University of Magdeburg

Faculty of Computer Science

Master's Thesis

Analyzing Software Evolution Datasets and Their Use Cases

Author:

Sebastian Kittan

27. February 2023

Advisors:

Prof. Dr. rer. nat. Gunter Saake

Department of Technical and Business Information Systems

Dr.-Ing. David Broneske

German Centre for Higher Education Research and Science Studies

Dr.-Ing. Jacob Krüger

Eindhoven University of Technology

Kittan, Sebastian:
Analyzing Software Evolution Datasets and Their Use Cases
Master's Thesis, University of Magdeburg, 2023.

Abstract

Sharing research artifacts (e.g., software, data, protocols) is an immensely important topic for improving transparency, replicability, and reusability in research (i.e., open science), and has recently gained more and more traction in software engineering. For instance, recent studies have focused on artifact reviewing, the impact of open science, and specific legal or ethical issues of sharing artifacts. Most of such studies are concerned with artifacts created by the researchers themselves and processes for quality assuring these artifacts (e.g., through artifact-evaluation committees). In contrast, the more specific practices and challenges (e.g., technical, ethical, legal) of sharing software-evolution datasets (i.e., republished version control data) have only been scratched in such studies. To tackle this gap, we report a qualitative literature analysis of software-evolution datasets published at the International Conference on Mining Software Repositories from 2017 until 2021 and papers that build upon these datasets. By investigating 200 papers, we elicit what types of software-evolution datasets are shared, what use cases the datasets are (intended to be) used for, and what challenges researchers experience with sharing as well as using such datasets. Building on the results, we discuss challenges of, and propose recommendations for, sharing software-evolution datasets in a way that avoids typical problems (e.g., technical limitations, ethical concerns, data privacy). Our results extend and complement current research, and we are confident that it helps future researchers share software-evolution datasets in a reliable and trustworthy way.

Contents

List of Figures

List of Tables

1. Introduction

Sharing research artifacts has become a vital concern of most researchers in any field, and has gained major attention in software-engineering research, too [14, 15, 44, 83, 129, 131, 212, 213]. Making artifacts (e.g., software, data, protocols) used for a piece of research available promises many benefits for the research community. For instance, a shared artifact allows researchers to easily reuse it rather than needing to re-implement it, helps to validate the corresponding findings, and builds trust by contributing to open-science practices. Consequently, many researchers have started to investigate the practices, challenges, and benefits of artifact sharing within software engineering (cf. Chapter 7). In parallel, conferences (e.g., the Joint European Software Engineering Conference and Symposium on the Foundations on Software Engineering [165]), journals (e.g., Empirical Software Engineering [213]), publishers (e.g., the Association for Computing Machinery[1]), and funding institutions (e.g., the European Union [295]) are pushing for more open science, for instance, by introducing reviews, awards, badges, or legal frameworks for sharing research artifacts. Despite such efforts and a general agreement on the pros of open science, artifact sharing also faces critique, for example, because of the effort it takes to review artifacts, vague definitions of badges, or enforcing open-science on unfit types of artifacts (e.g., confidential data).

Interestingly, most artifact-sharing practices focus on the accessibility and reusability of artifacts created by researchers themselves (e.g., software, data measured during experiments). In contrast, other aspects of how artifacts are shared have often been neglected, for instance, what format an artifact is stored in, whether an artifact fulfills legal requirements (e.g., data protection, licenses), or ethical concerns like data privacy [17, 90, 92]. These aspects are particularly relevant when sharing data not created by the researchers themselves. A primary example for such data is version-control data extracted from software repositories, which is extensively used in empirical studies, such as mining studies, case studies, or benchmarks. Software repositories (e.g., from Git, GitHub, BitBucket) exhibit various types of data on the evolution of a software system, for example, source code, developer names, mail addresses, natural-language comments, commits, pull requests, or documentation. The diversity of this data poses several challenges to researchers when sharing and reusing software-evolution datasets (i.e., data extracted from a software repository's version-control and associated systems):

How to best organize the data (e.g., relational database versus csv files) in a way that enables others to reuse it (e.g., not requiring high-performance computer clusters)? How to ensure data privacy and anonymity (e.g., developer names, callouts in natural-language comments)? What is allowed to share under what license and legal requirements (e.g., data protection)? Is the dataset representative and feasible for the use case the research is concerned with (e.g., are bug reports somehow

[1]https://www.acm.org/publications/policies/artifact-review-and-badging-current

linked to code changes)? To support the ongoing research in this direction and integrate such aspects into current practices, a more profound understanding of how software-evolution datasets are and should be shared is required.

Therefor we want to answer the following research question:

RQ.1 What are the practices of sharing software-evolution datasets?

RQ.2 What are the originally intended and actually performed use cases for software-evolution datasets?

RQ.3 What challenges have researchers experienced with sharing and using software-evolution datasets?

In this thesis, we contribute to this understanding by reporting an in-depth literature analysis of 200 papers that have shared, modified, or used software-evolution datasets. For this purpose, we started collecting 43 relevant datasets and mining-challenge papers from recent iterations (2017–2021) of the International Conference on Mining Software Repositories (MSR), a prime venue for software-evolution research that has such dedicated tracks. We forwards-snowballed through the citing papers to obtain a broader picture of how the datasets have (not) been reused and what challenges researchers experienced while doing so. Building on a qualitative analysis of all papers, we contribute the following:

- We contribute an overview of 41 shared software-evolution datasets and their properties.
- We compare typical use cases for which software-evolution datasets are shared and used.
- We present a classification of the challenges of creating, sharing, and reusing software-evolution datasets.
- We discuss the previous three contributions to provide an understanding of current practices and challenges for researchers.
- We publish our own dataset of the 200 papers we analyzed in a persistent open-access repository.

Our results show the diversity of software-evolution datasets and their use cases, even though this is only a subtype of all software-engineering datasets. We highlight and discuss various problems regarding the sharing of software-evolution datasets (e.g., data formats, topicality). The consequent conflicts are important for researchers to consider when sharing, but also when reviewing and using, such datasets. Overall, we hope that our contributions help the community improve their artifact sharing of software-evolution artifacts, and thus increase the extent of open-science practices.

These remainder of the thesis is structured as follows. In Chapter 2 we describe the basic topics that are required for the continuation of the thesis. Next we specify in

Chapter 3 our methodology, that we employ to study our research question. Within Chapter 4 we present our data in unprocessed form and then in Chapter 5 we report the results we obtained for each of our research question. Then we discus this result in Chapter 6. Finally, we present in Chapter 7 the related work and then we finish in Chapter 8 with a conclusion and future work.

2. Background

In this chapter we describe the technicalities of concepts that are needed for the following chapters. First we give an overview why datasets are important. Then we show what software evolution is and the connection with Version Control Systems (VCS). Finally we describe how the artifact sharing is related to them.

2.1 Datasets

Nowadays everybody comes in contact with mountain of information. This data influences our everyday life, as data reflects knowledge. This is especially true for Software Engineers (SE). They need this data to do their day to day work. One of their task is to handle this data in form of information. Here to get a better overview and insight of the data they are using dataset as a representation of the data. This allows the SE to present the data in a more understandable way for themselves, colleagues or customers. As written in [255] organized data allows SE to measure data and take appropriate action. Good data provides SE indisputable evidence to taking action based on. This includes finding the problems, solving the problems and monitoring the data for emerging problems. Most of the time SE monitors data and explore new data. To meet these goals SE organize the data in datasets. These are data, which are collected and managed for a specific purpose.

In this work we deal with software-evolution datasets. These datasets are created from version control system (VCS) data. SE always endeavors more insight of this data, because most of the time developer are using this system to program the solution of the SE and to get it they analyze VCS data and organize them in software-evolution datasets to share them with other specialists in this field. The goal is to discuss the purpose of the created dataset and to further develop their knowledge.

The data generated by SE are also important information for all other software engineering related topics and therefore interface too many other important stakeholders.

- The project management needs the data and the data evolution for effort estimations and progress monitoring.
- The requirements engineer needs the data to track down the implementation of requirements.
- The contractor manager needs the data to integrate contractors into the project e.g. for interface information.
- The integrator needs the data evolution to update the build.

- The test engineer needs the data to relate software modules to requirements for test case creation and coverage as well as the data evolution for regression tests.

All this specialist need different views on the SE data and this means that, depending on the target group, specialized data sets are required and have to be created.

In summary dataset are important tools to organize data and present them to other specialist of the field, to understand correlations and abstract difficult data to easier to understand datasets.

2.2 Software-Evolution

As described by geeksforgeeks[1] Software-evolution is a process to develop software initially and then update in time steps to add features or remove functionality. The process of evolution includes topics of fundamental activities of change analysis, release planning, system implementation and releasing a system to customers. Fist is the cost of a change weighed against the feature that is to change, if this is accepted then the change of the system are designed, implemented and tested and released. After that the cycle starts again until the system is obsolete.

Software-evolution is necessary, because of the following reasons:

1. Change in requirement with time: Over time the organizations need and operations change and so must the software change.
2. Environment change: Over time the tools such as programming languages and other things change and the system adopt to the new environment.
3. Errors and bugs: With the age of deployed software their preciseness or impeccability decrease and the efficiency to bear the increasing complexity workload also continually degrades. In order to counteract this, old and no longer required software has to be evolved.
4. Security risks: Over time new software based cyberattacks are found. So it became necessary to avoid security breaches with assessments of the software and follow up patches of this security breaches.
5. For having new functionality and features: In order to implement new features or increase the performance, the organization need to continuously evolve the software to remain competitive with other products.

2.3 Version Control Systems

“Version control system (VCS) allows you to track the iterative changes you make to your code.”[32] VCS are important tools for SE that work in groups and are

[1]https://www.geeksforgeeks.org/software-engineering-software-evolution/

located in different locations. It also helps the developer of software to efficiently communicate and track the changes that have been made by a specific developer. The most popular VCS are Git[2], standalone or as Git server with GitHub or GitLab, or Mercurial.

The basics of VCS are repositories that encompasses the entire collection of files and folders associated with a project, along with each file's revision history. A developer that want to work on the project creates a personal copy of the files in the project and changes the files. To add this changes to the project the developer commits this changes to the repository. When the developer wants to get the version of the project on another time of the project he must update his local copy with one of the commits in the repository.

In the type of distributed VCS that we use mainly in this work, the repository is saved by a Git server service as GitHub. Multiple developer use separate local repository to commit too and update his local working copy. To save the changes from the local repository to the server repository the developer must push them to the server repository. Then all other developer can also use this changes of the developer after they pull the changes from the server repository to their local repository and update their local working copy. See this in Figure 2.1.

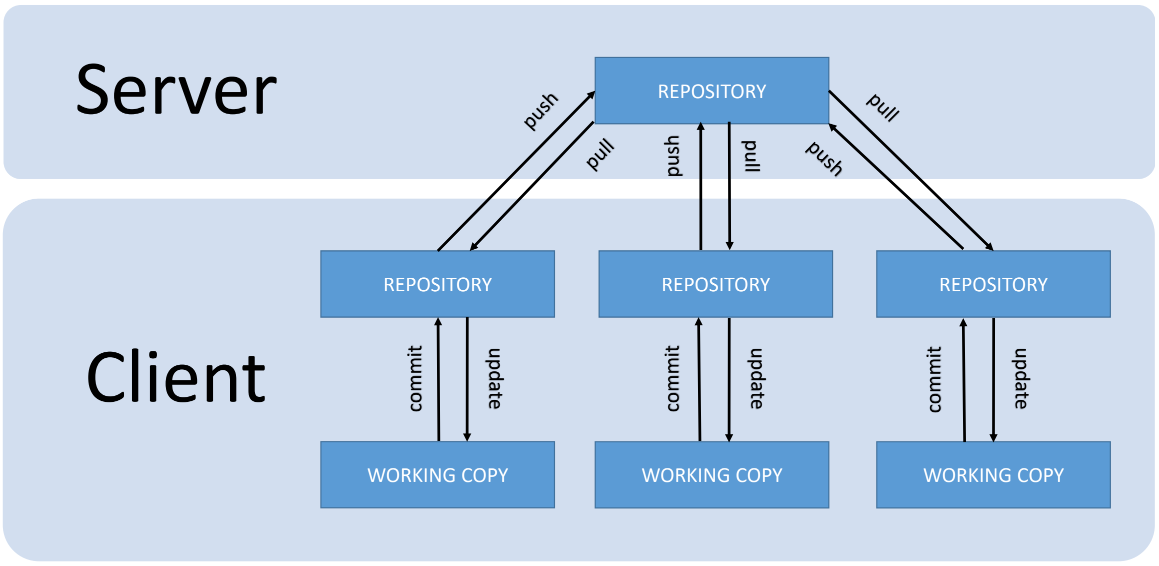

Figure 2.1: Distributed Version Control System[3]

There can be branches, means different changes of the same file. Each branch is represented by a pointer on the last committed file version. With only one branch the pointer points to the last commit and this branch is named master. Developer can create multiple branches, for example to create a version of the project that is used in production and a new branch that is used for developing and testing. The

[2]https://git-scm.com
[3]https://www.geeksforgeeks.org/version-control-systems/

developer can change the branch and merge pull request branches for example when new features should be added in the development branch to the production branch.

When a new group of developer want to add features to a existing repository then they can fork a repository. This create a new copy of the repository for the new group, were they can add features that they want. Later they can choose to pull with a pull request individual feature to the original repository or let it be a standalone repository.

2.4 Artifact Sharing

Software artifacts are every thing that isn't the code. This includes according to techtarget[4]:

- Code related artifacts that act as foundation of the software and tests such as compiled code, setup scripts, test suits and the logs of this test.
- Project management artifacts that ensure to fulfill the functionality such as minimum required standards, benchmarks, project vision statements, roadmaps, change logs, scope management plans and quality plans.
- Documentation artifacts that keep track of relevant documents such as diagrams, end-user agreements, internal documentation or written guides.

With the ever increasing amount of data and software products, it has become more and more important to improve the software development process and keep up with the times. This is exactly why artifact sharing has become more and more important, as it simplifies the software-evolution process and makes it more efficient by making it more structured and traceable. First of all, the creation of artifacts requires additional time, which can be recovered and improved by the increased traceability and thus reduced resources (manpower and time). It is in the interest of all organizations and researchers to share artifacts so that everyone in the organization can work with them and researchers to find more efficient ways to create and share them.

[4]https://www.techtarget.com/searchsoftwarequality/definition/artifact-software-development

3. Methodology

In this chapter of this work we describe our explained research questions and the methodology we used to address these. We display the steps of our methodology in Figure 3.1.

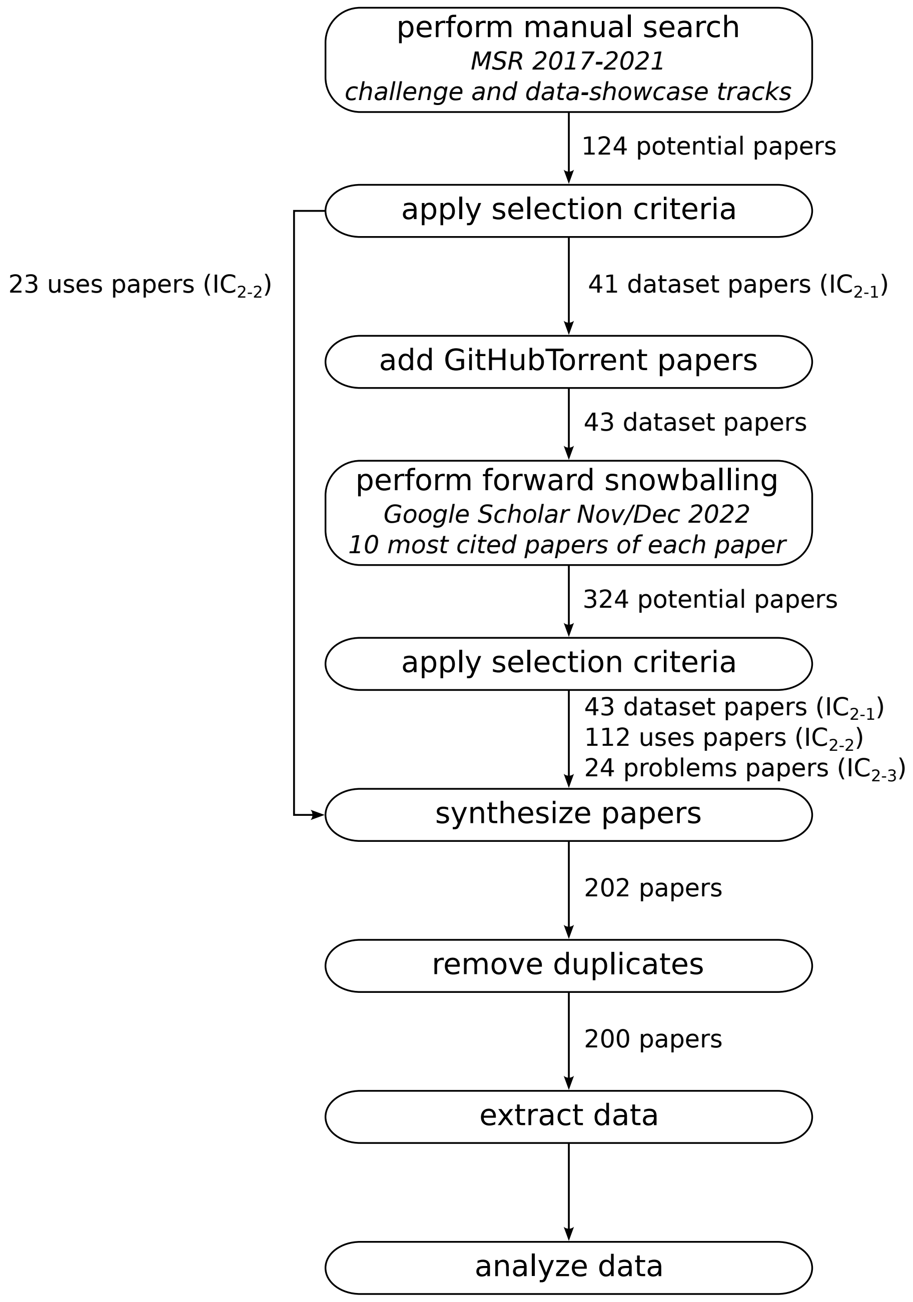
perform manual search
MSR 2017-2021
challenge and data-showcase tracks
124 potential papers
apply selection criteria
23 uses papers (IC$_{2-2}$)
41 dataset papers (IC$_{2-1}$)
add GitHubTorrent papers
43 dataset papers
perform forward snowballing
Google Scholar Nov/Dec 2022
10 most cited papers of each paper
324 potential papers
apply selection criteria
43 dataset papers (IC$_{2-1}$)
112 uses papers (IC$_{2-2}$)
24 problems papers (IC$_{2-3}$)
synthesize papers
202 papers
remove duplicates
200 papers
extract data
analyze data

Figure 3.1: Overview of our methodology.

3.1 Research Questions

While artifact sharing has become a widely established practice in research and is actively investigated (cf. Chapter 7), there are still many challenges associated to it. A notable challenge is how datasets handle the involvement of VCS information about software evolution and their developers. This topics had rarely be studied and there exist several challenges like copyrighting, anonymity, ethics, or feasible data storing. Our goal in this study is to shed light into current practices as well as challenges of sharing software-evolution datasets.

For this intended purpose we have developed 3 research questions (RQ):

RQ.1 **What are the practices of sharing software-evolution datasets?**

With this first RQ, we aim to discover datasets that are shared recently at a high-quality venue (i.e. dataset paper that where accepted and have thus gone through a peer review). With this purpose in mind, we decided to conduct a manual search through five years of Mining Software Repositories Conference (MSR) (cf. Section 3.2). Instead of using a survey to research what is claimed to be done, we performed a literature analysis of actually published datasets to know what is actually done. We want to contribute an overview of current practices for sharing software-evolution datasets, by analyzing the properties (e.g., storage format, size, where published, types of data) of these datasets.

RQ.2 **What are the originally intended and actually performed use cases for software-evolution datasets?**

In this second RQ, we aim to discover the sharing researcher's specified use cases of their software-evolution datasets and compare them to the use cases that are actually used by other researchers. For this purpose, we extended our literature search with a snowballing phase to identify additional papers that (try to) use one of the datasets we collected before (cf. Section 3.2). We want to contribute an overview of typical use cases for software-evolution datasets and show to what extent the sharing researchers can anticipate in advance for what analyses their datasets may be used for.

RQ.3 **What challenges have researchers experienced with sharing and using software-evolution datasets?**

In the last RQ we aim to link the different challenges researchers report on sharing as well as using software-evolution datasets. For this purpose, we analyzed our two datasets from the literature search and the snowballing using open-coding and open-card sorting to identify common topics. That mean we classify significant statements and then we group them into different categories. We want to contribute an overview of typical challenges of publishing software-evolution datasets and what concerns should be taken into account for new datasets.

These three RQs provide the context for the results we aim to achieve with our literature review (cf. Chapter 5). We then proceed to discuss these results in combination (cf. Chapter 6) to contribute a detailed understanding of practices, challenges, and recommendations connected to sharing software-evolution datasets.

3.2 Literature Search

To identify relevant papers for answering our RQs, we performed a two-step process based on guidelines and recommendations for searching papers for literature reviews and mapping studies in software engineering [36, 141, 158, 159, 167, 298, 344].

First, we performed a manual search through the mining-challenge and dataset tracks of the MSR, which are dedicated tracks at a high-quality conference for publishing (particularly software-evolution) datasets and solving challenges associated to them.

Second, we performed a snowballing search on all relevant papers from MSR. Thus, we extended our analysis of the research community and collected particularly publications that used the software-evolution datasets we identified in the first step. Next, we describe our selection criteria before explaining these two steps of our search in more detail.

