Lanza. Mining stackoverflow to turn the ide into a self-confident programming prompter. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 102–111, New York, NY, USA, 2014. ACM.
91. Martin F. Porter. An algorithm for suffix stripping. *Program: electronic library and information systems*, 14(3):130–137, 1980.
92. Sebastian Proksch, Sven Amann, and Mira Mezini. Towards standardized evaluation of developer-assistance tools. In *Proceedings of the 4th International Workshop on Recommendation Systems for Software Engineering*, RSSE 2014, pages 14–18, New York, NY, USA, 2014. ACM.
93. Dong Qiu, Bixin Li, and Zhendong Su. An empirical analysis of the co-evolution of schema and code in database applications. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 125–135, New York, NY, USA, 2013. ACM.
94. Foyzur Rahman, Sameer Khatri, Earl T. Barr, and Premkumar Devanbu. Comparing static bug finders and statistical prediction. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 424–434, New York, NY, USA, 2014. ACM.

95. Foyzur Rahman, Daryl Posnett, Israel Herraiz, and Premkumar Devanbu. Sample size vs. bias in defect prediction. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 147–157, New York, NY, USA, 2013. ACM.
96. Md Saidur Rahman, Amir Aryani, Chanchal K. Roy, and Fabrizio Perin. On the relationships between domain-based coupling and code clones: An exploratory study. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 1265–1268, Piscataway, NJ, USA, 2013. IEEE Press.
97. Mohammad Masudur Rahman and Chanchal K. Roy. An insight into the pull requests of github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 364–367, New York, NY, USA, 2014. ACM.
98. Beatrice Åkerblom, Jonathan Stendahl, Mattias Tumlin, and Tobias Wrigstad. Tracing dynamic features in python programs. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 292–295, New York, NY, USA, 2014. ACM.
99. Martin P. Robillard, Robert J. Walker, and Thomas Zimmermann. Recommendation systems for software engineering. *IEEE Software*, 27(4):80–86, July 2010.
100. Gregorio Robles, Jesús M. González-Barahona, Carlos Cervigón, Andrea Capiluppi, and Daniel Izquierdo-Cortázar. Estimating development effort in free/open source software projects by mining software repositories: A case study of openstack. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 222–231, New York, NY, USA, 2014. ACM.
101. Ripon K. Saha, Avigit K. Saha, and Dewayne E. Perry. Toward understanding the causes of unanswered questions in software information sites: A case study of stack overflow. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 663–666, New York, NY, USA, 2013. ACM.
102. Vaibhav Saini, Hitesh Sajnani, Joel Ossher, and Cristina V. Lopes. A dataset for maven artifacts and bug patterns found in them. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 416–419, New York, NY, USA, 2014. ACM.
103. Jelber Sayyad Shirabad and Tim J. Menzies. The PROMISE Repository of Software Engineering Databases. School of Information Technology and Engineering, University of Ottawa, Canada, 2005.
104. Matthias Schur, Andreas Roth, and Andreas Zeller. Mining behavior models from enterprise web applications. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 422–432, New York, NY, USA, 2013. ACM.
105. C. Shearer. The crisp-dm model: the new blueprint for data mining. *Data Warehousing*, 5:13–22, 2000.
106. Jyoti Sheoran, Kelly Blincoe, Eirini Kalliamvakou, Daniela Damian, and Jordan Ell. Understanding "watchers" on github. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 336–339, New York, NY, USA, 2014. ACM.
107. M. Shepperd and G. Kadoda. Comparing software prediction techniques using simulation. *Software Engineering, IEEE Transactions on*, 27(11):1014–1022, Nov 2001.
108. August Shi, Alex Gyori, Milos Gligoric, Andrey Zaytsev, and Darko Marinov. Balancing trade-offs in test-suite reduction. In *Proceedings of the 22Nd ACM SIG-*