3.2.1 Selection Criteria

To select feasible papers, we defined three inclusion criteria (ICs):

- IC_1 The paper is written in English.
- IC_2 The paper (non-exclusive OR)
  - $IC_{2\text{-}1}$ shares a software-evolution dataset (e.g., version-control data, issues, pull requests); or
  - $IC_{2\text{-}2}$ uses a software-evolution dataset and does not just mention it, for instance, in the related work; or
  - $IC_{2\text{-}3}$ describes or analyzes why software-evolution datasets could not be reused.

We remark that we also considered non-peer-reviewed papers (e.g., master theses, technical reports), since these are often longer and comprise more technical details. As a consequence, such papers helped us elicit problems of using software-evolution datasets that are often omitted in space-restricted peer-reviewed papers. Regarding IC_2, we had to identify whether a publication shares a software-evolution dataset ($IC_{2\text{-}1}$); uses such a dataset, for instance, to evaluate a new technique ($IC_{2\text{-}2}$); or analyzes the problems of using a dataset, for example, by attempting to reuse it or comparing datasets ($IC_{2\text{-}3}$). Note that sharing in this context can also mean that an existing dataset has been modified and reshared, in which case a publication would fulfill both $IC_{2\text{-}1}$ and $IC_{2\text{-}2}$. We elicited papers fulfilling $IC_{2\text{-}3}$ to ensure that we do not only capture finally successful attempts of reusing a dataset, but also challenges (with the dataset) that prevented other researchers from that reuse.

3.2.2 Manual Literature Search

As our first step, we analyzed papers published at the MSR mining-challenge and dataset tracks. MSR is being one of the premier venues for research on software-evolution and sharing corresponding datasets. MSR is ranked A in the CORE[1] (Computing Research & Education) ranking since 2018 and the following years. The conference is only one rank under a flagship conference, hence it is an excellent conference and that is highly respected in the discipline area. We decided to perform a manual search through MSR, because

1. MSR has such dedicated tracks, in contrast to other flagship software-engineering venues (e.g., International Conference on Software Engineering, International Conference on Automated Software Engineering, Joint European Software Engineering Conference and Foundations on Software Engineering);

2. the primary topics of MSR relate to software-evolution data, which is why we expect high levels of expertise, quality, and best practices when sharing software-evolution datasets; and

3. a manual search through dedicated MSR tracks promises a high ratio of relevant and high-quality papers, in contrast to an automated search that faces technical problems, is hard to replicate, and yields many irrelevant or low-quality publications.

Both tracks we considered involve papers that share software-evolution datasets and the challenge track also involves papers reusing these datasets (i.e., solutions tackling the proposed challenge). As a consequence, we argue that MSR provides an ideal starting point for eliciting papers that are relevant to address our RQs, essentially following a similar search strategy as Gold and Krinke [90, 92] to focus our literature search and avoid the problems of automated searches (e.g., replicability, technical problems).

To identify relevant papers, we manually inspected the MSR entries in dblp.[2] Precisely, we analyzed papers from 2017–2021, which we considered a feasible time span, covering current practices (compared to older papers) while also providing time for datasets to be reused (compared to newer papers from 2022 or 2023). We then identified all papers that are part of either the dataset or mining-challenge (proposal and solution) tracks.

Unfortunately, the naming in dblp is inconsistent with these track names, for instance, in 2019 the dataset-track papers are listed under the publisher-provided categories “representations for mining” and “large-scale mining.” To handle such inconsistencies, we verified that we identified the right and all papers of these tracks against the MSR conference website of each respective year.

Of the 124 papers in these five years and two tracks, we considered 64 relevant according to our ICs (i.e., 41 papers based on $IC_{2\text{-}1}$ and 23 according to $IC_{2\text{-}2}$).

Year	Track	Papers		
		all	Datasets ($IC_{2\text{-}1}$)	Uses ($IC_{2\text{-}2}$)
2017	ds	7	4	0
	mc	15	1	13
2018	ds	15	9	0
	mc	14	0	0
2019	ds	11	2	0
	mc	15	0	0
2020	ds	19	14	0
	mc	4	1	2
2021	ds	16	10	0
	mc[1]	8	0	8
	Total	124	41	23

ds: data showcase – mc: mining challenge

[1] Note that the challenge case of this year [150] had already been published as a data showcase at MSR 2020, and thus is included in our dataset.

Table 3.1: Number of papers we manually elicited from MSR.

We show an overview of the number of papers we included for each year and track in Table 3.1.

As we can see, focusing on the two MSR tracks yielded the rather high ratio (51.61 %) of relevant papers we hoped for, drastically facilitating our search compared to an automated one. Moreover, it was easier to identify papers that did not fulfill our ICs, for instance, the challenge cases (and consequent solutions) of MSR 2018 (developer activities in IDEs [265]) and 2019 (SOTorrent [19]) did not build on software-evolution datasets. During an exploratory analysis of the papers, we noted that many papers refer and sometimes use parts of the GHTorrent dataset [99, 101]. Since this dataset has been widely used in software-engineering research, we decided to add the two papers relating to it into our analysis, even though these papers do not fulfill our ICs. So, we ended up with a total of 126 papers, 43 of which share a software-evolution dataset.

3.2.3 Snowballing

In November and December 2022, we performed a forwards snowballing using Google Scholar to extend our dataset. Snowballing is a technique to increase a pool of elements with further elements starting from previous ones. Particularly, we aimed to identify more papers that use one of the shared datasets and that report on the challenges of doing so. To limit the effort of this process, we elicited for each of the 43 dataset papers we collected the ten most cited papers at that point in time. Since not all datasets have been cited at least ten times, we ended up with 324 snowballed papers.

[1]https://www.core.edu.au/conference-portal

[2]https://dblp.org/db/conf/msr/index.html

We then checked whether these ten publications fulfill our ICs, particularly whether they use ($IC_{2\text{-}2}$) and potentially re-share ($IC_{2\text{-}1}$) a dataset or analyze problems of using the dataset ($IC_{2\text{-}3}$). After this step, we ended up with 136 new papers, 112 that use a dataset and 24 that do not use a dataset but analyze or report problems that prevent such a use. So, after both searches, we considered 448 papers, of which we included 200 as relevant to answer our RQs (44.64 %).

3.3 Data Extraction and Analysis

We used a spreadsheet to collect all papers and their bibliographic information (e.g., authors, publication year, title, and venue). Then, we performed an open-coding process in which the first author (alone to ensure consistency) extracted relevant statements and data from each paper into individual text documents.

Based on our RQs, we defined the following data as relevant:

- A non-exclusive *categorization* for each paper, namely if it shares a software-evolution dataset ($IC_{2\text{-}1}$), uses such a dataset ($IC_{2\text{-}2}$), or is relevant due to analyzing problems of such datasets ($IC_{2\text{-}3}$).
- The *dataset* itself and its properties, for which we also studied the actual dataset (RQ.1):
  - The name of the dataset.
  - The types of data included in the dataset (e.g., repository metadata, developer behavior).
  - The platform where it has been shared (e.g., Zenodo, GitHub).
  - The storage format and potentially supported queries (e.g., relational database, csv files).
  - The size of the dataset.
- The *use cases* proposed (when sharing) or researched (when using) for a dataset (RQ.2).
- The *problems* of sharing or using a dataset as reported by the authors of a paper (RQ.3).

After extracting this data, we performed an open card-sorting-like process [376] to triangulate common themes from the extracted data to answer our RQs.

During this process, we revisited the individual papers and datasets to check the correctness and level of detail of the data extraction. After agreeing on the final themes, we added these into our spreadsheet to complete our dataset.

4. Data

In this chapter, we present the raw/exact data in order to be able to assign accurate values to the datasets. The synthesis of the data is presented in the Chapter 5. In the following sections we give for every section the needed data to answer the 3 RQ.

In the Section 4.1 we collected an overview of the datasets and their characteristics to answer RQ.1. In the Section 4.2 we collect data to answer RQ.2. Therefor we identified the proposed use cases of the dataset researchers and then the real world use cases of the snowballed publications. In the Section 4.3 we identified the possible problems that the researcher have predicted and the actual problems in the snowballed publication to answer RQ.3.

4.1 Dataset Characteristics

In this section we describe the dataset and their characteristics. There are 41 datasets and 43 papers that describe them. Two datasets are described by 2 papers. For each paper from the Section 3.2.2 we extracted bibliographic data, namely the title, the authors, the publisher and the publication year. In addition we extracted significant data for our RQ.1:

- the nickname of the title (that we will use from now on and the following tables) and a reference to the paper in the bibliography
- the data source or sources of the dataset described in the papers
- the host and the number of cites
- the data format of the data that it is published
- quantitative data of the dataset

In addition we extracted the availability of every dataset.

In the table (cf. Table 4.1) are the results of the work:

Title	Source	Host, Cites	Data Format	Quantity
GE526 [323]	Git CLI, GitHub	osf.io, 0	• .csv	• 526 repositories • metadata, 582.079 commits, 20.138 pull requests, 30.287 issues reports and 2.111 releases
Qscored [300]	GitHub	zenodo.org, 2	• PostgreSQL database	• 86.652 repositories

Title	Source	Host, Cites	Data Format	Quantity
Duets [72]	GitHub	github.com, 3	• folder structure with .json, .xml, .log • has dataset, project and commits	• 395 Libraries • 2.874 Clients
Wonderless [76]	GitHub	zenodo.org, 3	• .csv	• 1.877 repositories
Andror2 [340]	GitHub	zenodo.org, 3	• folder structure (with information to reproduce as metadata, apk, scripts, code and issues • has bug reports as .html	• 90 manually reproduced bug reports
Quantifying [66]	GitHub	zenodo.org, 5	• MongoDB data dump	• GitHub projects 3.000 • Commits 3.948.945 • Commit Comments 41.099 • Issues 819.031 • Issue Comments 1.898.101 • Issue Events 2.300.490 • Contributors 62.597 • Lines of Code Analyzed 5.216.361.494 • database size 125GB (19.8GB compressed)
JTeC [49]	GitHub	zenodo.org, 3	• .csv	• JTeC (Java Test Classes), provides 2,5M+ test classes collected from a set of 31K+ projects
Android Compass [226]	Git, GitHub	zenodo.org, 4	• .csv	• 80.324 individual single-line code changes of Android compatibility checks and their respective meta data, which we collected from 1.394 projects of the F-Droid catalog
Denchmark [155]	GitHub	github.com, 4	• .xlsx	• 193 Project
Andromeda [237]	Ansible Galaxy, GitHub	figshare.com, 3	• .yaml	• Andromeda, a dataset consisting of four parts: • Metadata 140K entities spread across seven entity types • Git commit and tag metadata from more than 25K repositories containing roles • Structural models for over 125K role versions • Distilled changes between structural model versions, totaling more than 800K concrete changes categorized into 41 change types

Title	Source	Host, Cites	Data Format	Quantity
Mixed Graph-Relational [11]	GitHub, Jira	zenodo.org, 4	• MySQL database • Neo4J graphs	• 20 open source projects
Linux Kernel [351]	Linux kernel mailing list, Git	zenodo.org, 7	• MySQL 5.7 dump	• Patch: 666.550 • Author: 9.243 • Patch linked to commits: 139.664 • Comment to patch: 1.208.320 • Patch set: 136.139
Enterprise-Driven [308]	[101]	zenodo.org, 7	• 2 txt files	• 17.264 record tab-separated
Shoulders of Giants [365]	[103],[99]	zenodo.org, 5	• .csv • .SQL	• 11.230 projects comprising of 3.347.937 pull requests • 96 features
LogChunks [35]	Travis,[101]	zenodo.org, 6	• .xml • .csv	• 797 annotated Travis CI build logs from 80 GitHub repositories
20-MAD [48]	Apache's list of Git repositories, Apache's Jira	osf.io, 6	• Apache Parquet files(column-oriented data file format)	• 765 projects, 3.4M commits, 2.3M issues, and 17.3M issue comments
Many Types4Py [217]	GitHub	zenodo.org, 7	• zip file • .json for Python projects • with projects' URL and their latest commit hash an duplicate files in the dataset • .csv list of files and their corresponding set	• 5.382 Python projects with more than 869K type annotations
Software evolution [352]	Subversion (SVN), later Git, GitLab	opendata. soccerlab. polymtl.ca, 8	• .Xsl/.xlsx	• 6 projects
SHGD [260]	GitHub, GitLab, Debian, and PyP	zenodo.org, 9	• same as [259]	• same as [259]
Dockerfiles [127]	GitHub	zenodo.org, 11	• Abstract Syntax Trees (ASTs) with .json	• 178.000 unique Dockerfiles with structured representations
Andro ZooOpen [187]	GitHub, [5]	knowledgezoo .xyz, 17	• knowledge graph (announced by Google in May 2012)	• 46.521 app repositories

Title	Source	Host, Cites	Data Format	Quantity
Git Repositories [219]	[101, 197]	github.com, 14	• ultimateMap2.s (gzipped semicolon-separated list of repositories • ghForks.gz produce each forked repository, its ultimate parent	• Set the megacluster of [197] from 14 mio repositories to under 400k
Repository Deduplication [309]	[101]	zenodo.org, 15	• 2 txt files with tab-separated records	• 10.649.348 records mapping a duplicated source project to a definitive target project • forks_clones_noise_names is a 50.324.363 member superset of the source projects, containing also projects that were excluded from the mapping as noise.s
Semantic Changes [372]	GitHub	github.com, 16	• .yaml	• 81 semantic change data from 8 open source Java projects
50K-C [206]	[99]	mondego.ics.uci.edu, 21	• 3 .tgz as projects (source code), .jar (dependencies), buildresults (metadata) and mapping .txt	• 50.000 compilable Java projects
Sampling Projects [52]	GitHub	zenodo.org, 23	• API: .json, .xml, .csv • data dump: .csv, .sql	• 25 characteristics (e.g., number of commits, license, etc.) of 735,669 repositories
OCL expressions [233]	GitHub	github.com, 24	• original meta-models and the generated abstract syntax trees	• 9.188 OCL expressions originating from 504 EMF meta-models in 245 systematically selected GitHub repositories
Duplicate Pull-Requests [359]	GitHub	github.com, 26	• MySQL database dump	• 2.323 pairs of duplicate Pull Requests, collected from 26 popular open source projects
Identity Resolution [84]	[197]	zenodo.org, 25	• compressed .csv with ‘;’ as the separator for duplication and second .csv for characteristic	• 5.427.024 commit authors
CROP [242]	Gerrit(git)	github.io, 30	• three main directories: Metadata (.csv), Git Repositories and Discussion (.txt)	• 8 repositories • 48.975 reviews and 112.617 patches

Title	Source	Host, Cites	Data Format	Quantity
Structured inf [293]	GitHub, Google Big-Query, Docker-Hub	github.com, 29	• PostgreSQL database archive	• over 100.000 unique Dockerfiles in over 15.000 GitHub projects
Git Archive [204]	[99]	github.com, 34	• Siva files with Git repositories • Index file (.csy)	• 182.014 top-bookmarked Git repositories
A C_C++ Code Vuly [78]	CVE database, GitHub	github.com, 42	• .csv	• 3.754 code vulnerabilities spanning 91 different vulnerability types are extracted from 348 Github projects
SHG [259]	GitHub, Git-Lab.com, Debian, and PyP	zenodo.org, 39	• encoded in the dataset as a set of relational tables—roughly, one for each node type with different sources • PostgreSQL database dump in CSV format • Apache Parquet files • public dataset on Amazon Athena	• spans more than 5 billion unique source code files • one billion unique commits, coming from more than 80 million software projects
UML models [277]	[101]	oss.models-db.com, 48	• 2 .csv (UMLFiles List, Project FileTypes) as relational database	• over 93.000 UML diagrams from over 24,000 projects
VulinOSS [88]	nvd, Git, Mercurial, and Subversion	github.com, 52	• Mysql database dump	• Projects: 153 • Project versions: 23.884 • Mapped Versions: 8.694 • Number of Vulnerabilities: 17.738 • Project Versions with Testing Code: 38.650 • Project Version employing CI: 1.538
Android apps [86]	GitHub	github.io, 55	• Neo4j graph database • repositories cloned to a local GitLab instance	• 8.431 real open-source Android apps
World of code [197]	GitHub, Bit-bucket, GitLab, Source-Forge and other git	bitbucket.org, 62	• C database library called TokyoCabi-net split in 24 (hashed)	• 5.313.256.585 blob (considered by git to be text files) • 1.419.161.099 commit • 5.786.313.329 tree • 9.518.401 tag

Title	Source	Host, Cites	Data Format	Quantity
Many SStuBs4J [150]	[99]	zenodo.org, 84	• .json, .csv	• 100 Java Maven Project Bugs • 1.000 Java Project Bugs bugsLarge.json • 100 Java Maven Project SStuBs sstubs.json • 1.000 Java Project SStuBs sstubsLarge.json • Top 100 Ranked Java Maven Projects topJavaMavenProjects.csv • Ranked Java Projects topProjects.csv
Bugs.jar [284]	GitHub	github.com, 129	• every bug instance is in one branch of the GitHub repository (tree structure)	• 1.158 bugs and patches, drawn from 8 large, popular open source Java projects
Travis Torrent [25]	Travis CI, [99]	travistorrent.testroots.org, 175	• in-browser SQL shell to run their queries on our infrastructure • download SQL dumps or the compressed data set as a CSV file (1.8 GB unpacked)	• 1.300 projects • 691,184 builds
GHTorrent [101]	GitHub	ghtorrent.org, 297	• Bit-Torrent peer-to-peer protocol as incremental database • Every month one MongoDB data dump • later MySQL as .csv file and offline dblite querry on website	• Commits: 8.817.685 • Events: 4.512.000 • Repositories: 424.995 • Committers: 303.470

Title	**Source**	**Host, Cites**	**Data Format**	**Quantity**
The GHTorent [99]	GitHub	ghtorrent.org, 676	• same as [101]	• Events: 43.090.195 • Projects (repositories): 1.326.900 • Users: 793.855 • Project_members (repo_collabs): 34.924 • Organization_members (org_members):34.924 • Commits: 29.978.291 • Watchers: 7.744.61ß • Followers: 1797.343 • Issues: 2.326.069 • Issue_events: 4.085.294 • Issue_comments: 2.886.006 • Pull_requests: 1.144.251 • Pull_request_comments (pull-req_comnts): 2.228.894

Table 4.1: Characteristics of the datasets with, bibliographic data, source, host, cites, data format and quantity

In summary the 43 papers that describe 41 datasets have many similarity like the host as zenodo and github, but differences like the citation and quantity exist. More detailed analyzes are shown in Section 5.1.

4.2 Dataset Use Cases

For each found paper in Section 3.2.2 we identified the use cases that the researcher have described in the paper for the datasets and sorted them to types of use cases. The types are software evolution, quality issue detection and repair, process, study quality, system analyse and other analyses. A more detailed description is shown in Section 5.2.

The results are shown in Section 4.2 as below:

Title	**Use Cases**	**Type Use Case**
GE526 [323]	• software evolution data • release engineering • natural language processing for developer sentiment	• software evolution
Qscored [300]	• Prediction, Maintenance Code Smells/Bugs • Maintenance	• software evolution • quality issue detection and repair
Duets [72]	• software evolution API liberties and client • quality (test generation, changed liberties not break clients, repair)	• software evolution • quality issue detection and repair
Wonderless [76]	• software evolution Serverless application • antipattern analysis	• software evolution • system analyse
Andror2 [340]	• NL processing to reproduce crashes • fault localization	• quality issue detection and repair
Quantifying [66]	• Behavior extraction(issue labeling and resolve time, issue resolving as fast as possible helpful) • software evolution	• software evolution • process
JTeC [49]	• static analysis (tests smells, pattern, code refactoring) • test case generation • benchmark(Regression technics)	• software evolution • quality issue detection and repair • process
Android-Compass [226]	• Automated Program Repair • Benchmark	• quality issue detection and repair
Denchmark [155]	• locating and fixing DLSW bugs	• quality issue detection and repair
Andromeda [237]	• analyzing the structure and evolution of role implementations	• software evolution
Mixed Graph-Relational [11]	• developer interaction • socio-technical congruence in micro-service architecture	• developers • process
Linux Kernel [351]	• collecting, cleaning, and processing patch data	• software evolution
Enterprise-Driven [308]	• involvement enterprises in OSS development • OSS business model and supply, value chain	• process
Shoulders of Giants [365]	• collaborative environment • software maintenance • development process • human factor in computing systems	• developers • process • quality issue detection and repair
LogChunks [35]	• Build analysis	• quality issue detection and repair

Title	Use Cases	Type Use Case
20-MAD [48]	• natural language processing • repository mining tasks (that need daily, weekly developer pattern, sentiment analysis or NLP)	• developers
Many Types 4Py [217]	• ML-based type inference • Learning-based code completion	• quality issue detection and repair
Software evolution [352]	• quality issue detection and repair • software evolution • developers	• defect prediction • analyzing software evolution • developer activities
SHGD [260]	• Cross-repository analysis • Cross-origin analysis	• software evolution
Dockerfiles [127]	• program repair • Dockerfile analysis of software dependencies	• quality issue detection and repair
Andro-ZooOpen [187]	• fixes of bugs (Android) • energy anti-patterns and performance bottlenecks • security vulnerabilities, compatibility issues and code smells • evolution of open-source Android apps	• quality issue detection and repair • software evolution • system analyse
Repositories [219]	• community detection algorithms	• software evolution • developers
Repository Deduplication [309]	• repeat studies with deduplication projects	• study quality
Semantic Changes [372]	• Semantic History Slicing • benchmark for semantic history slicing • dynamic feature location	• system analyse
50K-C [206]	• studies that need mapped static source code and runnable components	• quality issue detection and repair
Sampling Projects [52]	• support researchers in sampling projects from GitHub	• study quality
OCL expressions [233]	• measuring frequency of OCL constructs and called operations • limiting a threat to validity of another study	• other analyses • software evolution • study quality
Duplicate Pull-Requests [359]	• redundant/duplicate Pull request detection • software evolution of duplicates • developer activities spanning across redundant contributions	• software evolution • developers • process
Identity Resolution [84]	• identity resolution	• developers
CROP [242]	• SE research • code review influence build, tests • process of patch quality over submissions • review transfer between developer	• process • software evolution
Structured information [293]	• software evolution Dockerfiles • quality (build quality quantity, documentation connection with StackOverflow and GitHub • build analysis	• quality issue detection and repair • software evolution

Title	Use Cases	Type Use Case
Git Archive [204]	• statistical machine learning and natural language processing on source code • automatic naming suggestion , program prediction , topic modeling and semantic clustering , bug detection, and automated software transpiration • inter-project source code clone detection	• software evolution • quality issue detection and repair • system analyse
A C/ C++ Code Vulnerabilities [78]	• vulnerability (characteristic, code changes, detection, repair)	• quality issue detection and repair • software evolution
SHG [259]	• Cross-repository analysis • Cross-origin analysis	• software evolution
UML models [277]	• pro and contra UML • UML use	• other analyses
VulinOSS [88]	• relation between bugs and CI	• quality issue detection and repair
Android apps [86]	• "automatically analyzing, understanding, reproducing, localizing and fixing bugs	• quality issue detection and repair
World of code [197]	• Cross-ecosystem comparison studies • Python ecosystem analysis • Correcting Developer Identity Errors • Repository filtering tool	• software evolution • developers • study quality
Many-SStuBs4J [150]	• bug localization (how and when) • evaluation of program repair technique • SSTubs common, spotted by tools, test cover	• quality issue detection and repair • software evolution
Bugs.jar [284]	• benchmark suite (automated debugging, patching, and testing) • techniques for testing(coverage, selection, generation, anomaly detection, repair, etc.) • bug (localization, quality, etc.)	• quality issue detection and repair
Travis-Torrent [25]	• with CI: • more quality with faster bug localization? • fewer test regression? • successful projects more CI runs and tests? • Compatible with CD? • Failed builds effect developer? • Development models: "holy grail" or embrace breaking and fixing	• software evolution • quality issue detection and repair • developers • process
GHTorrent [101]	• community dynamics • global software engineering • distributed collaboration • code authorship and attribution	• software evolution
The GHTorent [99]	• Unified developer identities • Software ecosystems • Network analysis • Collaboration and promotion • Replications of existing studies	• developers • software evolution • study quality

Table 4.2: Specific use cases from shared researcher about their datasets and grouping in types

Then as described in Section 3.2.3 we snowballed on the base of the datasets. There we differentiated between publication that applied the dataset on problems and publications that not applied the dataset on problems (e.g. where the dataset is only mentioned in the introduction or related work). After that we picked the 135 publication (112 from Section 3.2.3 and 23 from Section 3.2.2) that are applied and analyzed their use cases. There are datasets that have no publications that applied them in our 10 snowballed publications, that are GE526 [323], Qscored [300], Andror2 [340], Quantifying [66], Shoulders of Giants [365], LogChunks [35], Software evolution [352], Identity Resolution [84], Structured information [293], UML models [277] and VulinOSS [88].

In the Table 4.3 there is shown the title of the dataset, the publications that applied the dataset, the related use cases and their type of use case. The type of the use case is determined by the best matching type determined by us, for example ManySStuBs4J [150] has the publication, Mea culpa: How developers fix their own simple bugs differently from other developers [374], that the quality and developers addressed, but we decided it treats the side developer more. The publications that use the software-evolution datasets as a collection of subject systems, for instance, for evaluating techniques for scheduling in serverless software systems, we marked them as "subject systems in study unrelated to software evolution" for short utse.

The use cases for the publications are shown in the following:

Dataset	**Used by**	**Use Case**	**Type Use Case**
Duets [72]	[329]	• visualizing library dependencies for change impact analysis	• system analyse
Wonderless [76]	[363]	• topology-based scheduling	• utse
	[55]	• topology-based scheduling	• utse
JTeC [49]	[336]	• relation of test automation maturity with product quality	• process
Android-Compass [226]	[228]	• program repair, misuse detection	• quality issue detection and repair
	[188]	• benchmark on bug localization	• quality issue detection and repair
Denchmark [155]	[156]	• bug localization	• quality issue detection and repair
Andromeda [237]	[238]	• empirical study on semantic versioning	• system analyse
	[236]	• Smelly Variables: Detection, Prevalence, and Lifetime	• quality issue detection and repair
	[235]	• code smell prevalence	• utse
Mixed Graph-Relational [11]	[12]	• evolution of developer communities	• developers
Linux Kernel [351]	[313]	• developer communication during patch submitting	• process
Enterprise-Driven [308]	[87]	• relation of software reuse and security	• utse
	[8]	• analysis of security activities in continuous integration pipelines	• process
	[342]	• cryptographic misuses	• quality issue detection and repair

Dataset	Used by	Use Case	Type Use Case
	[330]	• software vulnerability, benchmark	• quality issue detection and repair
20-MAD [48]	[170]	• Benchmark on identifying technical debt (change dependent)	• quality issue detection and repair
	[336]	• Empirical study on the impact of test automation on continuous integration activities	• quality issue detection and repair
	[211]	• new technique for link recovery between issues and commits	• quality issue detection and repair
Many Types 4Py [217]	[252]	• type inference	• utse
	[180]	• knowledge graph creation	• utse
	[107]	• evaluation of a deep learning-based type inference system	• utse
	[194]	• evaluation of a unit test creation tool with regression testing	• process
SHGD [260]	[29]	• analyzing software evolution (Cross-platform forking)	• software evolution
	[9]	• analyzing software security awareness (how fast developers identify/discuss security problems)	• developers
	[257]	• improving software-evolution dataset analyses (deduplication, server side tool, graph compression)	• improve dataset
	[337]	• data analysis performance	• utse
	[338]	• data analysis performance	• utse
Dockerfiles [127]	[128]	• program repair	• quality issue detection and repair
	[264]	• misconfiguration localization and repair and benchmark for misconfiguration localization	• quality issue detection and repair
AndroZoo-Open [187]	[268]	• malicious software detection	• quality issue detection and repair
	[267]	• static analyze detect malicious software	• quality issue detection and repair
	[319]	• malware detection	• quality issue detection and repair
	[358]	• subject systems not related to software evolution (test script documentation)	• utse
Git Repositories [219]	[309]	• creating a new software-evolution dataset	• new dataset
Repository Deduplication [309]	[356]	• developer collaboration and awarding of contributions	• developers
	[308]	• creating a new software-evolution dataset	• new dataset
	[310]	• empirical study of software evolution (lifespan of fine grained changes)	• software evolution
	[196]	• developer identification	• developers
Semantic Changes [372]	[176]	• feature location	• system analyse
	[346]	• eliciting software-change patterns to manage software evolution	• software evolution
	[373]	• semantic slicing	• process
	[177]	• semantic slicing	• process
50K-C [206]	[286]	• variable naming	• utse
	[152]	• benchmarking AI	• utse

Dataset	Used by	Use Case	Type Use Case
Sampling Projects [52]	[46]	• code completion	• utse
	[262]	• software licensing	• utse
	[96]	• detection of bots in software evolution	• bot detection
	[53]	• create other dataset	• new dataset
	[333]	• programming language usage	• process
	[56]	• empirical analysis of continuous integration tooling	• quality issue detection and repair
	[106]	• coding style	• utse
	[45]	• improving regression testing	• process
	[234]	• program repair	• quality issue detection and repair
OCL expressions [233]	[294]	• analyze of model-driven artefacts	• utse
	[253]	• Optical Character Recognition	• utse
	[214]	• difference industrial and open source OCL	• utse
Duplicate Pull-Requests [359]	[274]	• Identifying development redundancy(training and test dataset)	• process
	[178]	• Identifying development redundancy	• process
	[335]	• Identifying development redundancy(training and test dataset)	• process
	[179]	• Identifying development redundancy(new approaches and evaluation on dataset)	• process
	[155]	• creating another software-evolution dataset	• new dataset
CROP [242]	[245]	• empirical study on software evolution - refactoring in code reviews	• process
	[243]	• empirical study on software evolution - impact of code reviews on design changes	• process
	[320]	• empirical study on software evolution - predicting design impactful changes	• process
	[249]	• empirical study on software evolution - code smells	• quality issue detection and repair
	[244]	• empirical study on software evolution - rebasing in code reviews	• process
	[321]	• empirical study on software evolution - code review on software degradation	• process
Git Archive [204]	[95]	• analyze code clone and license violations	• utse
	[74]	• transformer models	• utse
	[205]	• identifier identification	• utse
A C/C++ Code Vulnerabilities [78]	[175]	• vulnerability detection	• utse
	[42]	• vulnerability detection	• utse
	[232]	• vulnerability injection	• utse
	[273]	• program repair	• quality issue detection and repair
	[43]	• creating a new software-evolution dataset	• new dataset
SHG [259]	[361]	• impact of gender on code contributions	• developers
	[282]	• analyzing code provenance	• software evolution
	[33]	• improving techniques/guidelines for software evolution analysis	• improve dataset
	[281]	• analyzing code provenance	• software evolution
	[82]	• improving techniques/guidelines for software evolution analysis	• improve dataset

Dataset	Used by	Use Case	Type Use Case
Android apps [86]	[89]	• empirical study of quality change when introducing new programming language	• process
	[287]	• empirical study of versioning of APIs	• software evolution
	[224]	• subject systems not on software evolution (API recommender)	• utse
World of code [197]	[61]	• bot identification	• bot detection
	[84]	• identity resolution	• developers
	[62]	• bot identification	• bot detection
	[57]	• evolution of developer communities	• developers
	[219]	• empirical study of repository relationships	• software evolution
	[60]	• predictor of pull request acceptance	• software evolution
Many-SStuBs4J [150]	[220]	• empirical study on the evolution of bugs	• quality issue detection and repair
	[169]	• study on the impact of CI on bugs	• quality issue detection and repair
	[207]	• program repair	• quality issue detection and repair
	[199]	• program repair	• quality issue detection and repair
	[374]	• difference if bug fixed by orginal and other developer	• developers
	[149]	• occurrences of bugs in Python	• quality issue detection and repair
	[254]	• occurrences of bugs in test code	• quality issue detection and repair
	[138]	• vulnerability detection	• utse
	[314]	• program repair	• quality issue detection and repair
	[151]	• language models	• utse
	[345]	• program repair	• quality issue detection and repair
	[208]	• AI, as training set	• utse
Bugs.jar [284]	[174]	• program repair	• quality issue detection and repair
	[71]	• program repair	• quality issue detection and repair
	[200]	• creating a new software-evolution dataset	• utse
	[285]	• program repair	• quality issue detection and repair
Travis-Torrent [25]	[239]	• studying integration testing	• quality issue detection and repair
	[225]	• predicting testing efforts/time	• process
	[306]	• sentiment analysis for CI	• system analyse
	[140]	• CI build failures	• quality issue detection and repair
	[203]	• development overhead of CI	• process
	[270]	• software developer/development	• developers
	[241]	• non functional requirements and CI	• process
	[303]	• social attributes for commit success	• developers
	[31]	• predicting testing efforts/time	• process

Dataset	**Used by**	**Use Case**	**Type Use Case**
	[110]	• developer attraction and retention due to CI	• developers
	[85]	• software development	• software evolution
	[223]	• CI integration errors	• quality issue detection and repair
	[201]	• predicting defects	• quality issue detection and repair
	[24]	• CI build failures	• quality issue detection and repair
	[362]	• software development usage analyse tools	• software evolution
	[328]	• program repair benchmark	• quality issue detection and repair
GHTorrent [101]	[145]	• perils and promise of GitHub mining	• improve dataset
	[100]	• study development processes	• process
	[2]	• code summarization	• utse
	[325]	• study development processes	• process
The GHTorent [99]	[145]	• perils and promise of GitHub mining	• improve dataset
	[100]	• study of development process	• process
	[327]	• quality and continuous integration	• quality issue detection and repair
	[105]	• study of development process	• process
	[326]	• gender diversity in developer communities	• developers
	[133]	• understanding CI practices	• quality issue detection and repair
	[221]	• improve software evolution analytics	• software evolution
	[111]	• sentiment analysis	• system analyse

Table 4.3: Specific use cases from real world publications that applied the dataset and grouping in types

4.3 Problems

In this part we present the problems/challenges of the analyzed datasets. First we identify the predicted problems of the datasets and then we collect the real-world problems of the datasets and mark connections between them.

At first we identified in Table 4.4 the limitations or problems that the authors of the datasets describe. For each paper of the datasets, we ordered the paper to the predicted problems and issues that the dataset may have according to the researchers. Entries with null mean that no specific problems were named.

Title	Predicted Issues
GE526 [323]	• game engines repositories that are not in this topic but game engines are not represented • quality dataset obtained only from stargazers count and fork count metrics • likelihood of unintentional noisy data
Qscored [300]	• repositories only written in Java and C#
Duets [72]	• null
Wonderless [76]	• Wonderless is restricted to the applications developed with Serverless Framework, not every developer uses a framework to program Serverless applications • possibility of uninteresting cases, including toy software and stub applications
Andror2 [340]	• null
Quantifying [66]	• null
JTeC [49]	• null
Android-Compass [226]	• null
Denchmark [155]	• some bugs exist without test files in the fixing commit and some bug reports are linked to changed test files (35 %)
Andromeda [237]	• auxiliary files or other types of content for the roles(tests, plugin in Python) are not captured by the models we build • other IaC languages with similar ecosystems as Galaxy such as Puppet's PuppetForge3, and Chef's Supermarket4 exist and can be used to construct similar datasets
Mixed Graph-Relational [11]	• Mapping between Jira and GitHub component not absolutely certain • other communication channel used such as mailing list
Linux Kernel [351]	• Recognizing the multiple identities of a single author and recovering relationships between sub-patches in a patch set can never be accomplished perfectly
Enterprise-Driven [308]	• null
Shoulders of Giants [365]	• builds on decade of research on pull based development and inherit limitation of the feature they used • data sources have different level of abstraction, that can lead to differences in the outcome for more representation of the dataset there can be code-related metrics which otherwise are found not important for decision making and less explored or metrics that cannot be studied objectively

Title	**Predicted Issues**
LogChunks [35]	• bigger dataset • Build failure cause add • more metadata
20-MAD [48]	• timestamps are incomplete, there are no timezone information • identity merging performed rudimentary • NLoN's prediction model hasn't been retrained with any Apache data
Many-Types-4Py [217]	• Cannot parse Python 2
Software evolution [352]	• only use medium-sized, Java-based, three layered architecture, web-based, information systems • only individual task edited, work not useful for collaborative environments • tasks resemble backlog items in a single sprint/iteration within the Agile context • use 14 years old technology • tools are not available anymore • no explicit corrective tasks • not straightforward, which commit counts to which task • not all the commit logs were associated with an issue
SHGD [260]	• not reproducible because data source started of 2015 • cannot claim full coverage of the entire software commons and several forges and package repositories are still missing from it
Dockerfiles [127]	• challenges and techniques are, in theory, applicable to a wide range of DevOps artifacts, the dataset consists solely of Dockerfiles • single source: GitHub • possible that other DevOps artifacts are not as amenable to the ideas we present
Andro-ZooOpen [187]	• other open-source Android app repositories that do not come with the Android topic and thereby overlooked by our approach • open-source Android apps may not only be hosted on GitHub
Git Repositories [219]	• not investigating finer types of relationships, for example, light forks done for a single pull request vs hard forks where projects evolve independently or all shades of grey in between • accuracy of our approach is not easy to establish
Repository Deduplication [309]	• null
Semantic Changes [372]	• extending the dataset require that the projects have well-organized version controlled histories (so that there is a clear way of identifying a particular functionality) and have corresponding test cases • can be extended to include project histories containing known bugs and failed test cases that manifest the buggy behaviors
50K-C [206]	• null
Sampling Projects [52]	• SEART GitHub search engine (seart-ghs) does not offer an overview of the historic evolution of said characteristics
OCL expressions [233]	• limitations of the search functionality of GitHub • only files smaller than 384 KB are searchable • GitHub search covers only repositories with fewer than 500,000 files
Duplicate Pull-Requests [359]	• Only small size • Rules maybe incomplete

Title	Predicted Issues
Identity Resolution [84]	• author has no first last name only alias • method not clear to work with foreigner names • blank names make method inefficient • ID that are shared by many people(organizational IDs or admin IDs) • size of test sample is small(check of correctness)
CROP [242]	• null
Structured information [293]	• null
Git Archive [204]	• not up to date because of GHTorrent • Selecting repositories based on the number of stargazers is arguable and may introduce bias
A C/ C++ Code Vulnerability [78]	• rows related to the Chrome project miss out some descriptive information of some CVEs, e.g., the CVE IDs, CWE IDs, etc
SHG [259]	• not reproducible because data source get from start of 2015 • cannot claim full coverage of the entire software commons, and several forges and package repositories are still missing from it
UML models [277]	• Don't cover all possible file types for UML • GitHub is dynamic so repository may have been deleted • Filter not desired project out e.g. student projects files identified as UML are false positives
VulinOSS [88]	• threat to the internal validity of our dataset construction could be the limited analysis that our worker processes perform to identify test code
Android apps [86]	• mined Google Play only in researcher region • only mined GitHub • Resorting to a heuristic approach for matching Google Play listings to GitHub repositories entails the risk of mismatches
World of code [197]	• only used git and no older version control systems • no guarantee it closely approximates the entirety of public version control systems as the project discovery procedure is only an approximation • Reproducibility may pose an issue in a constantly updated database it is a concern how reliably clean, correct, integrate, and augment the collected data are, because of the • tradeof with the performance of the analytic layer
ManySStuBs4J [150]	• possible to extract a pair of aligned statements that are unrelated • not useful for evaluating whether repair systems are good at fixing larger bugs • restricted to Java but could be replicated for other languages by using a parser and creating a module that checks if an AST pair fits any of the SStuB patterns
Bugs.jar [284]	• For automatic program repair, we needed to run test cases using JUnit APIs • using APIs was tricky for subjects with many dependent libraries since correctly specifying configurations, such as classpaths was considerably more difficult with the APIs • instrumentation was an issue for a subject such as Log4J2 due to library conflicts
Travis-Torrent [25]	• null
GHTorrent [101]	• To reconstruct API schema need initial state for events that is not there • plan to automate the generation and distribution of torrent files through RSS feeds and scripts that will monitor those and automatically download and update remote databases

Title	Predicted Issues
The GHTorent [99]	• Data is additive and deletions are not reported • Important entities are not timestamped • Creates a fake user entry, when commit user can not been resolved • Pull requests merged outside GitHub • Issue tracking is open ended • During the lifetime of the project, the commit entry schema changed twice, while the watchers entity has been renamed to stargazers • Some events may be missing (Malfunctions in the mirroring system (software or network))

Table 4.4: Predicted challenges from the sharing researchers

In the Table 4.5 we showed the categorized problems that the datasets have in the real-world (analyzed paper of the snowballing). Therefor we collected the problems and grouped them to categories. This is then presented by specifying if the dataset belongs to one of the categories.

The seven categories are:

1. *Faulty or Invalid Data*

   To these problem count datasets that have incorrect metadata (e.g. timestamps).

2. *Redundant Data*

   To these problem count datasets that use ”toy” projects such as student projects, ”example” projects such as server blueprints, not useful code/data for the user of the dataset and forked data (most don’t want to have forked repositories in datasets).

3. *Quantity and Reliability*

   To these problem count datasets that have not enough ground truth or are not representable for the problem. Most of the time it is not big enough or respected in the community, so additional datasets or queries from the original data source are needed for supplementation.

4. *Topicality*

   To these problem count datasets that the researcher from the snowballing want an updated version of but the dataset is not updated with new edition of the dataset or a periodic update such as GHTorrent.

5. *Missing data*

   To these problem count datasets that need for the studies that are based on them more available attributes that can possible be useful e.g. version information of bug reports [155].

6. *Accessibility*

   To these problem count datasets that have complicated structures for the dataset or high hardware challenges for researchers.

7. *Others*

   To these problem count datasets which is not one of the others e.g. the problem of no common tool support by Semantic Changes [372]

Title	**1**	**2**	**3**	**4**	**5**	**6**	**7**
GE526							
Qscored							✓
Duets			✓				
Wonderless		(✓)	✓				
Andror2							
Quantifying							
JTeC							✓
AndroidCompass			✓	✓			
Denchmark					✓		
Andromeda							✓
Mixed Graph-Relational							
Linux Kernel							
Enterprise-Driven			✓	✓			
Shoulders of Giants							
LogChunks		✓					
20-MAD		✓			✓		
ManyTypes4Py		✓		✓	✓		
Software evolution							
SHG						✓	
Dockerfiles				✓	✓		
AndroZooOpen							✓
Git Repositories							
Repository Deduplication			✓	✓			
Semantic Changes			✓		✓		✓
50K-C							
Sampling Projects		✓	(✓)				
OCL expressions	✓	✓			✓		
Duplicate Pull-Requests	✓				✓		
Identity Resolution							
CROP		✓					
Structured information							
Git Archive	✓	✓	✓	(✓)	✓	✓	
Code Vulnerability			✓			✓	
SHG	✓	✓				✓	
UML models			✓				
VulinOSS					✓		
Android apps	✓		✓	✓	✓		
World of code	✓	✓					
ManySStuBs4J	✓	✓	✓	✓	✓	✓	
Bugs.jar			✓		✓		
TravisTorrent		✓	✓	✓	✓	✓	
GHTorrent	(✓)						
The GHTorent			✓	✓			

1: Faulty or Invalid Data, 2: Redundant Data, 3: Quantity and Reliability, 4: Topicality, 5: Missing data, 6: Accessibility, 7: Others

Table 4.5: Grouped challenges

It is to note that in the Software Heritage Graph Dataset, ManySStuBs4J and TravisTorrent datasets there are more problems/challenges, this can partly be due to the fact that there are more analyzed paper, because we analyzed the 10 paper of the snowballing and the paper from the manual search. Furthermore the embraced entries means, that the predicted problem is a real-world problem, that was worth mentioning.