SOFT International Symposium on Foundations of Software Engineering, FSE 2014, pages 246–256, New York, NY, USA, 2014. ACM.
109. Forrest J. Shull, Jeffrey C. Carver, Sira Vegas, and Natalia Juristo. The role of replications in empirical software engineering. *Empirical Softw. Engg.*, 13(2):211–218, April 2008.
110. Jacek Sliwerski, Thomas Zimmermann, and Andreas Zeller. When do changes induce fixes? In *Proceedings of the 2005 International Workshop on Mining Software Repositories, MSR*, 2005.
111. Jaime Spacco, Jaymie Strecker, David Hovemeyer, and William Pugh. Software repository mining with marmoset: An automated programming project snapshot and testing system. In *Proceedings of the 2005 International Workshop on Mining Software Repositories*, MSR '05, pages 1–5, New York, NY, USA, 2005. ACM.
112. K. Srinivasan and D. Fisher. Machine learning approaches to estimating software development effort. *Transactions on Software Engineering*, 21(2):126 – 137, Feb 1995.
113. Daniela Steidl, Benjamin Hummel, and Elmar Juergens. Incremental origin analysis of source code files. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 42–51, New York, NY, USA, 2014. ACM.
114. Igor Steinmacher, Igor Scaliante Wiese, Tayana Conte, Marco Aurélio Gerosa, and David Redmiles. The hard life of open source software project newcomers. In *Proceedings of the 7th International Workshop on Cooperative and Human Aspects of Software Engineering*, CHASE 2014, pages 72–78, New York, NY, USA, 2014. ACM.
115. Siddharth Subramanian, Laura Inozemtseva, and Reid Holmes. Live api documentation. In *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, pages 643–652, New York, NY, USA, 2014. ACM.
116. Rebecca Tiarks and Walid Maalej. How does a typical tutorial for mobile development look like? In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 272–281, New York, NY, USA, 2014. ACM.
117. Varun Tulsian, Aditya Kanade, Rahul Kumar, Akash Lal, and Aditya V. Nori. Mux: Algorithm selection for software model checkers. In *Mining Software Repositories (MSR)*. ACM, May 2014.
118. Yuriy Tymchuk, Andrea Mocci, and Michele Lanza. Collaboration in open-source projects: Myth or reality? In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 304–307, New York, NY, USA, 2014. ACM.
119. Harold Valdivia Garcia and Emad Shihab. Characterizing and predicting blocking bugs in open source projects. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 72–81, New York, NY, USA, 2014. ACM.
120. Lucian Voinea and Alexandru Telea. Mining software repositories with cvsgrab. In *Proceedings of the 2006 International Workshop on Mining Software Repositories*, MSR '06, pages 167–168, New York, NY, USA, 2006. ACM.
121. James R. Williams, Davide Di Ruscio, Nicholas Matragkas, Juri Di Rocco, and Dimitris S. Kolovos. Models of oss project meta-information: A dataset of three forges. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 408–411, New York, NY, USA, 2014. ACM.
122. Michael Würsch, Giacomo Ghezzi, Matthias Hert, Gerald Reif, and Harald Gall. Seon: A pyramid of ontologies for software evolution and its applications. *Computing*, 94(11):857–885, 2012.

123. Michael Würsch, Giacomo Ghezzi, Gerald Reif, and Harald C. Gall. Supporting developers with natural language queries. In *Proceedings of the 32nd International Conference on Software Engineering*. ACM, May 2010.
124. Michael Würsch, Emanuel Giger, and Harald Gall. Evaluating a query framework for software evolution data. *ACM Transactions on Software Engineering and Methodology*, 22(4):38–38, 2013.
125. Kazuhiro Yamashita. Modular construction of an analysis tool for mining software repositories. In *Proceedings of the 12th Annual International Conference Companion on Aspect-oriented Software Development*, AOSD '13 Companion, pages 37–38, New York, NY, USA, 2013. ACM.
126. Kazuhiro Yamashita, Shane McIntosh, Yasutaka Kamei, and Naoyasu Ubayashi. Magnet or sticky? an oss project-by-project typology. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 344–347, New York, NY, USA, 2014. ACM.
127. Andy Zaidman, Bart Van Rompaey, Arie van Deursen, and Serge Demeyer. Studying the co-evolution of production and test code in open source and industrial developer test processes through repository mining. *Empirical Software Engineering*, 16(3):325–364, 2011.
128. Marcelo Serrano Zanetti, Ingo Scholtes, Claudio Juan Tessone, and Frank Schweitzer. Categorizing bugs with social networks: A case study on four open source software communities. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 1032–1041, Piscataway, NJ, USA, 2013. IEEE Press.
129. Motahareh Bahrami Zanjani, George Swartzendruber, and Huzefa Kagdi. Impact analysis of change requests on source code based on interaction and commit histories. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 162–171, New York, NY, USA, 2014. ACM.
130. Chenlei Zhang and Abram Hindle. A green miner's dataset: Mining the impact of software change on energy consumption. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 400–403, New York, NY, USA, 2014. ACM.
131. Feng Zhang, Audris Mockus, Iman Keivanloo, and Ying Zou. Towards building a universal defect prediction model. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 182–191, New York, NY, USA, 2014. ACM.
132. Hongyu Zhang, Liang Gong, and Steve Versteeg. Predicting bug-fixing time: An empirical study of commercial software projects. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 1042–1051, Piscataway, NJ, USA, 2013. IEEE Press.