5. Result

In this section we analyze and discuss the results for each of our research question individually. We use the data from Chapter 4 and arrange them according to the related research questions.

5.1 RQ.1: Dataset Sharing

For RQ.1, we are concerned with how software-evolution datasets are shared. To this end, we investigated (cf. Section 3.3) the datasets themselves, where they are shared with the use of the previous Chapter 4, the included data, and how they are stored (i.e., formats, size).

5.1.1 Identified Software-Evolution Datasets

Type	**Datasets**	**#**
Repository metadata	20-MAD [48], Andromeda [237], CROP [242], Dockerfiles [127], Duplicate Pull-Requests [359], Enterprise-Driven [308], GE526 [323], Git Archive [204], Git Repositories [219], GHTorrent [99, 101], Linux Kernel [351], ManyTypes4Py [217], OCL expressions [233], Quantifying [66], Repository Deduplication [309], Sampling Projects [52], Semantic Changes [372], Shoulders of Giants [365], Software Heritage Graph Dataset [259, 260], Structured information [293], UML models [277], Wonderless [76], World of code [197]	23
Software quality	Andror2 [340], Bugs.jar [284], C/C++ Code Vulnerabilities [78], Denchmark [155], ManySStuBs4J [150], QScored [300], Software evolution [352], VulinOSS [88]	8
Human factors	CROP [242], Enterprise-Driven [308], Identity Resolution [84], Linux Kernel [351], Mixed Graph-Relational Dataset [11], Software evolution [352]	6
Testing and deployment	50K-C [206], Duets [72], JTeC [49], LogChunks [35], TravisTorrent [25]	5
Mobile apps	Android apps [86], AndroidCompass [226], Andror2 [340], AndroZooOpen [187]	4

Table 5.1: Overview of the 41 datasets we identified and the type we assigned them to (five datasets have two types).

In Table 5.1, we display an overview of the 41 software-evolution datasets we identified as our search criteria in Section 3.2 demonstrated. There is shown the high-level types of datasets described further below, then the datasets that are sorted to them and at last the count (#) of the datasets that are sorted to them. Note that two datasets have two publications describing them, namely GHTorrent [99, 101] and the Software Heritage Graph Dataset [259, 260]. Furthermore, we assigned five datasets [242, 308, 340, 351, 352] to two types, which is why the counts do add up neither to the 41 datasets nor the 43 publications.

During our card sorting, we derived five high-level types of datasets:

Repository metadata refers to datasets that have collected and share typical version-control data (e.g., commits, pull requests). While such datasets usually involve various other types of data (e.g., software quality through bug reports or human factors through commit authors), we did not assign such datasets to another type except if the authors enriched their dataset for or with such other data. For instance, Yamashita et al. [352] combine software-evolution and software-quality data, which is why we assigned their dataset to repository metadata as well as software quality.

Software quality refers to datasets that involve data about the quality of evolving software systems. Such datasets are enriched or narrowed down to focus on, for instance, bugs [284], code smells [300], or code vulnerabilities [78].

Human factors refers to datasets that focus on the stakeholders of a software system. For example, Fry et al. [84] share a dataset on which they performed identity resolution using Git commit author identifiers.

Testing and deployment refers to datasets that are concerned with the respective development activities. For instance, such datasets include additional data on build logs [35] or continuous integration [25].

Mobile apps refers to datasets that collect software-evolution data related particularly to mobile apps. For example, these datasets represent collections of Android apps [187] or commits that touch Android compatibility checks [226].

Not surprisingly, we can see that software-evolution datasets are mostly concerned with typical repository metadata. This includes datasets that represent the entire software-evolution data corpus, that means datasets that want to describe the totality of the given data sources (i.e. GHTorrent [99, 101], Software Heritage Graph Dataset [259, 260], Sampling Projects [52] and World of code [197]), as well as individual datasets that represent specific branches of the corpus (e.g. duplicated data and specific file types). Moreover, it is not surprising that various other important software-engineering research topics have led to dedicated datasets (e.g., software quality datasets for bug localization). Reflecting on the datasets and their types, we argue that they seem representative for the broader research on software-evolution.

5.1.2 Trend of published datasets

Year	**Count of Datasets**	**Citations**	**Avg. Citations per Paper/ per Year**
2012	1	297	29.7
2013	1	676	75.11
2017	5	271	9.03
2018	9	383	8.51
2019	2	101	12.65
2020	15	253	5.62
2021	10	52	2.6
Total	43	2033	-

Table 5.2: Count of dataset and cites of every year. The average of citations per paper and year have been calculated by dividing the number of citations by the amount of papers in that year and by the number of years between their publishing year and 2022, which is the last possible publishing year of citing papers in our corpus.

In Table 5.2, we look at the practice of how often software-evolution datasets are published in the 5-year period that we considered in our work. The table shows for every year the datasets that are used in this thesis with the sum of the citations of them and the average citation per paper per year. For example 2021 there are 10 datasets that were published and they have a sum of 52 citations, then the average citation is 52 divided by 10 papers and that divided by 2 (year 2021 and 2022) and the result is the 2.6 average citation.

Through the Section 3.2.2 we identified that software-evolution datasets are a stable component of the datasets from the data showcase of the MSR over the years. With 60.29 % of the datasets in there are software-evolution datasets. The software-evolution datasets as a whole are a constant popular theme. They have every year more than 100 citations with the exception of 2021, because the dataset has not yet 1 year time to accumulate citations. As the number of datasets has increased over the years, it has become apparent that there are also some not so popular datasets and therefore the average number of citations has decreased. However, the total number of citations has remained relatively constant. Unlike the newer datasets, GHTorrent [99, 101] demonstrates that it is well established dataset in the community. It has in total 973 citations with the 2 papers and the avg. citations are also higher as the other datasets. The higher average of the citations for the GHTorrent can be distributed to the less concurrence in the beginning of the dataset, since other datasets are recently published that cover the same data corpus.

In summary the software-evolution datasets are strong representatives of the datasets at MSR and consistently possess strong attention from researchers in the area.

5.1.3 Datasets Source

Data Source		**#**
Version-Control Data		**30**
	GitHub	26
	“other” Gits	3
	GitLab	3
	Subversion (SVN)	2
	Apache Git	1
	BitBucket	1
	Gerrit	1
	Mercurial	1
	SourceForge	1
Existing Datasets		**13**
	GHTorrent [99, 101]	11
	World of Code [197]	2
	AndroZoo [4]	1
	Pull-based development [104]	1
Other Sources		**9**
	Jira	2
	Google Big Query	2
	security databases (CVE, NVD)	2
	Travis CI	2
	DockerHub	1
	Google Play	1
	Linux Kernel Mailing List	1
	package repositories (e.g., Debian, PyPi, NPM)	1

Table 5.3: Data sources used to create the 41 datasets.

We investigated from what sources the researchers elicited the involved data, which we display in Table 5.3. There we show the dataset sources as a part of one of three groups (version-control data, existing datasets and other sources) with the count of how many use this source. It should be noted that datasets can use more than one source as example World of Code [197], that use GitHub, Bitbucket, GitLab, and SourceForge and other git as a source. The groups represent how many datasets use this type of source and not how often they use this type. For example World of Code use version-control data, but it count only as one in the count of the group even if it uses several of them.

Since we are studying software-evolution datasets, it is not surprising that all datasets but one involve version-control data. The exception is the dataset of Linux mails by Xu and Zhou [351], which also refers to software-evolution and version-control data within the mails.

In detail, 26 of the 41 datasets have extracted data from GitHub directly, while 11 other datasets have built on the GHTorrent data dump. Consequently, GitHub contributes to more than 90 % of the datasets. A few other datasets used different version-control systems or software-hosting platforms, such as BitBucket, Subversion, and SourceForge. Some datasets (e.g., the Software Heritage Graph, World of Code) also combine data from various version-control systems.

Only nine of our datasets explicitly involve additional data from other sources, such as Jira, security databases, or different package repositories. Such other data sources do not only help to enrich a dataset with diverse projects, but to achieve a certain goal and consequent type from Table 5.1 (e.g., using security databases for data on software quality).

Still, datasets that are systematically enriched with additional data are sparse in our sample.

5.1.4 Sharing Platforms

Host	#
zenodo.org	18
github.com	13
osf.io	2
ghtorrent.org	1
travistorrent.testroots.org	1
figshare.com	1
oss.models-db.com	1
mondego.ics.uci.edu	1
opendata.soccerlab.polymtl.ca	1
bitbucket.org	1
knowledgezoo.xyz	1
Total	41

Table 5.4: Host of the Datasets

We identified on what platform each dataset has been shared (i.e., is hosted), which we display in Table 5.4.

This results in two dominant ones:

- Zenodo with 18 and
- GitHub with 13 datasets.

Other sharing platforms are used by one or two datasets only, specifically those platforms are OSF (2), BitBucket (1), and figshare (1). Thus, 85.36 % percent of the published datasets are stored on sharing platforms.

While it is a valuable trend that more datasets are published in persistent repositories, we also found six instances in which datasets are still shared on apparently personal or university websites. In this instances, the probability to lose the datasets is higher, than the one on sharing platforms, because no dataset in our work has been deleted that was hosted by a sharing platform. In October 2022, three of those six websites were not accessible anymore, namely those of AndroZooOpen [187], where the website no longer exists, UML models [277], where it was hosted on an university website (forwarded to main page) and no longer available is and TravisTorrent [25], where we only get the main domain of the website and are shown similar sites on the domain. All other software-evolution datasets were still accessible via the platform linked in the papers.

In summary, datasets are mostly hosted on Zenodo and Github, and unlike the other hosts also still available.

5.1.5 Data Storing

Format	**Example Storage Formats**	#
spreadsheet	.csv, .xlsx	19
text files	.txt, .xml, .yaml, .json	14
relational database	MySQL, PostgreSQL	11
repository	GitHub repository	4
graph database	Neo4j, Google knowledge graph	3
document-oriented database	MongoDB	2

Table 5.5: Formats used to store data in the datasets.

Here, we investigated the formats researchers have used to store their datasets and their frequency of occurrence. We display an overview of the primary format types and respective examples and their count in Table 5.5. Note that some of the datasets use multiple storage formats, which is why the sum of the rightmost column in Table 5.5 does not add up to the 41 datasets.

As we can see, we identified six primary formats, the two most common ones are spreadsheets (19) and text files (14).

While the two formats are also sometimes a means to import data into a database, some researchers offer their datasets directly as an exported database scheme. Aligning to the previous two formats, the relational scheme is most common (11) of the directly exported database scheme, with graph (3) and document-oriented (2) databases being rarer.

The last format are GitHub repositories with four datasets. One of these datasets form the GitHub repositories format stores its data in a particularly interesting way. It (Bugs.jar [284]) links the researched bugs as branches to the solving commits.

In summary, datasets are for the most part stored in formats that are structured as tables, such as spreadsheets, text files or direct exported database scheme.

5.1.6 Quantity

Interestingly, the datasets span a variety of sizes of data, ranging from six projects in a dataset to far more than 14 million software projects. This include, for instance, plain version-control data (e.g., commits pull requests, issues), blobs, compiled binaries, additional documents, and metrics. As a consequence, the data storing of the datasets is quite diverse, in both the quantity of the datasets and the overall additional data to the version-control data.

5.1.7 Summary RQ.1: Dataset Sharing

To summarize the section there are four points to be considered:

- The 41 software-evolution datasets we identified involve primarily repository metadata (23), sometimes enriched or narrowed down with respect to software quality (8), human factors (6), testing and deployment (5), or mobile apps (4).
- The primary source for the datasets is version-control data, either crawled directly (30) or reused from another dataset (13), that is sometimes (9) enriched with other sources.
- The data is often shared on established platforms like Zenodo (18) or GitHub (13), but six datasets were shared on individual websites, of which three seem to be offline.
- The datasets span a variety of data and sizes, resulting in different storage formats, primarily spreadsheets (19), text files (14), and relational databases (11).

Overall, our results for RQ.1 show a highly diverse landscape of practices for sharing software-evolution datasets.

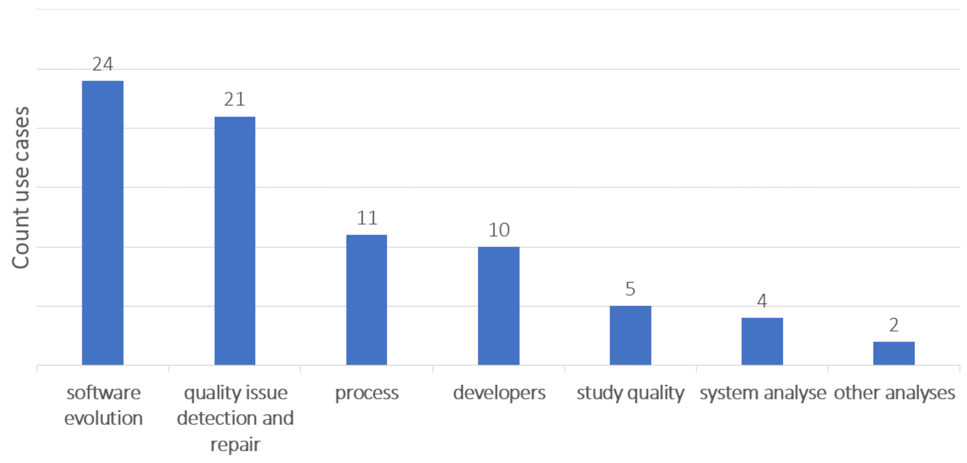

Figure 5.1: Overview of the high-level use cases suggested in dataset papers.

5.2 RQ.2: Analysis Use Cases

For RQ.2, we are interested in the use cases for which researchers have shared software-evolution datasets, and for which they have been used. To this end, we synthesized high-level use cases based on the individual use cases we extracted from each paper, the types of datasets we defined (Table 5.1), and ensuring that at least three papers fit into each category. We display the resulting numbers and use cases in Figure 5.1 and Figure 5.2. Please note that we put each paper of the dataset use cases in possible more than one category and every each publications that use them is sorted in one category.

5.2.1 Well-Established Use Cases

We summarized most use cases by the datasets into one of two categories software-evolution or quality issue detection and repair. The former is concerned with any type of research that aims to improve our understanding of software-evolution and is mentioned on this high level as a motivating use case in many dataset papers. The latter summarizes all research related to studying or resolving quality problems, including bug evolution, code smells, and automatic program repair. The next two topics occur less often, even though they are concerned with the developers (e.g., identification, collaboration, interactions, gender) as well as the processes they employ (e.g., regression testing, reviewing, economical reasons). Unfortunately, such human(-centered) aspects are mentioned and researched fewer times than plain software-evolution or quality; even though they raise serious ethical and legal concerns (e.g., of identifying individuals). One other group is related to the improvement of other studies in the respective research field, which the datasets explicit mentioned. That are for example dataset that want to repeat studies, support the sampling of projects or limit the threat to validity of another study. One last group of papers is related to typical system (e.g., history slicing, feature location) and other analyses (e.g., evolution of UML models).

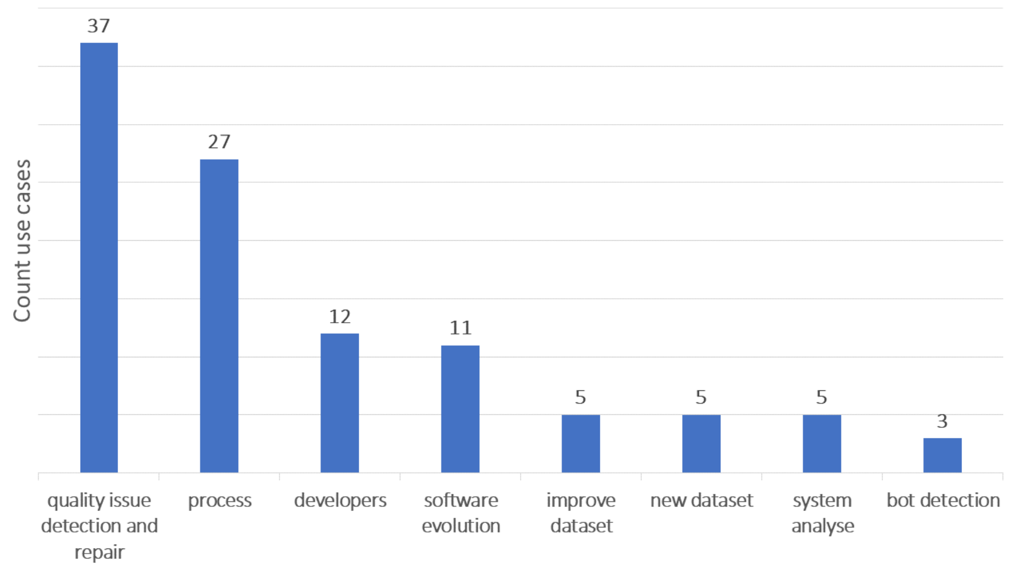

Figure 5.2: Overview of the high-level use cases addressed in publications using the datasets.

By the use cases of the papers that use the datasets the distribution of the categories is slightly changed. The most cases they deal with studying or resolving quality problems or the process that the developer employ. The next two topics are concerned with the developers and software evolution. Here by the publication that use the dataset for software evolution, the picture got a bit broader, for instance, such papers are concerned with redundant development effort in forks and branches or with eliciting change patterns. The following 2 topics are concerned with the improvement of datasets (e.g. improving techniques/guidelines, deduplication and graph compression or the mining of datasets in general) and the creation of new datasets (e.g. the use of the same tool or lessons learned from analyzing the strengths and weaknesses of the datasets). The last group of well-established use cases is related to typical system, the same as by the datasets.

5.2.2 Unaware and Novel Use Cases

From Figure 5.1 and Figure 5.2, we can also see that novel use cases, such as bot identification, are logically hard to anticipate (particularly since the relevant tools may not even exist). This is quite normal, and the further advances in software engineering will continue to lead to new technologies that change how (and whether) software-evolution datasets can be used.

However, we are surprised that only one of the datasets mentioned any use cases related to improving research itself, for instance, using the datasets to design better methods for mining software repositories or combining them to create novel datasets. It exist dataset that want to improve the quality of follow-up studies, but all but one dataset (Sampling Projects [52]) that can also be used in general to create new datasets, this was also found in our study with this publication [53]. This topic was taken up more on the side of the used publications, but also only sporadically, 10 of the 135 publication discuss improvement and new datasets.

Also to note is that 57 of the use cases of the publication matched with the use cases of the datasets of 135, which is a percentage of 42.22 %. Especially to observe is that studying or resolving quality problems matches the use cases and software evolution splits into the other categories of use cases, which are used by the used publications.

Lastly, we excluded 30 use cases from the papers that use a dataset, since these were not concerned with software-evolution and did not require these specific datasets. Instead, they perceived the software-evolution datasets as a collection of subject systems, for instance, for evaluating techniques for scheduling in serverless software systems.

5.2.3 Summary RQ.2: Analysis Use Cases

To summarize the section there are three points to be considered:

- The most common use cases are related to achieving general insights on software-evolution, quality and testing concerns, developers, and development processes.
- The distribution of use cases matches quite well between those assumed in shared datasets and those the datasets are used for.
- The idea of using datasets to improve research itself and derive new datasets is in the queue at the back.

Overall, our results indicate a good alignment between the software-evolution use cases assumed when sharing a dataset and those for which it will be used.

5.3 RQ.3: Challenges of Sharing

In this section we analyze the results of the collected data from Section 4.3. First we show the categorized problems and the connection to the challenges that the researcher predicted in their papers about the datasets.

5.3.1 Challenges/Problems/Limitations of Sharing

Problem	Description	Impacted Datasets	#
quantity and reliability	datasets may not involve entries or a reliable ground-truth to serve as a feasible databasis	Android apps [86], AndroidCompass [226], Bugs.jar [284], C/C++ Code Vulnerabilities [78], Duets [72], Enterprise-Driven [308], Git Archive [204], GHTorrent [99, 101], ManySStuBs4J [150], Repository Deduplication [309], Sampling Projects [52], Semantic Changes [372], TravisTorrent [25], UML models [277], Wonderless [76]	15
missing data	datasets may not involve the right data for an analysis	20-MAD [48], Android apps [86], Bugs.jar [284], Denchmark [155], Dockerfiles [127], Duplicate Pull-Requests [359], Git Archive [204], ManySStuBs4J [150], ManyTypes4Py [217], OCL expressions [233], Semantic Changes [372], TravisTorrent [25], VulinOSS [88]	13
redundant data	datasets may involve example, toy, or stale-fork data	20-MAD [48], CROP [242], Git Archive [204], LogChunks [35], ManySStuBs4J [150], Many-Types4Py [217], OCL expressions [233] Sampling Projects [52], Software Heritage Graph Dataset [259, 260], TravisTorrent [25], Wonderless [76], World of code [197]	12
topicality	datasets age and thereby may become outdated or even unavailable	Android apps [86], AndroidCompass [226], Dockerfiles [127], Enterprise-Driven [308], Git Archive [204], GHTorrent [99, 101], ManySStuBs4J [150], Many-Types4Py [217], TravisTorrent [25], Repository Deduplication [309]	10
faulty or invalid data	datasets may involve manipulated or incorrectly extracted data	Android apps [86], Duplicate Pull-Requests [359], Git Archive [204], GHTorrent [99, 101], ManySStuBs4J [150], OCL expressions [233], Software Heritage Graph Dataset [259, 260], World of code [197]	8
accesibility	datasets may exhibit complicated structures or require high-performance resources	C/C++ Code Vulnerabilities [78], Git Archive [204], ManySStuBs4J [150], Semantic Changes [372], Software Heritage Graph Dataset [259, 260], TravisTorrent [25]	6
others	datasets may face other problems, such as missing tool support for analyses	Andromeda [237], AndroZooOpen [187], Qscored [300]	3
	reused without problems (mentioned)	50K-C [206], Andror2 [340], Git Repositories [219], Identity Resolution [84], Linux Kernel [351], Mixed Graph-Relational Datase [11], Quantifying [66], Shoulders of Giants [365], Software evolution [352], Structured information [293]	10
	no reuse (attempted)	GE526 [323], JTeC [49]	2

Table 5.6: Overview of the most commonly reported problems we identified from the papers (aiming to) use the 41 datasets.

For RQ.3, we are concerned with the problems other researchers experienced with reusing (or attempting to do so) the software-evolution datasets, which represent challenges for researchers creating a software-evolution dataset. Consequently, we are now reporting on the 158 papers that used (133) or analyzed problems with (24) the datasets we identified in our manual search as well as the snowballing. We provide an overview of the six problems we identified most often and the impacted datasets in Table 5.6.

In more detail, these problems are:

Quantity and Reliability: We found 15 instances in which authors raised the problem that a dataset is not large or representative enough to serve as a reliable ground-truth. Most often, these authors argued that a dataset is too small or is not respected in a community. As a consequence, the using researchers had to replicate the original queries, construct their own mining pipeline, or extend a dataset. As a concrete example, the C/C++ Code Vulnerability [78] dataset has been considered too small to employ deep-learning models on it.

Missing Data: In 13 instances, we identified the problem that a dataset does not involve enough of the right data to properly represent the studied phenomenon. For instance, it has been argued that Denchmark [155] lacks version information for the included bug reports, which would be helpful in attribute for different analyses.

Redundant Data: Quite on the opposite, we also found 12 instances in which authors argued that a dataset involves too much redundant or irrelevant data. Particularly, the typical concern of toy and example projects (e.g., server blueprints) or stale forks (which most researchers consider not useful to analyze) have been raised. As a concrete example, some researchers aimed to use the World of code [197] dataset to conduct an expertise-identification analysis. However, they argue that the existence of projects with many forks and other clones biases such an analysis, since the respective developers may occur in many more commits.

Topicality: Since software-evolution is continuous, any shared dataset will become outdated if it is not regularly updated. Specifically, we found 10 instances in which authors raised this concern and argued that they required more recent data (e.g., mining it themselves) or periodic updates of existing datasets (e.g., as was the case with GHTorrent [99, 101]). In the most extreme cases (cf. section Section 5.1), the datasets may even become unavailable, not because they are outdated but simply because a sharing platform shuts down.

Faulty or Invalid Data: Faulty or invalid data threatens all scientific work, making the obtained results meaningless. We identified eight instances in which researchers raised the problem of incorrect metadata (e.g., timestamps). Two examples that are prone to this problem are the GHTorrent dataset in which the authorship date of commits can be overwritten and the Software Heritage Dataset in which duplicated commit identifiers are relabeled to make them unique. Even typical version-control systems and research tools [121] allow developers to manipulate recorded version histories. All such things threaten a dataset and may make it useless for the intended analysis.

Accessibility: We found six instances in which researchers reported on accessibility problems, for instance, because the storage format and structure of a dataset are complicated. Furthermore, software-evolution data is constantly growing, which means that researchers also need more and more computing power as well as hardware to analyze this data. For example, the Software Heritage Graph requires around 850 TiB storage, and even smaller datasets that have been re-shared from it are often too large for typical computers.

Others: Lastly, we identified three individual problems that do not fit into the previous themes. For instance, we found that there is apparently no common tool support for semantic changes, and thus using the corresponding dataset [372] remains a challenge.

Also we analyzed the predicted problems/ limitations of the dataset that the researcher of the datasets presented. There it is necessary to consider that we only found four cases, where the predicted problems/limitations match with the found problem of our analyze. The four datasets are:

- Wonderless [76], where the researcher warn against toy software and that problem also exist by the analyzed papers
- Sampling Projects [52], where it is said it is needed to get extra data to analyze historic evolution of said characteristics
- Git Archive [204], where the dataset is only so actual as the GHTorrent dataset
- GHTorrent , where it is reported that data without timestamps get a selected timestamp and the event stream doesn't report deletion

That are the only matches of the predicted and analyzed problems/challenges. A large part of the predicted limitation/problems are that the dataset can be extended and that is not relevant for the paper that selected the dataset. It can be assumed that the other predicted problems are included in the selection process of the datasets that are used and thus not not considered so important to include them again in their work as they have already been mentioned in the datasets and they always work within the boundaries of the datasets.

On a final note, we would like to remark that all problems mentioned relate to methodological or technical decisions, but we found no statements about ethical or legal concerns related to the datasets.

5.3.2 SummaryRQ.3: Challenges of Sharing

The six recurring challenges for sharing and using a software-evolution dataset are

1. ensuring quantity and reliability;
2. having the right data;
3. handling redundant data;
4. reasoning on topicality;
5. preventing faulty or invalid data; and
6. considering the accessibility of computing resources.

Overall, our results for RQ.3 indicate key technical and methodological challenges for sharing software-evolution datasets.

6. Discussion

In the following, we discuss the implications of our findings and of what we did not find, which are concerns about the ethical and legal aspects of sharing software-evolution datasets. We consider these topics critical and discuss them first, since we argue that such concerns must be solved before any technical solution.

6.1 Privacy, Licensing, and Ethical Concerns

There are many concerns that should be incorporated into the practices of sharing software-evolution datasets, and we were surprised that these were not mentioned.

6.1.1 Privacy

While software-evolution data is mostly concerned with code and changes, human factors are always an important research area (cf. Section 5.2). Usually, developers' information must be anonymized via identifiers in the datasets. However, this is not easy if, for instance, personal information (e.g., names, mail addresses) are scattered across a datasets, for instance, in commit messages, code comments, or discussions. Thus, it is surprising to us, that privacy concerns have not been raised as a challenge in the 200 papers.

Despite the importance of anonymization, it is sometimes necessary to reach out to developers, for example, to fill in a questionnaire to characterize their intentions. This is one of the numerous cases where privacy requirements may harm downstream analyses. Finding the right granularity of anonymization in such cases is known as the privacy-utility challenge.

Furthermore, when creating a dataset, the original personal data is not in charge of the original system anymore. If a developer demanded the erasure of their personal data, that dataset is a serious violation of this demand. Consequently, we must be more careful about how we share software-evolution artifacts and must assess the usefulness of the data against developers' privacy right in different countries.

6.1.2 Licensing

A critical concern when creating and sharing software is the license [16, 276]. Since software-evolution datasets combine a set of software artifacts, the challenge is that all these licenses need to be interoperable so that it is legal to share a dataset. None of the dataset papers mentions this issue, which is quite critical considering also other legal aspects, for instance, of the developers and other stakeholders (e.g.., privacy). We stress that the community needs more guidelines regarding licensing for shared software-evolution datasets and ways to enforce it.

6.1.3 Ethics

The above points are only some of the major concerns with current practices of sharing software-evolution datasets. Recently, Gold and Krinke [93] have analyzed these and other ethical concerns with mining software repositories. For instance, Gold and Krinke stress for consent (by checking the license) and privacy (via pseudonymosation). Their work is a great step in the right direction, but we need to advance further and enforce ethical sharing of software-evolution datasets, for example, by getting approval from an ethics review board.

6.2 Technical Concerns

A core outcome of our study is the dominance of text and spreadsheet formats for storing version-control data (cf. Section 5.1, directly causing several of the problems described by researchers (cf. Section 5.3). While these formats can be easily reused due to their simplicity, they inherently cause problems in later analyses:

- *Missing data types and encoding issues* create problems when a dataset is used on different systems and across different tools. Each tool has its own encoding of files and only supports a limited set of data types and representations (e.g., date formats, identifier sizes). A system-independent format or a clean description of the data types is mostly missing, which creates conversion effort when being reused by other researchers. Such conversion errors and technical problems are well-known sources of errors.
- *Use-case specific data representation* means that the underlying representation of data differs between use cases (e.g., different ways of storing commits and their branches, different format for pull requests). Currently, extracting the right data from the internal representation is not standardized and needs familiarization for each new use case (cf. Section 5.2).
- *Non-standardized analysis pipelines* lead to the problem of reinventing the same analyses. Mostly, each research team that reuses a shared datasets implements their own analysis pipeline, which usually involves the redundantly implemented steps of data cleaning and data transformation. As a result, the initial investment when reusing the data is quite high, which discourages researchers to reuse shared datasets.

Some of the datasets already come with more semantic information or formats, for instance, as a relational, document-oriented, or graph database. These representations cause initial overhead when setting up these systems. However, they promise to improve the understandability and reuse due to their inherent standardization. We argue that exploring different data-storing techniques can help with solving problems when reusing software-evolution dataset.

Most of the problems that we identified in RQ.3 are due to missing standardization in the datasets, but also in the analysis pipeline. The former means that cleaning the data is a major challenge when reusing a shared dataset, which keeps researchers from achieving their actual goal: analyzing the dataset to answer their research questions. Furthermore, problems and inconsistencies within the data are still found, which questions the reliability of conclusions drawn from that data.

The latter (i.e., missing standardization in analysis pipelines), may lead to inconsistencies between the results of two different pipelines, a problem uncovered by reproducibility papers. This problem is even exacerbated when using artificial-intelligence techniques, due to their individual hyperparameters.

To increase the efficiency of research, we argue that it is a major goal to create a standard representations for software-evolution datasets and the corresponding analyses tooling.

6.3 Validity

6.3.1 Internal Validity

Regarding the internal validity, our data analysis is prone to interpretations. We employed open coding and card sorting methods to reduce the risks, but in the end all categories are our interpretation of the data. Similarly, we may have misinterpreted some statements in the papers we read or in additional documentation of the datasets. To allow other researchers to verify and replicate our analysis, we share our data in a persistent open-access repository.

6.3.2 External Validity

Considering the external validity, we recognize that we have performed a manual search and snowballing starting from a single conference only. As a consequence, we may have missed highly relevant papers from other venues that would lead to a different outcome. However, MSR is a prime conference on software evolution, and is the only conference with dedicated tracks (challenges, datasets) that are promoting researchers to publish (high-quality) software-evolution datasets. For this reason, we argue that using MSR's dedicated tracks provides a reliable overview of current (best) practices in terms of sharing such datasets in our community; thus our findings should be transferable to other such datasets, too.

For our snowballing search, we used Google Scholar and considered only the ten most cited papers. So, we may have missed relevant papers. For example topics in use cases that are not so popular, but are also part of the community. We decided for this method to limit the effort of the literature search and performed several pretests, in which we found that the ten snowballed papers provide a good overview of the problems of sharing software-evolution datasets. Consequently, while this threat remains, we argue that we mitigated its impact.

6.4 Closing Remarks

Interestingly, the problems we identified in Chapter 5 and discussed in this section should be well-known in research, but apparently researchers are experiencing them over and over again. Arguably, the individual goals of sharing software-evolution datasets cause challenges that are in conflict with each other as well as with ethics and legal concerns. As a consequence, we argue that we will never be able to create perfect software-evolution datasets. For instance, data protection and privacy are in fundamental conflicts with a complete and persistent software-evolution dataset. So, we as a research community must first define the minimum standard that software-evolution datasets should fulfill, for example, regarding legal requirements, ethical concerns, and best sharing practices. Then, we could start to develop and setup appropriate infrastructures that help mitigate the problems we identified and tackle the discussed challenges. This way, our research community can advance towards more reliable and trustworthy software-evolution datasets as well as analyses, which in turn, contributes to open science, building trust, and accelerating research.

7. Related Work

Even though sharing research artifacts is a long debated issue, software-engineering researchers have only recently started to investigate this topic systematically and in more detail. For instance, Timperley et al. [315] have surveyed 153 software-engineering researchers to understand how artifacts are created, used, and reviewed. The authors aimed to understand current practices and derive recommendations for improving the quality of artifacts. Similarly, Hermann et al. [130] have surveyed 257 researchers who participated in artifact-evaluation (i.e., reviewing) processes to understand their expectations for shared research artifacts and their evaluation. The authors identified specific quality expectations, but also inconsistencies in terminology as well as expectations that should be resolved. Overall, such insights are helpful means to further improve artifact sharing and the conduct of artifact-evaluation tracks, which many major software-engineering conferences and journals have started to adopt. Sharing research artifacts is not only important for transparency and replicability, but also has a small positive impact on a publication's citations, as found by Heumüller et al. [131] who have performed an empirical analysis of papers published at 11 instances of the International Conference on Software Engineering. Other researchers have discussed the pros and cons of artifact sharing in the context of open science [15, 129, 212], proposed or improved guidelines for sharing and reviewing artifacts [54, 165, 166, 213, 292, 343, 375], or further analyzed the incentives of artifact sharing and badges [44, 83, 291], in software engineering and computer science in general. While representing extensive research on artifact sharing, none of these works is concerned with specific practices and challenges of sharing software-evolution data.

In the context of software-evolution data, additional concerns like copyright, licensing, and data privacy [16, 276] are of high importance, since researchers are not sharing their own artifacts, but data contributed by and including information of others (e.g., open-source developers). For instance, many large-scale datasets, such as GHTorrent or SoftwareHeritage [19, 63, 99, 101, 261], or analysis frameworks for such data [316, 317] have been shared in the past. However, when working with software-evolution data, ethical concerns are highly relevant for researchers to consider, for instance, to not spam developers [17] and avoid wasting developers' time or undermining their trust in research.[1] While several researchers have been concerned with the pros and cons of mining software-evolution data [30, 146, 147], they rarely discuss the ethics or good practices of sharing the corresponding datasets. So, it is not surprising that some of the mentioned attempts to release large software-evolution datasets have also been criticized for ethical concerns.

The most detailed research in this direction stems from Gold and Krinke [90, 92], who discuss the ethics of mining software repositories in general. For this purpose,

[1]https://www.theverge.com/2021/4/30/22410164/linux-kernel-university-of-minnesota-banned-open-source

Gold and Krinke investigated the MSR mining challenges from 2006 until 2021 and conducted a community survey; using individual cases (including the creation of a software-evolution dataset) to showcase ethical concerns that exist in such cases. This research is closely related to ours (e.g., similar methodology and focus on research artifacts), but we are concerned specifically with software-evolution datasets and more broadly with all concerns related to their sharing (not only ethical ones). As a consequence, we provide a more detailed overview of the practices and challenges of sharing software-evolution datasets, complementing such previous works. Finally, Kotti and Spinellis [163] as well as Kotti et al. [161] are concerned with the datasets published at MSR, but investigate their scientific impact (e.g., use, citations). Some parts of their work are similar to ours, for instance, investigating what papers use the published datasets. However, we have a different focus, since we are interested only in software-evolution datasets and perform a more qualitative analysis of the practices and challenges of sharing such datasets.

In summary, we extend the current state-of-the-art on sharing software-evolution datasets by focusing on this specific case. Particularly, we elicit what researchers are doing instead of surveying their opinions or what they claim to do by qualitatively analyzing papers instead of surveying researchers. We also provide an overview of the shared datasets, their intended use cases, as well as the problems of sharing and reusing these datasets that researchers have experienced and reported. By discussing the use cases and corresponding challenges (those mentioned and not mentioned), we aim to provide a more in-depth understanding of best practices for sharing software-evolution datasets.

8. Conclusion

In this study, we produced a qualitative literature analysis of 200 papers based on which we improved our understanding of how software-evolution datasets are shared and used.

We found

RQ.1: that the practices for sharing software-evolution datasets are inconsistent regarding, for instance, data formats, sharing platforms, and the involved data (cf. Section 5.1);

RQ.2: that the use cases assumed by researchers who share and use datasets align well, but dedicated datasets for improving research itself seem irrelevant (cf. Section 5.2); and

RQ.3: six recurring challenges for using shared software-evolution datasets relating to reliability, feasibility for the research, data redundancy, topicality, faulty data, and available computing resources (cf. Section 5.3).

Reflecting on these insights as well as what has not been reported in the 200 papers, we discussed future challenges for improving the sharing of software-evolution datasets. In particular, we discussed technical steps for improving the management of the datasets, but also legal and ethical issues, which were not mentioned as major concerns in the papers. We consider this highly problematic and argue that we need to first define the general rules/guidelines (e.g., on ethics) before larger and larger datasets or infrastructures are designed. It may be possible to construct automatic tools to enforce this rules and guidelines. In this regard, we hope that our contributions are a helpful means for researchers to analyze and discuss how to improve current practices. As a step into that direction, we aim to analyze the case of sharing software-evolution datasets with experts on the topics of data privacy and protection. Furthermore, it is to be seen whether these problems also occur in other data sets from other conferences than MSR with different focuses.

A. Appendix

Dataset	Cited
GE526 [323]	
Qscored [300]	[115, 193]
Duets [72]	[304, 305, 329]
Wonderless [76]	[55, 339, 363]
Andror2 [340]	[79, 135, 288]
Quantifying [66]	[65, 82, 210, 300, 360]
JTeC [49]	[82, 331, 336]
AndroidCompass [226]	[188, 227–229]
Denchmark [155]	[39, 135, 156, 157]
Andromeda [237]	[235, 236, 238]
Mixed Graph-Relational [11]	[12, 38, 69, 247]
Linux Kernel [351]	[81, 82, 164, 280, 289, 313, 332]
Enterprise-Driven [308]	[8, 26, 82, 87, 123, 330, 342]
Shoulders of Giants [365]	[132, 154, 324, 367]
LogChunks [35]	[34, 73, 160, 231, 283, 312]
20-MAD [48]	[69, 81, 82, 170, 211, 336]
ManyTypes4Py [217]	[107, 142, 180, 194, 218, 252, 366]
Software evolution [352]	[94, 162, 164, 189, 190, 302, 357]
Software Heritage Graph [260]	[94, 257, 258, 290, 337, 338, 341]
Dockerfiles [127]	[13, 116–118, 128, 182, 264, 318, 364]
AndroZooOpen [187]	[124, 171, 172, 267, 268, 319, 322, 354, 358, 368]
Git Repositories [219]	[6, 38, 57, 75, 139, 153, 257, 272, 278, 309]
Repository Deduplication [309]	[72, 196, 198, 219, 250, 278, 299, 308, 310, 356]
Semantic Changes [372]	[162, 164, 176, 177, 256, 296, 311, 346, 348, 373]
50K-C [206]	[37, 68, 72, 126, 152, 162, 164, 286, 297, 301]
Sampling Projects [52]	[10, 45, 46, 53, 56, 96, 106, 234, 262, 333]
OCL expressions [233]	[136, 162, 164, 214–216, 253, 256, 275, 294]
Duplicate Pull-Requests [359]	[64, 155, 162, 164, 178, 179, 222, 274, 335, 371]
Identity Resolution [84]	[57, 59, 61, 62, 94, 139, 198, 334, 350, 356]
CROP [242]	[91, 109, 119, 164, 243–245, 249, 320, 321]
Structured information [293]	[41, 108, 137, 164, 168, 192, 307, 347, 355, 370]
Git Archive [204]	[27, 52, 74, 94, 95, 162, 164, 205, 281, 282]
C_C Code Vulnerability [78]	[28, 42, 43, 173, 175, 230, 232, 273, 349, 353]
Software Heritage Graph [259]	[33, 69, 82, 94, 97, 198, 281, 282, 334, 361]
UML models [277]	[70, 98, 120, 125, 164, 191, 240, 256, 271, 275]
VulinOSS [88]	[20, 28, 77, 80, 112, 144, 164, 183, 263, 282]
Android apps [86]	[51, 89, 113, 114, 202, 209, 224, 248, 251, 287]
World of code [197]	[7, 57, 58, 60–62, 69, 84, 122, 219]
ManySStuBs4J [150]	[1, 28, 40, 67, 151, 208, 266, 279, 314, 345]
Bugs.jar [284]	[71, 143, 150, 174, 184–186, 195, 200, 285]
TravisTorrent [25]	[22–24, 47, 181, 246, 269, 328, 362, 369]
GHTorrent [101]	[2, 3, 50, 99, 100, 102, 134, 145, 148, 325]
The GHTorent [99]	[18, 21, 100, 105, 111, 133, 145, 221, 326, 327]

Table A.1: Overview of dataset to all snowballed documents of them

Bibliography

[1] Miltiadis Allamanis, Henry Jackson-Flux, and Marc Brockschmidt. 2021. Self-supervised bug detection and repair. *Advances in Neural Information Processing Systems* 34 (2021), 27865–27876. (cited on Page 59)

[2] Miltiadis Allamanis, Hao Peng, and Charles Sutton. 2016. A convolutional attention network for extreme summarization of source code. In *International conference on machine learning*. PMLR, 2091–2100. (cited on Page 29 and 59)

[3] Miltiadis Allamanis and Charles Sutton. 2013. Mining source code repositories at massive scale using language modeling. In *MSR*. IEEE, 207–216. (cited on Page 59)

[4] Kevin Allix, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. 2016. Androzoo: Collecting millions of android apps for the research community. In *MSR*. (cited on Page 40)

[5] Kevin Allix, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2016. AndroZoo. In *MSR*. ACM. (cited on Page 17)

[6] Idan Amit and Dror G Feitelson. 2021. Corrective commit probability: a measure of the effort invested in bug fixing. *Software Quality Journal* 29, 4 (2021). (cited on Page 59)

[7] Sadika Amreen, Audris Mockus, Russell Zaretzki, Christopher Bogart, and Yuxia Zhang. 2020. ALFAA: Active Learning Fingerprint based Anti-Aliasing for correcting developer identity errors in version control systems. *Empirical Software Engineering* 25, 2 (2020). (cited on Page 59)

[8] Florian Angermeir, Markus Voggenreiter, Fabiola Moyon, and Daniel Mendez. 2021. Enterprise-driven open source software: a case study on security automation. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 278–287. (cited on Page 25 and 59)

[9] Gábor Antal, Márton Keleti, and Péter Hegedűs. 2020. Exploring the Security Awareness of the Python and JavaScript Open Source Communities. In *MSR*. ACM. (cited on Page 26)

[10] Pasquale Ardimento, Lerina Aversano, Mario Luca Bernardi, Marta Cimitile, and Martina Iammarino. 2022. Just-in-time software defect prediction using deep temporal convolutional networks. *Neural Computing and Applications* 34, 5 (2022). (cited on Page 59)

[11] Usman Ashraf, Christoph Mayr-Dorn, Alexander Egyed, and Sebastiano Panichella. 2020. A Mixed Graph-Relational Dataset of Socio-technical Interactions in Open Source Systems. In *MSR*, Sunghun Kim, Georgios Gousios, Sarah Nadi, and Joseph Hejderup (Eds.). ACM. (cited on Page 17, 22, 25, 30, 37, 47, and 59)

[12] Usman Ashraf, Christoph Mayr-Dorn, Atif Mashkoor, Alexander Egyed, and Sebastiano Panichella. 2021. Do communities in developer interaction networks align with subsystem developer teams? An empirical study of open source systems. In *2021 IEEE/ACM Joint 15th International Conference on Software and System Processes (ICSSP) and 16th ACM/IEEE International Conference on Global Software Engineering (ICGSE)*. IEEE, 61–71. (cited on Page 25 and 59)

[13] Hideaki Azuma, Shinsuke Matsumoto, Yasutaka Kamei, and Shinji Kusumoto. 2022. An empirical study on self-admitted technical debt in Dockerfiles. *Empirical Software Engineering* 27, 2 (2022). (cited on Page 59)

[14] Monya Baker. 2016. 1,500 scientists lift the lid on reproducibility. *Nature* 533, 7604 (2016). (cited on Page 1)

[15] Maria Teresa Baldassarre, Neil Ernst, Ben Hermann, Tim Menzies, and Rahul Yedida. 2023. (Re)Use of Research Results (Is Rampant). *Commun. ACM* 66, 2 (jan 2023). (cited on Page 1 and 55)

[16] Miriam Ballhausen. 2019. Free and Open Source Software Licenses Explained. *Computer* 52, 6 (2019). (cited on Page 51 and 55)

[17] Sebastian Baltes and Stephan Diehl. 2016. Worse Than Spam: Issues In Sampling Software Developers. In *ESEM*. ACM. (cited on Page 1 and 55)

[18] Sebastian Baltes and Paul Ralph. 2022. Sampling in software engineering research: A critical review and guidelines. *Empirical Software Engineering* 27, 4 (2022). (cited on Page 59)

[19] Sebastian Baltes, Christoph Treude, and Stephan Diehl. 2019. SOTorrent: Studying the Origin, Evolution, and Usage of Stack Overflow Code Snippets. In *MSR*. IEEE. (cited on Page 13 and 55)

[20] Canan Batur Şahin and Laith Abualigah. 2021. A novel deep learning-based feature selection model for improving the static analysis of vulnerability detection. *Neural Computing and Applications* 33, 20 (2021). (cited on Page 59)

[21] Jason Baumgartner, Savvas Zannettou, Brian Keegan, Megan Squire, and Jeremy Blackburn. 2020. The pushshift reddit dataset. In *Proceedings of the international AAAI conference on web and social media*, Vol. 14. (cited on Page 59)

[22] Jonathan Bell, Owolabi Legunsen, Michael Hilton, Lamyaa Eloussi, Tifany Yung, and Darko Marinov. 2018. DeFlaker: Automatically detecting flaky tests. In *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*. IEEE, 433–444. (cited on Page 59)

[23] Moritz Beller, Georgios Gousios, Annibale Panichella, Sebastian Proksch, Sven Amann, and Andy Zaidman. 2017. Developer testing in the ide: Patterns, beliefs, and behavior. *IEEE Transactions on Software Engineering* 45, 3 (2017). (cited on Page)

[24] Moritz Beller, Georgios Gousios, and Andy Zaidman. 2017. Oops, my tests broke the build: An explorative analysis of travis ci with github. In *MSR*. IEEE, 356–367. (cited on Page 29 and 59)

[25] Moritz Beller, Georgios Gousios, and Andy Zaidman. 2017. TravisTorrent: synthesizing Travis CI and GitHub for full-stack research on continuous integration. In *MSR*, Jesús M. González-Barahona, Abram Hindle, and Lin Tan (Eds.). IEEE Computer Society. (cited on Page 20, 24, 28, 32, 37, 38, 42, 47, and 59)

[26] Lodewijk Bergmans, Xander Schrijen, Edwin Ouwehand, and Magiel Bruntink. 2021. Measuring source code conciseness across programming languages using compression. In *2021 IEEE 21st International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 47–57. (cited on Page 59)

[27] David Berrocal-Macías, Zakieh Alizadeh-Sani, Francisco Pinto-Santos, Alfonso González-Briones, Pablo Chamoso, and Juan M Corchado. 2021. Services Extraction for Integration in Software Projects via an Agent-Based Negotiation System. In *Practical Applications of Agents and Multi-Agent Systems*. Springer, 241–252. (cited on Page 59)

[28] Guru Bhandari, Amara Naseer, and Leon Moonen. 2021. CVEfixes: automated collection of vulnerabilities and their fixes from open-source software. In *Proceedings of the 17th International Conference on Predictive Models and Data Analytics in Software Engineering.* (cited on Page 59)

[29] Avijit Bhattacharjee, Sristy Sumana Nath, Shurui Zhou, Debasish Chakroborti, Banani Roy, Chanchal K. Roy, and Kevin Schneider. 2020. An Exploratory Study to Find Motives Behind Cross-platform Forks from Software Heritage Dataset. In *MSR*. ACM. (cited on Page 26)

[30] Christian Bird, Peter C. Rigby, Earl T. Barr, David J. Hamilton, Daniel M. German, and Prem Devanbu. 2009. The Promises and Perils of Mining Git. In *MSR*. IEEE. (cited on Page 55)

[31] Ekaba Bisong, Eric Tran, and Olga Baysal. 2017. Built to Last or Built Too Fast? Evaluating Prediction Models for Build Times. In *MSR*. IEEE. (cited on Page 28)

[32] John D. Blischak, Emily R. Davenport, and Greg Wilson. 2016. A Quick Introduction to Version Control with Git and GitHub. *PLOS Computational Biology* 12, 1 (19 jan 2016), e1004668. https://doi.org/10.1371/journal.pcbi.1004668 (cited on Page 6)

[33] Paolo Boldi, Antoine Pietri, Sebastiano Vigna, and Stefano Zacchiroli. 2020. Ultra-large-scale repository analysis via graph compression. In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 184–194. (cited on Page 27 and 59)

[34] Casper Boone, Carolin Brandt, and Andy Zaidman. 2022. Fixing Continuous Integration Tests From Within the IDE With Contextual Information. (2022). (cited on Page 59)

[35] Carolin E. Brandt, Annibale Panichella, Andy Zaidman, and Moritz Beller. 2020. LogChunks: A Data Set for Build Log Analysis. In *MSR*, Sunghun Kim, Georgios Gousios, Sarah Nadi, and Joseph Hejderup (Eds.). ACM. (cited on Page 17, 22, 25, 31, 37, 38, 47, and 59)

[36] O. Pearl Brereton, Barbara A. Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil. 2007. Lessons from Applying the Systematic Literature Review Process within the Software Engineering Domain. *Journal of Systems and Software* 80, 4 (2007). (cited on Page 11)

[37] Rebecca Brunner, Robert Dyer, Maria Paquin, and Elena Sherman. 2020. PAClab: a program analysis collaboratory. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* (cited on Page 59)

[38] Javier Luis Cánovas Izquierdo and Jordi Cabot. 2022. On the analysis of non-coding roles in open source development. *Empirical Software Engineering* 27, 1 (2022). (cited on Page 59)

[39] Junming Cao, Bihuan Chen, Chao Sun, Longjie Hu, and Xin Peng. 2021. Characterizing Performance Bugs in Deep Learning Systems. *arXiv preprint arXiv:2112.01771* (2021). (cited on Page 59)

[40] Saikat Chakraborty, Yangruibo Ding, Miltiadis Allamanis, and Baishakhi Ray. 2020. Codit: Code editing with tree-based neural models. *IEEE Transactions on Software Engineering* (2020). (cited on Page 59)

[41] Wei Chen, Jia-Hong Zhou, Jia-Xin Zhu, Guo-Quan Wu, and Jun Wei. 2019. Semi-supervised learning based tag recommendation for docker repositories. *Journal of Computer Science and Technology* 34, 5 (2019). (cited on Page 59)

[42] Zimin Chen, Steve Kommrusch, and Martin Monperrus. 2021. Neural transfer learning for repairing security vulnerabilities in c code. *arXiv preprint arXiv:2104.08308* (2021). (cited on Page 27 and 59)

[43] Xiao Cheng, Guanqin Zhang, Haoyu Wang, and Yulei Sui. 2022. Path-sensitive code embedding via contrastive learning for software vulnerability detection. In *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis.* (cited on Page 27 and 59)

[44] Bruce R. Childers and Panos K. Chrysanthis. 2017. Artifact Evaluation: Is It a Real Incentive?. In *2017 IEEE 13th International Conference on e-Science (e-Science).* (cited on Page 1 and 55)

[45] Senior Scientist Dipl-Ing Dr Christian. 2022. *Fine-grained Change Analysis for TypeScript based Systems.* Ph. D. Dissertation. Institut für Informatik. (cited on Page 27 and 59)

[46] Matteo Ciniselli, Nathan Cooper, Luca Pascarella, Antonio Mastropaolo, Emad Aghajani, Denys Poshyvanyk, Massimiliano Di Penta, and Gabriele Bavota. 2021. An Empirical Study on the Usage of Transformer Models for Code Completion. *IEEE Transactions on Software Engineering* (2021). (cited on Page 27 and 59)

[47] Jürgen Cito, Gerald Schermann, John Erik Wittern, Philipp Leitner, Sali Zumberi, and Harald C Gall. 2017. An empirical analysis of the docker container ecosystem on github. In *MSR*. IEEE, 323–333. (cited on Page 59)

[48] Maëlick Claes and Mika V. Mäntylä. 2020. 20-MAD: 20 Years of Issues and Commits of Mozilla and Apache Development. In *MSR*, Sunghun Kim, Georgios Gousios, Sarah Nadi, and Joseph Hejderup (Eds.). ACM. (cited on Page 17, 23, 26, 31, 37, 47, and 59)

[49] Federico Corò, Roberto Verdecchia, Emilio Cruciani, Breno Miranda, and Antonia Bertolino. 2020. JTeC: A Large Collection of Java Test Classes for Test Code Analysis and Processing. In *MSR*, Sunghun Kim, Georgios Gousios, Sarah Nadi, and Joseph Hejderup (Eds.). ACM. (cited on Page 16, 22, 25, 30, 37, 47, and 59)

[50] Valerio Cosentino, Javier L Cánovas Izquierdo, and Jordi Cabot. 2017. A systematic mapping study of software development with GitHub. *IEEE Access* 5 (2017). (cited on Page 59)

[51] Luis Cruz, Rui Abreu, and David Lo. 2019. To the attention of mobile software developers: guess what, test your app! *Empirical Software Engineering* 24, 4 (2019). (cited on Page 59)

[52] Ozren Dabic, Emad Aghajani, and Gabriele Bavota. 2021. Sampling Projects in GitHub for MSR Studies. In *MSR*. IEEE, IEEE, 560–564. https://doi.org/10.1109/MSR52588.2021.00074 (cited on Page 18, 23, 27, 31, 37, 38, 45, 47, 49, and 59)

[53] Samip Dahal, Adyasha Maharana, and Mohit Bansal. 2022. Scotch: A Semantic Code Search Engine for IDEs. In *Deep Learning for Code Workshop.* (cited on Page 27, 45, and 59)

[54] Carlos Diego Nascimento Damasceno and Daniel Strüber. 2021. Quality Guidelines for Research Artifacts in Model-Driven Engineering. In *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS).* 285–296. (cited on Page 55)

[55] Giuseppe De Palma, Saverio Giallorenzo, Jacopo Mauro, Matteo Trentin, and Gianluigi Zavattaro. 2022. Topology-aware Serverless Function-Execution Scheduling. *arXiv preprint arXiv:2205.10176* (2022). (cited on Page 25 and 59)

[56] Alexandre Decan, Tom Mens, Pooya Rostami Mazrae, and Mehdi Golzadeh. 2022. On the Use of GitHub Actions in Software Development Repositories. (cited on Page 27 and 59)

[57] Tapajit Dey, Andrey Karnauch, and Audris Mockus. 2021. Representation of developer expertise in open source software. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 995–1007. (cited on Page 28 and 59)

[58] Tapajit Dey, Yuxing Ma, and Audris Mockus. 2019. Patterns of effort contribution and demand and user classification based on participation patterns in npm ecosystem. In *Proceedings of the fifteenth international conference on predictive models and data analytics in software engineering*. (cited on Page 59)

[59] Tapajit Dey and Audris Mockus. 2020. Effect of technical and social factors on pull request quality for the npm ecosystem. In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. (cited on Page 59)

[60] Tapajit Dey and Audris Mockus. 2020. Which pull requests get accepted and why? a study of popular npm packages. *arXiv preprint arXiv:2003.01153* (2020). (cited on Page 28 and 59)

[61] Tapajit Dey, Sara Mousavi, Eduardo Ponce, Tanner Fry, Bogdan Vasilescu, Anna Filippova, and Audris Mockus. 2020. Detecting and characterizing bots that commit code. In *MSR*. (cited on Page 28 and 59)

[62] Tapajit Dey, Bogdan Vasilescu, and Audris Mockus. 2020. An exploratory study of bot commits. In *Proceedings of the IEEE/ACM 42nd international conference on software engineering workshops*. (cited on Page 28 and 59)

[63] Roberto Di Cosmo. 2018. Software Heritage: Collecting, Preserving, and Sharing All Our Source Code. In *ASE*. ACM. (cited on Page 55)

[64] Andrea Di Sorbo, Gerardo Canfora, and Sebastiano Panichella. 2019. " Won't We Fix this Issue?" Qualitative Characterization and Automated Identification of Wontfix Issues on GitHub. *arXiv preprint arXiv:1904.02414* (2019). (cited on Page 59)

[65] Themistoklis Diamantopoulos, Christiana Galegalidou, and Andreas L Symeonidis. 2021. Software Task Importance Prediction based on Project Management Data.. In *ICSOFT*. (cited on Page 59)

[66] Themistoklis Diamantopoulos, Michail D. Papamichail, Thomas Karanikiotis, Kyriakos C. Chatzidimitriou, and Andreas L. Symeonidis. 2020. Employing Contribution and Quality Metrics for Quantifying the Software Development Process. In *MSR*, Sunghun Kim, Georgios Gousios, Sarah Nadi, and Joseph Hejderup (Eds.). ACM. (cited on Page 16, 22, 25, 30, 37, 47, and 59)

[67] Yangruibo Ding, Baishakhi Ray, Premkumar Devanbu, and Vincent J Hellendoorn. 2020. Patching as translation: the data and the metaphor. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 275–286. (cited on Page 59)

[68] Yiwen Dong, Tianxiao Gu, Yongqiang Tian, and Chengnian Sun. 2022. SnR: constraint-based type inference for incomplete Java code snippets. In *Proceedings of the 44th International Conference on Software Engineering*. (cited on Page 59)

[69] Santiago Dueñas, Valerio Cosentino, Jesus M Gonzalez-Barahona, Alvaro del Castillo San Felix, Daniel Izquierdo-Cortazar, Luis Cañas-Díaz, and Alberto Pérez García-Plaza. 2021. GrimoireLab: A toolset for software development analytics. *PeerJ Computer Science* 7 (2021). (cited on Page 59)

[70] Santiago Dueñas, Valerio Cosentino, Gregorio Robles, and Jesus M Gonzalez-Barahona. 2018. Perceval: software project data at your will. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. (cited on Page 59)

[71] Thomas Durieux, Fernanda Madeiral, Matias Martinez, and Rui Abreu. 2019. Empirical review of Java program repair tools: A large-scale experiment on 2,141 bugs and 23,551 repair attempts. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* (cited on Page 28 and 59)

[72] Thomas Durieux, César Soto-Valero, and Benoit Baudry. 2021. Duets: A Dataset of Reproducible Pairs of Java Library-Clients. In *MSR*. IEEE, IEEE, 545–549. https://doi.org/10.1109/MSR52588.2021.00071 (cited on Page 16, 22, 25, 30, 37, 47, and 59)

[73] Omar M Elazhary. 2021. *Exploring the socio-technical impact of continuous integration: tools, practices, and humans*. Ph. D. Dissertation. (cited on Page 59)

[74] Ahmed Elnaggar, Wei Ding, Llion Jones, Tom Gibbs, Tamas Feher, Christoph Angerer, Silvia Severini, Florian Matthes, and Burkhard Rost. 2021. CodeTrans: Towards Cracking the Language of Silicon's Code Through Self-Supervised Deep Learning and High Performance Computing. *arXiv preprint arXiv:2104.02443* (2021). (cited on Page 27 and 59)

[75] Kalvin Eng and Abram Hindle. 2021. Revisiting Dockerfiles in Open Source Software Over Time. In *MSR*. IEEE, 449–459. (cited on Page 59)

[76] Nafise Eskandani and Guido Salvaneschi. 2021. The Wonderless Dataset for Serverless Computing. In *MSR*. IEEE. (cited on Page 16, 22, 25, 30, 37, 47, 49, and 59)

[77] Jiahao Fan, Yi Li, Shaohua Wang, and Tien N Nguyen. 2020. AC/C++ code vulnerability dataset with code changes and CVE summaries. In *MSR*. (cited on Page 59)

[78] Jiahao Fan, Yi Li, Shaohua Wang, and Tien N. Nguyen. 2020. A C/C++ Code Vulnerability Dataset with Code Changes and CVE Summaries. In *MSR*, Sunghun Kim, Georgios Gousios, Sarah Nadi, and Joseph Hejderup (Eds.). ACM. (cited on Page 19, 24, 27, 32, 37, 38, 47, 48, and 59)

[79] Mattia Fazzini, Chase Choi, Juan Manuel Copia, Gabriel Lee, Yoshiki Kakehi, Alessandra Gorla, and Alessandro Orso. 2022. Use of Test Doubles in Android Testing: An In-Depth Investigation. (2022). (cited on Page 59)

[80] Rudolf Ferenc, Péter Hegedűs, Péter Gyimesi, Gábor Antal, Dénes Bán, and Tibor Gyimóthy. 2019. Challenging machine learning algorithms in predicting vulnerable javascript functions. In *2019 IEEE/ACM 7th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)*. IEEE, 8–14. (cited on Page 59)

[81] Samuel W Flint, Jigyasa Chauhan, and Robert Dyer. 2021. Escaping the time pit: Pitfalls and guidelines for using time-based git data. In *MSR*. IEEE, 85–96. (cited on Page 59)

[82] Samuel W Flint, Jigyasa Chauhan, and Robert Dyer. 2022. Pitfalls and guidelines for using time-based Git data. *Empirical Software Engineering* 27, 7 (2022). (cited on Page 27 and 59)

[83] Eitan Frachtenberg. 2022. Research artifacts and citations in computer systems papers. *PeerJ Computer Science* 8 (2022). (cited on Page 1 and 55)

[84] Tanner Fry, Tapajit Dey, Andrey Karnauch, and Audris Mockus. 2020. A Dataset and an Approach for Identity Resolution of 38 Million Author IDs extracted from 2B Git Commits. In *MSR*, Sunghun Kim, Georgios Gousios, Sarah Nadi, and Joseph Hejderup (Eds.). ACM. (cited on Page 18, 23, 25, 28, 32, 37, 38, 47, and 59)

[85] Aakash Gautam, Saket Vishwasrao, and Francisco Servant. 2017. An Empirical Study of Activity, Popularity, Size, Testing, and Stability in Continuous Integration. In *MSR*. IEEE. (cited on Page 29)

[86] Franz-Xaver Geiger, Ivano Malavolta, Luca Pascarella, Fabio Palomba, Dario Di Nucci, and Alberto Bacchelli. 2018. A graph-based dataset of commit history of real-world Android apps. In *MSR*, Andy Zaidman, Yasutaka Kamei, and Emily Hill (Eds.). ACM. (cited on Page 19, 24, 28, 32, 37, 47, and 59)

[87] Antonios Gkortzis, Daniel Feitosa, and Diomidis Spinellis. 2021. Software reuse cuts both ways: An empirical analysis of its relationship with security vulnerabilities. *Journal of Systems and Software* 172 (2021). (cited on Page 25 and 59)

[88] Antonios Gkortzis, Dimitris Mitropoulos, and Diomidis Spinellis. 2018. VulinOSS: a dataset of security vulnerabilities in open-source systems. In *MSR*, Andy Zaidman, Yasutaka Kamei, and Emily Hill (Eds.). ACM. (cited on Page 19, 24, 25, 32, 37, 47, and 59)

[89] Bruno Góis Mateus and Matias Martinez. 2019. An empirical study on quality of Android applications written in Kotlin language. *Empirical Software Engineering* 24, 6 (2019). (cited on Page 28 and 59)

[90] Nicolas E. Gold and Jens Krinke. 2020. Ethical Mining: A Case Study on MSR Mining Challenges. In *MSR* (Seoul, Republic of Korea) *(MSR '20)*. Association for Computing Machinery, New York, NY, USA, 265–276. (cited on Page 1, 12, and 55)

[91] Nicolas E Gold and Jens Krinke. 2020. Ethical mining: A case study on MSR mining challenges. In *MSR*. (cited on Page 59)

[92] Nicolas E Gold and Jens Krinke. 2022. Ethics in the mining of software repositories. *Empirical Software Engineering* 27, 1 (2022). (cited on Page 1, 12, 55, and 56)

[93] Nicolas E. Gold and Jens Krinke. 2022. Ethics in the Mining of Software Repositories. *Empirical Software Engineering* 27, 1 (2022). (cited on Page 52)

[94] Nicolas E Gold and Jens Krinke. 2022. Ethics in the mining of software repositories. *Empirical Software Engineering* 27, 1 (2022). (cited on Page 59)

[95] Yaroslav Golubev, Maria Eliseeva, Nikita Povarov, and Timofey Bryksin. 2020. A study of potential code borrowing and license violations in java projects on github. In *MSR*. (cited on Page 27 and 59)

[96] Mehdi Golzadeh, Alexandre Decan, and Natarajan Chidambaram. 2022. On the accuracy of bot detection techniques. In *International Workshop on Bots in Software Engineering (BotSE). IEEE.* (cited on Page 27 and 59)

[97] Teresa Gomez-Diaz and Tomas Recio. 2021. Open comments on the Task Force SIRS report: Scholarly Infrastructures for Research Software (EOSC Executive Board, EOSCArchitecture). *arXiv preprint arXiv:2108.06127* (2021). (cited on Page 59)

[98] Bethany Gosala, Sripriya Roy Chowdhuri, Jyoti Singh, Manjari Gupta, and Alok Mishra. 2021. Automatic classification of UML class diagrams using deep learning technique: convolutional neural network. *Applied Sciences* 11, 9 (2021). (cited on Page 59)

[99] Georgios Gousios. 2013. The GHTorent dataset and tool suite. In *MSR*, Thomas Zimmermann, Massimiliano Di Penta, and Sunghun Kim (Eds.). IEEE, IEEE Computer Society, 233–236. https://doi.org/10.1109/MSR.2013.6624034 (cited on Page 13, 17, 18, 19, 20, 21, 24, 29, 33, 37, 38, 39, 40, 47, 48, 55, and 59)

[100] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An exploratory study of the pull-based software development model. In *Proceedings of the 36th international conference on software engineering.* (cited on Page 29 and 59)

[101] Georgios Gousios and Diomidis Spinellis. 2012. GHTorrent: Github's data from a firehose. In *MSR*, Michele Lanza, Massimiliano Di Penta, and Tao Xie (Eds.). IEEE Computer Society. (cited on Page 13, 17, 18, 19, 20, 21, 24, 29, 32, 37, 38, 39, 40, 47, 48, 55, and 59)

[102] Georgios Gousios, Bogdan Vasilescu, Alexander Serebrenik, and Andy Zaidman. 2014. Lean GHTorrent: GitHub data on demand. In *MSR*. (cited on Page 59)

[103] Georgios Gousios and Andy Zaidman. 2014. A dataset for pull-based development research. In *MSR*. ACM. (cited on Page 17)

[104] Georgios Gousios and Andy Zaidman. 2014. A dataset for pull-based development research. In *MSR*. (cited on Page 40)

[105] Georgios Gousios, Andy Zaidman, Margaret-Anne Storey, and Arie Van Deursen. 2015. Work practices and challenges in pull-based development: The integrator's perspective. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, Vol. 1. IEEE, 358–368. (cited on Page 29 and 59)

[106] Konstantin Grotov, Sergey Titov, Vladimir Sotnikov, Yaroslav Golubev, and Timofey Bryksin. 2022. A Large-Scale Comparison of Python Code in Jupyter Notebooks and Scripts. *arXiv preprint arXiv:2203.16718* (2022). (cited on Page 27 and 59)

[107] Bernd Gruner, Tim Sonnekalb, Thomas S Heinze, and Clemens-Alexander Brust. 2022. Cross-Domain Evaluation of a Deep Learning-Based Type Inference System. *arXiv preprint arXiv:2208.09189* (2022). (cited on Page 26 and 59)

[108] Michele Guerriero, Martin Garriga, Damian A Tamburri, and Fabio Palomba. 2019. Adoption, support, and challenges of infrastructure-as-code: Insights from industry. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 580–589. (cited on Page 59)

[109] Chenkai Guo, Dengrong Huang, Naipeng Dong, Quanqi Ye, Jing Xu, Yaqing Fan, Hui Yang, and Yifan Xu. 2019. Deep review sharing. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 61–72. (cited on Page 59)

[110] Yash Gupta, Yusaira Khan, Keheliya Gallaba, and Shane McIntosh. 2017. The Impact of the Adoption of Continuous Integration on Developer Attraction and Retention. In *MSR*. IEEE. (cited on Page 29)

[111] Emitza Guzman, David Azócar, and Yang Li. 2014. Sentiment analysis of commit comments in GitHub: an empirical study. In *MSR*. (cited on Page 29 and 59)

[112] Péter Gyimesi, Béla Vancsics, Andrea Stocco, Davood Mazinanian, Arpád Beszédes, Rudolf Ferenc, and Ali Mesbah. 2019. BugsJS: a benchmark of JavaScript bugs. In *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 90–101. (cited on Page 59)

[113] Sarra Habchi, Naouel Moha, and Romain Rouvoy. 2019. The rise of android code smells: Who is to blame?. In *MSR*. IEEE, 445–456. (cited on Page 59)

[114] Sarra Habchi, Romain Rouvoy, and Naouel Moha. 2019. On the survival of android code smells in the wild. In *2019 IEEE/ACM 6th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 87–98. (cited on Page 59)

[115] Mouna Hadj-Kacem and Nadia Bouassida. 2021. A multi-label classification approach for detecting test smells over java projects. *Journal of King Saud University-Computer and Information Sciences* (2021). (cited on Page 59)

[116] K Hanayama, S Matsumoto, and S Kusumoto. 2020. Humpback: Code Completion System for Dockerfiles Based on Language Models. In *Proc. Workshop on Natural Language Processing Advancements for Software Engineering.* (cited on Page 59)

[117] Kaisei Hanayama, Shinsuke Matsumoto, and Shinji Kusumoto. 2021. Development of Code Completion System for Dockerfiles. 38, 4 (2021), 4_53–4_59. (cited on Page)

[118] Mubin Ul Haque and M Ali Babar. 2022. Well begun is half done: An empirical study of exploitability & impact of base-image vulnerabilities. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 1066–1077. (cited on Page 59)

[119] Mark Harman and Peter O'Hearn. 2018. From start-ups to scale-ups: Opportunities and open problems for static and dynamic program analysis. In *2018 IEEE 18th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. IEEE, 1–23. (cited on Page 59)

[120] Johannes Härtel, Marcel Heinz, and Ralf Lämmel. 2018. EMF patterns of usage on GitHub. In *European conference on modelling foundations and applications*. Springer, 216–234. (cited on Page 59)

[121] Shinpei Hayashi, Daiki Hoshino, Jumpei Matsuda, Motoshi Saeki, Takayuki Omori, and Katsuhisa Maruyama. 2015. Historef: A tool for edit history refactoring. In *SANER*. IEEE. (cited on Page 48)

[122] Hao He. 2019. Understanding source code comments at large-scale. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. (cited on Page 59)

[123] Xincheng He, Lei Xu, Xiangyu Zhang, Rui Hao, Yang Feng, and Baowen Xu. 2021. Pyart: Python api recommendation in real-time. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1634–1645. (cited on Page 59)

[124] Geoffrey Hecht and Alexandre Bergel. 2021. Quantifying the adoption of Kotlin on Android stores: Insight from the bytecode. In *2021 IEEE/ACM 8th International Conference on Mobile Software Engineering and Systems (MobileSoft)*. IEEE, 94–98. (cited on Page 59)

[125] Thomas S Heinze, Viktor Stefanko, and Wolfram Amme. 2020. Mining BPMN processes on Github for tool validation and development. In *Enterprise, Business-Process and Information Systems Modeling*. Springer. (cited on Page 59)

[126] Joseph Hejderup, Moritz Beller, Konstantinos Triantafyllou, and Georgios Gousios. 2022. Präzi: from package-based to call-based dependency networks. *Empirical Software Engineering* 27, 5 (2022). (cited on Page 59)

[127] Jordan Henkel, Christian Bird, Shuvendu K. Lahiri, and Thomas W. Reps. 2020. A Dataset of Dockerfiles. In *MSR*, Sunghun Kim, Georgios Gousios, Sarah Nadi, and Joseph Hejderup (Eds.). ACM. (cited on Page 17, 23, 26, 31, 37, 47, and 59)

[128] Jordan Henkel, Denini Silva, Leopoldo Teixeira, Marcelo d'Amorim, and Thomas Reps. 2021. Shipwright: A Human-in-the-Loop System for Dockerfile Repair. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1148–1160. (cited on Page 26 and 59)

[129] Ben Hermann. 2022. What Has Artifact Evaluation Ever Done for Us? *IEEE Security & Privacy* 20, 5 (2022). (cited on Page 1 and 55)

[130] Ben Hermann, Stefan Winter, and Janet Siegmund. 2020. Community Expectations for Research Artifacts and Evaluation Processes. In *ESEC/FSE*. ACM. (cited on Page 55)

[131] Robert Heumüller, Sebastian Nielebock, Jacob Krüger, and Frank Ortmeier. 2020. Publish or Perish, but do not Forget Your Software Artifacts. *Empirical Software Engineering* (2020). (cited on Page 1 and 55)

[132] Robert Heumüller, Sebastian Nielebock, and Frank Ortmeier. 2021. Exploit those code reviews! bigger data for deeper learning. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* (cited on Page 59)

[133] Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. 2016. Usage, costs, and benefits of continuous integration in open-source projects. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 426–437. (cited on Page 29 and 59)

[134] Abram Hindle, Alex Wilson, Kent Rasmussen, E Jed Barlow, Joshua Charles Campbell, and Stephen Romansky. 2014. Greenminer: A hardware based mining software repositories software energy consumption framework. In *MSR*. (cited on Page 59)

[135] Thomas Hirsch and Birgit Hofer. 2022. A systematic literature review on benchmarks for evaluating debugging approaches. *Journal of Systems and Software* (2022). (cited on Page 59)

[136] Truong Ho-Quang, Michel RV Chaudron, Gregorio Robles, and Guntur Budi Herwanto. 2019. Towards an infrastructure for empirical research into software architecture: challenges and directions. In *2019 IEEE/ACM 2nd International Workshop on Establishing the Community-Wide Infrastructure for Architecture-Based Software Engineering (ECASE)*. IEEE, 34–41. (cited on Page 59)

[137] Eric Horton and Chris Parnin. 2018. Gistable: Evaluating the executability of python code snippets on github. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 217–227. (cited on Page 59)

[138] Jiayi Hua and Haoyu Wang. 2021. On the Effectiveness of Deep Vulnerability Detectors to Simple Stupid Bug Detection. In *MSR*. IEEE. (cited on Page 28)

[139] Ahmed Imam, Tapajit Dey, Alexander Nolte, Audris Mockus, and James D Herbsleb. 2021. The Secret Life of Hackathon Code Where does it come from and where does it go?. In *MSR*. IEEE, 68–79. (cited on Page 59)

[140] Md Rakibul Islam and Minhaz F. Zibran. 2017. Insights into Continuous Integration Build Failures. In *MSR*. IEEE. (cited on Page 28)

[141] Samireh Jalali and Claes Wohlin. 2012. Systematic Literature Studies: Database Searches vs. Backward Snowballing. In *ESEM*. ACM. (cited on Page 11)

[142] Kevin Jesse and Premkumar T Devanbu. 2022. ManyTypes4TypeScript: A Comprehensive TypeScript Dataset for Sequence-Based Type Inference. (2022). (cited on Page 59)

[143] Nan Jiang, Thibaud Lutellier, and Lin Tan. 2021. Cure: Code-aware neural machine translation for automatic program repair. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1161–1173. (cited on Page 59)

[144] Matthieu Jimenez. 2018. *Evaluating vulnerability prediction models.* Ph. D. Dissertation. University of Luxembourg, Luxembourg. (cited on Page 59)

[145] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. 2014. The promises and perils of mining github. In *MSR*. (cited on Page 29 and 59)

[146] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. 2014. The Promises and Perils of Mining GitHub. In *MSR*. ACM. (cited on Page 55)

[147] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. 2016. An In-Depth Study of the Promises and Perils of Mining GitHub. *Empirical Software Engineering* 21, 5 (2016). (cited on Page 55)

[148] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M German, and Daniela Damian. 2016. An in-depth study of the promises and perils of mining GitHub. *Empirical Software Engineering* 21, 5 (2016). (cited on Page 59)

[149] Arthur V. Kamienski, Luisa Palechor, Cor-Paul Bezemer, and Abram Hindle. 2021. PySStuBs: Characterizing Single-Statement Bugs in Popular Open-Source Python Projects. In *MSR*. IEEE. (cited on Page 28)

[150] Rafael-Michael Karampatsis and Charles Sutton. 2020. How Often Do Single-Statement Bugs Occur?: The ManySStuBs4J Dataset. In *MSR*, Sunghun Kim, Georgios Gousios, Sarah Nadi, and Joseph Hejderup (Eds.). ACM. (cited on Page 13, 20, 24, 25, 28, 32, 37, 47, and 59)

[151] Rafael-Michael Karampatsis and Charles Sutton. 2020. Scelmo: Source code embeddings from language models. *arXiv preprint arXiv:2004.13214* (2020). (cited on Page 28 and 59)

[152] Anjan Karmakar and Romain Robbes. 2021. What do pre-trained code models know about code?. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1332–1336. (cited on Page 26 and 59)

[153] Andrey Karnauch. 2020. Developer Reputation Estimator: Increasing the Transparency of Developer Contributions in Open Source Software. (2020). (cited on Page 59)

[154] SayedHassan Khatoonabadi, Diego Elias Costa, Rabe Abdalkareem, and Emad Shihab. 2021. On Wasted Contributions: Understanding the Dynamics of Contributor-Abandoned Pull Requests. *arXiv preprint arXiv:2110.15447* (2021). (cited on Page 59)

[155] Misoo Kim, Youngkyoung Kim, and Eunseok Lee. 2021. Denchmark: A Bug Benchmark of Deep Learning-related Software. In *MSR*. IEEE, IEEE, 540–544. https://doi.org/10.1109/MSR52588.2021.00070 (cited on Page 16, 22, 25, 27, 30, 33, 37, 47, 48, and 59)

[156] Misoo Kim, Youngkyoung Kim, and Eunseok Lee. 2022. An Empirical Study of IR-based Bug Localization for Deep Learning-based Software. In *2022 IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 128–139. (cited on Page 25 and 59)

[157] Youngkyoung Kim, Misoo Kim, and Eunseok Lee. 2022. Tracking Down Misguiding Terms for Locating Bugs in Deep Learning-Based Software (Student Abstract). (2022). (cited on Page 59)

[158] Barbara A. Kitchenham, David Budgen, and O. Pearl Brereton. 2015. *Evidence-Based Software Engineering and Systematic Reviews*. CRC Press. (cited on Page 11)

[159] Barbara A. Kitchenham and Stuart Charters. 2007. *Guidelines for Performing Systematic Literature Reviews in Software Engineering*. Technical Report EBSE-2007-01. Keele University. (cited on Page 11)

[160] Łukasz Korzeniowski and Krzysztof Goczyła. 2022. Landscape of Automated Log Analysis: a Systematic Literature Review and Mapping Study. *IEEE Access* (2022). (cited on Page 59)

[161] Zoe Kotti, Konstantinos Kravvaritis, Konstantina Dritsa, and Diomidis Spinellis. 2020. Standing on shoulders or feet? An extended study on the usage of the MSR data papers. *Empirical Software Engineering* 25, 5 (2020). (cited on Page 56)

[162] Zoe Kotti, Konstantinos Kravvaritis, Konstantina Dritsa, and Diomidis Spinellis. 2020. Standing on shoulders or feet? An extended study on the usage of the MSR data papers. *Empirical Software Engineering* 25, 5 (2020). (cited on Page 59)

[163] Zoe Kotti and Diomidis Spinellis. 2019. Standing on shoulders or feet? The usage of the MSR data papers. In *MSR*. IEEE. (cited on Page 56)

[164] Zoe Kotti and Diomidis Spinellis. 2019. Standing on shoulders or feet? the usage of the MSR data papers. In *MSR*. IEEE, 565–576. (cited on Page 59)

[165] Shriram Krishnamurthi. 2013. Artifact Evaluation for Software Conferences. *SIGSOFT Softw. Eng. Notes* 38, 3 (may 2013). (cited on Page 1 and 55)

[166] Shriram Krishnamurthi and Jan Vitek. 2015. The Real Software Crisis: Repeatability as a Core Value. *Commun. ACM* 58, 3 (feb 2015). (cited on Page 55)

[167] Jacob Krüger, Christian Lausberger, Ivonne von Nostitz-Wallwitz, Gunter Saake, and Thomas Leich. 2020. Search. Review. Repeat? An Empirical Study of Threats to Replicating SLR Searches. *Empirical Software Engineering* 25, 1 (2020). (cited on Page 11)

[168] Indika Kumara, Martín Garriga, Angel Urbano Romeu, Dario Di Nucci, Fabio Palomba, Damian Andrew Tamburri, and Willem-Jan van den Heuvel. 2021. The do's and don'ts of infrastructure code: A systematic gray literature review. *Information and Software Technology* 137 (2021). (cited on Page 59)

[169] Jasmine Latendresse, Rabe Abdalkareem, Diego Elias Costa, and Emad Shihab. 2021. How Effective is Continuous Integration in Indicating Single-Statement Bugs?. In *MSR*. IEEE. (cited on Page 28)

[170] Jason Lefever, Yuanfang Cai, Humberto Cervantes, Rick Kazman, and Hongzhou Fang. 2021. On the lack of consensus among technical debt detection tools. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 121–130. (cited on Page 26 and 59)

[171] Li Li, Jun Gao, Pingfan Kong, Haoyu Wang, Mengyu Huang, Yuan-Fang Li, and Tegawendé F Bissyandé. 2020. KnowledgeZooClient: constructing knowledge graph for Android. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)*. IEEE, 73–78. (cited on Page 59)

[172] Wen Li, Xiaoqin Fu, and Haipeng Cai. 2021. Androct: ten years of app call traces in android. In *MSR*. IEEE, 570–574. (cited on Page 59)

[173] Wen Li, Li Li, and Haipeng Cai. 2022. On the Vulnerability Proneness of Multilingual Code. In *ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. (cited on Page 59)

[174] Yi Li, Shaohua Wang, and Tien N Nguyen. 2020. Dlfix: Context-based code transformation learning for automated program repair. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. (cited on Page 28 and 59)

[175] Yi Li, Shaohua Wang, and Tien N Nguyen. 2021. Vulnerability detection with fine-grained interpretations. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. (cited on Page 27 and 59)

[176] Yi Li, Chenguang Zhu, Julia Rubin, and Marsha Chechik. 2017. FHistorian: Locating features in version histories. In *Proceedings of the 21st International Systems and Software Product Line Conference-Volume A*. (cited on Page 26 and 59)

[177] Yi Li, Chenguang Zhu, Julia Rubin, and Marsha Chechik. 2018. CSlicerCloud: a web-based semantic history slicing framework. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. (cited on Page 26 and 59)

[178] Zhixing Li, Yue Yu, Minghui Zhou, Tao Wang, Gang Yin, Long Lan, and Huaimin Wang. 2020. Redundancy, context, and preference: An empirical study of duplicate pull requests in OSS projects. *IEEE Transactions on Software Engineering* (2020). (cited on Page 27 and 59)

[179] Zhi-Xing Li, Yue Yu, Tao Wang, Gang Yin, Xin-Jun Mao, and Huai-Min Wang. 2021. Detecting duplicate contributions in pull-based model combining textual and change similarities. *Journal of Computer Science and Technology* 36, 1 (2021). (cited on Page 27 and 59)

[180] Lu Liang, Yong Li, Ming Wen, and Ying Liu. 2022. KG4Py: A toolkit for generating Python knowledge graph and code semantic search. *Connection Science* 34, 1 (2022). (cited on Page 26 and 59)

[181] Jackson A Prado Lima and Silvia R Vergilio. 2020. Test Case Prioritization in Continuous Integration environments: A systematic mapping study. *Information and Software Technology* 121 (2020). (cited on Page 59)

[182] Changyuan Lin, Sarah Nadi, and Hamzeh Khazaei. 2020. A large-scale data set and an empirical study of docker images hosted on docker hub. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 371–381. (cited on Page 59)

[183] Bingchang Liu, Guozhu Meng, Wei Zou, Qi Gong, Feng Li, Min Lin, Dandan Sun, Wei Huo, and Chao Zhang. 2020. A large-scale empirical study on vulnerability distribution within projects and the lessons learned. In *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*. IEEE, 1547–1559. (cited on Page 59)

[184] Kui Liu, Anil Koyuncu, Dongsun Kim, and Tegawendé F Bissyandé. 2019. Avatar: Fixing semantic bugs with fix patterns of static analysis violations. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 1–12. (cited on Page 59)

[185] Kui Liu, Anil Koyuncu, Dongsun Kim, and Tegawendé F Bissyandé. 2019. TBar: Revisiting template-based automated program repair. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*. (cited on Page)

[186] Kui Liu, Shangwen Wang, Anil Koyuncu, Kisub Kim, Tegawendé F Bissyandé, Dongsun Kim, Peng Wu, Jacques Klein, Xiaoguang Mao, and Yves Le Traon. 2020. On the efficiency of test suite based program repair: A systematic assessment of 16 automated repair systems for java programs. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. (cited on Page 59)

[187] Pei Liu, Li Li, Yanjie Zhao, Xiaoyu Sun, and John Grundy. 2020. AndroZooOpen: Collecting Large-scale Open Source Android Apps for the Research Community. In *MSR*, Sunghun Kim, Georgios Gousios, Sarah Nadi, and Joseph Hejderup (Eds.). ACM. (cited on Page 17, 23, 26, 31, 37, 38, 42, 47, and 59)

[188] Pei Liu, Yanjie Zhao, Haipeng Cai, Mattia Fazzini, John Grundy, and Li Li. 2022. Automatically Detecting API-induced Compatibility Issues in Android Apps: A Comparative Analysis (Replicability Study). *arXiv preprint arXiv:2205.15561* (2022). (cited on Page 25 and 59)

[189] Nick Lodewijks. [n. d.]. Clone-and-Own. ([n. d.]). (cited on Page 59)

[190] Nick Lodewijks. 2017. Analysis of a clone-and-own industrial automation system: An exploratory study. *SATToSE* (2017). (cited on Page 59)

[191] José Antonio Hernández López and Jesús Sánchez Cuadrado. 2020. MAR: a structure-based search engine for models. In *Proceedings of the 23rd ACM/IEEE international conference on model driven engineering languages and systems*. (cited on Page 59)

[192] Zhigang Lu, Jiwei Xu, Yuewen Wu, Tao Wang, and Tao Huang. 2019. An empirical case study on the temporary file smell in dockerfiles. *IEEE Access* 7 (2019). (cited on Page 59)

[193] Nikola Luburić, Simona Prokić, Katarina-Glorija Grujić, Jelena Slivka, Aleksandar Kovačević, Goran Sladić, and Dragan Vidaković. 2022. Towards a systematic approach to manual annotation of code smells. (2022). (cited on Page 59)

[194] Stephan Lukasczyk, Florian Kroiß, and Gordon Fraser. 2021. An Empirical Study of Automated Unit Test Generation for Python. *arXiv preprint arXiv:2111.05003* (2021). (cited on Page 26 and 59)

[195] Thibaud Lutellier, Hung Viet Pham, Lawrence Pang, Yitong Li, Moshi Wei, and Lin Tan. 2020. Coconut: combining context-aware neural translation models using ensemble for program repair. In *Proceedings of the 29th ACM SIGSOFT international symposium on software testing and analysis*. (cited on Page 59)

[196] Yuxing Ma. 2020. Software Supply Chain Development and Application. (2020). (cited on Page 26 and 59)

[197] Yuxing Ma, Chris Bogart, Sadika Amreen, Russell Zaretzki, and Audris Mockus. 2019. World of code: an infrastructure for mining the universe of open source VCS data. In *MSR*, Margaret-Anne D. Storey, Bram Adams, and Sonia Haiduc (Eds.). IEEE / ACM. (cited on Page 18, 19, 24, 28, 32, 37, 38, 40, 47, 48, and 59)

[198] Yuxing Ma, Tapajit Dey, Chris Bogart, Sadika Amreen, Marat Valiev, Adam Tutko, David Kennard, Russell Zaretzki, and Audris Mockus. 2021. World of code: Enabling a research workflow for mining and analyzing the universe of open source vcs data. *Empirical Software Engineering* 26, 2 (2021). (cited on Page 59)

[199] Fernanda Madeiral and Thomas Durieux. 2021. A large-scale study on human-cloned changes for automated program repair. In *MSR*. IEEE. (cited on Page 28)

[200] Fernanda Madeiral, Simon Urli, Marcelo Maia, and Martin Monperrus. 2019. Bears: An extensible java bug benchmark for automatic program repair studies. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 468–478. (cited on Page 28 and 59)

[201] Lech Madeyski and Marcin Kawalerowicz. 2017. Continuous Defect Prediction: The Idea and a Related Dataset. In *MSR*. IEEE. (cited on Page 29)

[202] Ivano Malavolta, Roberto Verdecchia, Bojan Filipovic, Magiel Bruntink, and Patricia Lago. 2018. How maintainability issues of android apps evolve. In *2018 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 334–344. (cited on Page 59)

[203] Marco Manglaviti, Eduardo Coronado-Montoya, Keheliya Gallaba, and Shane McIntosh. 2017. An Empirical Study of the Personnel Overhead of Continuous Integration. In *MSR*. IEEE. (cited on Page 28)

[204] Vadim Markovtsev and Waren Long. 2018. Public git archive: a big code dataset for all. In *MSR*, Andy Zaidman, Yasutaka Kamei, and Emily Hill (Eds.). ACM. (cited on Page 19, 24, 27, 32, 37, 47, 49, and 59)

[205] Vadim Markovtsev, Waren Long, Egor Bulychev, Romain Keramitas, Konstantin Slavnov, and Gabor Markowski. 2018. Splitting source code identifiers using bidirectional lstm recurrent neural network. *arXiv preprint arXiv:1805.11651* (2018). (cited on Page 27 and 59)

[206] Pedro Martins, Rohan Achar, and Cristina V. Lopes. 2018. 50K-C: a dataset of compilable, and compiled, Java projects. In *MSR*, Andy Zaidman, Yasutaka Kamei, and Emily Hill (Eds.). ACM. (cited on Page 18, 23, 26, 31, 37, 47, and 59)

[207] Ehsan Mashhadi and Hadi Hemmati. 2021. Applying CodeBERT for Automated Program Repair of Java Simple Bugs. In *MSR*. IEEE. (cited on Page 28)

[208] Ehsan Mashhadi and Hadi Hemmati. 2021. Applying codebert for automated program repair of java simple bugs. In *MSR*. IEEE, 505–509. (cited on Page 28 and 59)

[209] Bruno Gois Mateus and Matias Martinez. 2020. On the adoption, usage and evolution of Kotlin features in Android development. In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM).* (cited on Page 59)

[210] Vasileios Matsoukas, Themistoklis Diamantopoulos, Michail D Papamichail, and Andreas L Symeonidis. 2020. Towards Analyzing Contributions from Software Repositories to Optimize Issue Assignment. In *2020 IEEE 20th International Conference on Software Quality, Reliability and Security (QRS).* IEEE, 243–253. (cited on Page 59)

[211] Pooya Rostami Mazrae, Maliheh Izadi, and Abbas Heydarnoori. 2021. Automated Recovery of Issue-Commit Links Leveraging Both Textual and Non-textual Data. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME).* IEEE, 263–273. (cited on Page 26 and 59)

[212] Daniel Mendez, Daniel Graziotin, Stefan Wagner, and Heidi Seibold. 2020. Open science in software engineering. In *Contemporary empirical methods in software engineering.* Springer. (cited on Page 1 and 55)

[213] Daniel Méndez Fernández, Martin Monperrus, Robert Feldt, and Thomas Zimmermann. 2019. The open science initiative of the Empirical Software Engineering journal. *Empirical Software Engineering* 24 (2019). (cited on Page 1 and 55)

[214] Josh GM Mengerink, Jeroen Noten, Ramon RH Schiffelers, Mark GJ van den Brand, and Alexander Serebrenik. 2017. A Case of Industrial vs. Open-source OCL: Not So Different After All.. In *MoDELS (Satellite Events).* (cited on Page 27 and 59)

[215] Josh GM Mengerink, Jeroen Noten, and Alexander Serebrenik. 2019. Empowering OCL research: a large-scale corpus of open-source data from GitHub. *Empirical Software Engineering* 24, 3 (2019). (cited on Page)

[216] Josh GM Mengerink, Alexander Serebrenik, Ramon RH Schiffelers, and Mark GJ van den Brand. 2017. Automated analyses of model-driven artifacts: obtaining insights into industrial application of MDE. In *Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement.* (cited on Page 59)

[217] Amir M. Mir, Evaldas Latoskinas, and Georgios Gousios. 2021. ManyTypes4Py: A Benchmark Python Dataset for Machine Learning-based Type Inference. In *MSR*. IEEE. (cited on Page 17, 23, 26, 31, 37, 47, and 59)

[218] Amir M Mir, Evaldas Latoškinas, Sebastian Proksch, and Georgios Gousios. 2022. Type4Py: practical deep similarity learning-based type inference for python. In *Proceedings of the 44th International Conference on Software Engineering.* (cited on Page 59)

[219] Audris Mockus, Diomidis Spinellis, Zoe Kotti, and Gabriel John Dusing. 2020. A Complete Set of Related Git Repositories Identified via Community Detection Approaches Based on Shared Commits. In *MSR*, Sunghun Kim, Georgios Gousios, Sarah Nadi, and Joseph Hejderup (Eds.). ACM. (cited on Page 18, 23, 26, 28, 31, 37, 47, and 59)

[220] Balazs Mosolygo, Norbert Vandor, Gabor Antal, and Peter Hegedus. 2021. On the Rise and Fall of Simple Stupid Bugs: a Life-Cycle Analysis of SStuBs. In *MSR*. IEEE. (cited on Page 28)

[221] Nuthan Munaiah, Steven Kroh, Craig Cabrey, and Meiyappan Nagappan. 2017. Curating github for engineered software projects. *Empirical Software Engineering* 22, 6 (2017). (cited on Page 29 and 59)

[222] Sandeep Muvva, A Eashaan Rao, and Sridhar Chimalakonda. 2020. BuGL–A Cross-Language Dataset for Bug Localization. *arXiv preprint arXiv:2004.08846* (2020). (cited on Page 59)

[223] Ward Muylaert and Coen De Roover. 2017. Prevalence of Botched Code Integrations. In *MSR*. IEEE. (cited on Page 29)

[224] Phuong T Nguyen, Juri Di Rocco, Claudio Di Sipio, Davide Di Ruscio, and Massimiliano Di Penta. 2021. Recommending api function calls and code snippets to support software development. *IEEE Transactions on Software Engineering* (2021). (cited on Page 28 and 59)

[225] Ansong Ni and Ming Li. 2017. Cost-Effective Build Outcome Prediction Using Cascaded Classifiers. In *MSR*. IEEE. (cited on Page 28)

[226] Sebastian Nielebock, Paul Blockhaus, Jacob Krüger, and Frank Ortmeier. 2021. Android-Compass: A Dataset of Android Compatibility Checks in Code Repositories. In *MSR*. IEEE. (cited on Page 16, 22, 25, 30, 37, 38, 47, and 59)

[227] Sebastian Nielebock, Paul Blockhaus, Jacob Krüger, and Frank Ortmeier. 2021. An Experimental Analysis of Graph-Distance Algorithms for Comparing API Usages. *arXiv preprint arXiv:2108.12511* (2021). (cited on Page 59)

[228] Sebastian Nielebock, Paul Blockhaus, Jacob Krüger, and Frank Ortmeier. 2022. Automated Change Rule Inference for Distance-Based API Misuse Detection. *arXiv preprint arXiv:2207.06665* (2022). (cited on Page 25)

[229] Sebastian Nielebock, Robert Heumüller, Kevin Michael Schott, and Frank Ortmeier. 2021. Guided pattern mining for API misuse detection by change-based code analysis. *Automated Software Engineering* 28, 2 (2021). (cited on Page 59)

[230] Georgios Nikitopoulos, Konstantina Dritsa, Panos Louridas, and Dimitris Mitropoulos. 2021. CrossVul: a cross-language vulnerability dataset with commit data. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* (cited on Page 59)

[231] Ana Filipa Nogueira and Mário Zenha-Rela. 2021. Monitoring a ci/cd workflow using process mining. *SN Computer Science* 2, 6 (2021). (cited on Page 59)

[232] Yu Nong, Yuzhe Ou, Michael Pradel, Feng Chen, and Haipeng Cai. 2022. Generating Realistic Vulnerabilities via Neural Code Editing: An Empirical Study. In *ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE).* (cited on Page 27 and 59)

[233] Jeroen Noten, Josh Mengerink, and Alexander Serebrenik. 2017. A data set of OCL expressions on GitHub. In *MSR*, Jesús M. González-Barahona, Abram Hindle, and Lin Tan (Eds.). IEEE Computer Society. (cited on Page 18, 23, 27, 31, 37, 47, and 59)

[234] Martin Odermatt, Diego Marcilio, and Carlo A Furia. 2022. Static Analysis Warnings and Automatic Fixing: A Replication for C# Projects. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 805–816. (cited on Page 27 and 59)

[235] Ruben Opdebeeck and Coen De Roover. 2021. Using Program Dependence Graphs to Detect Misunderstandings of Ansible's Variable Precedence and Expression Evaluation Semantics. In *20th Belgium-Netherlands Software Evolution Workshop (BENEVOL 2021).* (cited on Page 25 and 59)

[236] Ruben Opdebeeck, Ahmed Zerouali, and Coen De Roover. 2022. Smelly Variables in Ansible Infrastructure Code: Detection, Prevalence, and Lifetime. In *MSR*. ACM. (cited on Page 25 and 59)

[237] Ruben Opdebeeck, Ahmed Zerouali, and Coen De Roover. 2021. Andromeda: A Dataset of Ansible Galaxy Roles and Their Evolution. In *MSR*. IEEE. (cited on Page 16, 22, 25, 30, 37, 47, and 59)

[238] Ruben Opdebeeck, Ahmed Zerouali, Camilo Velázquez-Rodríguez, and Coen De Roover. 2021. On the practice of semantic versioning for Ansible galaxy roles: An empirical study and a change classification model. *Journal of Systems and Software* 182 (2021). (cited on Page 25 and 59)

[239] Gerardo Orellana, Gulsher Laghari, Alessandro Murgia, and Serge Demeyer. 2017. On the Differences between Unit and Integration Testing in the TravisTorrent Dataset. In *MSR*. IEEE. (cited on Page 28)

[240] Jordan Ott, Abigail Atchison, and Erik J Linstead. 2019. Exploring the applicability of low-shot learning in mining software repositories. *Journal of Big Data* 6, 1 (2019). (cited on Page 59)

[241] Klerisson V. R. Paixao, Cricia Z. Felicio, Fernanda M. Delfim, and Marcelo De A. Maia. 2017. On the Interplay between Non-Functional Requirements and Builds on Continuous Integration. In *MSR*. IEEE. (cited on Page 28)

[242] Matheus Paixão, Jens Krinke, DongGyun Han, and Mark Harman. 2018. CROP: linking code reviews to source code changes. In *MSR*, Andy Zaidman, Yasutaka Kamei, and Emily Hill (Eds.). ACM. (cited on Page 18, 23, 27, 32, 37, 38, 47, and 59)

[243] Matheus Paixao, Jens Krinke, DongGyun Han, Chaiyong Ragkhitwetsagul, and Mark Harman. 2019. The impact of code review on architectural changes. *IEEE Transactions on Software Engineering* 47, 5 (2019). (cited on Page 27 and 59)

[244] Matheus Paixao and Paulo Henrique Maia. 2019. Rebasing in code review considered harmful: A large-scale empirical investigation. In *2019 19th international working conference on source code analysis and manipulation (SCAM)*. IEEE, 45–55. (cited on Page 27)

[245] Matheus Paixão, Anderson Uchôa, Ana Carla Bibiano, Daniel Oliveira, Alessandro Garcia, Jens Krinke, and Emilio Arvonio. 2020. Behind the intents: An in-depth empirical study on software refactoring in modern code review. In *MSR*. (cited on Page 27 and 59)

[246] Rongqi Pan, Mojtaba Bagherzadeh, Taher A Ghaleb, and Lionel Briand. 2022. Test case selection and prioritization using machine learning: a systematic literature review. *Empirical Software Engineering* 27, 2 (2022). (cited on Page 59)

[247] Sebastiano Panichella, Gerardo Canfora, and Andrea Di Sorbo. 2021. "Won't We Fix this Issue?" Qualitative characterization and automated identification of wontfix issues on GitHub. *Information and Software Technology* 139 (2021). (cited on Page 59)

[248] Luca Pascarella, Franz-Xaver Geiger, Fabio Palomba, Dario Di Nucci, Ivano Malavolta, and Alberto Bacchelli. 2018. Self-reported activities of android developers. In *2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 144–155. (cited on Page 59)

[249] Luca Pascarella, Davide Spadini, Fabio Palomba, and Alberto Bacchelli. 2019. On the effect of code review on code smells. *arXiv preprint arXiv:1912.10098* (2019). (cited on Page 27 and 59)

[250] Biagio Peccerillo, Mirco Mannino, Andrea Mondelli, and Sandro Bartolini. 2022. A survey on hardware accelerators: Taxonomy, trends, challenges, and perspectives. *Journal of Systems Architecture* (2022). (cited on Page 59)

[251] Fabiano Pecorelli, Gemma Catolino, Filomena Ferrucci, Andrea De Lucia, and Fabio Palomba. 2020. Testing of mobile applications in the wild: A large-scale empirical study on android apps. In *Proceedings of the 28th international conference on program comprehension.* (cited on Page 59)

[252] Yun Peng, Cuiyun Gao, Zongjie Li, Bowei Gao, David Lo, Qirun Zhang, and Michael Lyu. 2022. Static inference meets deep learning: a hybrid type inference approach for python. In *Proceedings of the 44th International Conference on Software Engineering.* (cited on Page 26 and 59)

[253] Jorge Perianez-Pascual, Roberto Rodriguez-Echeverria, Loli Burgueño, and Jordi Cabot. 2020. Towards the optical character recognition of DSLs. In *Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering.* (cited on Page 27 and 59)

[254] Anthony Peruma and Christian D. Newman. 2021. On the Distribution of "Simple Stupid Bugs" in Unit Test Files: An Exploratory Study. In *MSR*. IEEE. (cited on Page 28)

[255] Seth Petre. [n. d.]. 12 Reasons Why Data Is Important. https://www.c-q-l.org/wp-content/uploads/2019/12/12-Reasons-Why-Data-Is-Important.pdf (cited on Page 5)

[256] Peter Pickerill, Heiko Joshua Jungen, Mirosław Ochodek, Michał Maćkowiak, and Miroslaw Staron. 2020. PHANTOM: Curating GitHub for engineered software projects using time-series clustering. *Empirical Software Engineering* 25, 4 (2020). (cited on Page 59)

[257] Antoine Pietri. 2021. *Organizing the graph of public software development for large-scale mining.* Ph. D. Dissertation. Inria. (cited on Page 26 and 59)

[258] Antoine Pietri, Guillaume Rousseau, and Stefano Zacchiroli. [n. d.]. Quantifying Exogenous Software Forks. ([n. d.]). (cited on Page 59)

[259] Antoine Pietri, Diomidis Spinellis, and Stefano Zacchiroli. 2019. The software heritage graph dataset: public software development under one roof. In *MSR*, Margaret-Anne D. Storey, Bram Adams, and Sonia Haiduc (Eds.). IEEE / ACM. (cited on Page 17, 19, 24, 27, 32, 37, 38, 47, and 59)

[260] Antoine Pietri, Diomidis Spinellis, and Stefano Zacchiroli. 2020. The Software Heritage Graph Dataset: Large-scale Analysis of Public Software Development History. In *MSR*, Sunghun Kim, Georgios Gousios, Sarah Nadi, and Joseph Hejderup (Eds.). ACM. (cited on Page 17, 23, 26, 31, 37, 38, 47, and 59)

[261] Antoine Pietri, Diomidis Spinellis, and Stefano Zacchiroli. 2020. The Software Heritage Graph Dataset: Large-Scale Analysis of Public Software Development History. In *MSR*. ACM. (cited on Page 55)

[262] Dmitry Pogrebnoy, Ivan Kuznetsov, Yaroslav Golubev, Vladislav Tankov, and Timofey Bryksint. 2021. Sorrel: an IDE plugin for managing licenses and detecting license incompatibilities. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME).* IEEE, 574–578. (cited on Page 27 and 59)

[263] Serena Elisa Ponta, Henrik Plate, Antonino Sabetta, Michele Bezzi, and Cédric Dangremont. 2019. A manually-curated dataset of fixes to vulnerabilities of open-source software. In *MSR*. IEEE, 383–387. (cited on Page 59)

[264] Paolo Ernesto Prinetto, Dott Riccardo Bortolameotti, Giuseppe Massaro, and Lorenzo Antonio De Giorgi. [n. d.]. Security Misconfigurations Detection and Repair in Dockerfile. ([n. d.]). (cited on Page 26 and 59)

[265] Sebastian Proksch, Sven Amann, and Sarah Nadi. 2018. Enriched Event Streams: A General Dataset for Empirical Studies on In-IDE Activities of Software Developers. In *MSR*. (cited on Page 13)

[266] Md Rafiqul Islam Rabin, Vincent J Hellendoorn, and Mohammad Amin Alipour. 2021. Understanding neural code intelligence through program simplification. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. (cited on Page 59)

[267] Abir Rahali and Moulay A Akhloufi. 2021. MalBERT: Malware Detection using Bidirectional Encoder Representations from Transformers. In *2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 3226–3231. (cited on Page 26 and 59)

[268] Abir Rahali and Moulay A Akhloufi. 2021. Malbert: Using transformers for cybersecurity and malicious software detection. *arXiv preprint arXiv:2103.03806* (2021). (cited on Page 26 and 59)

[269] Thomas Rausch, Waldemar Hummer, Philipp Leitner, and Stefan Schulte. 2017. An empirical analysis of build failures in the continuous integration workflows of java-based open-source software. In *MSR*. IEEE, 345–355. (cited on Page 59)

[270] Marcel Reboucas, Renato O. Santos, Gustavo Pinto, and Fernando Castor. 2017. How Does Contributors' Involvement Influence the Build Status of an Open-Source Software Project?. In *MSR*. IEEE. (cited on Page 28)

[271] Jan Christof Recker, Roman Lukyanenko, Mohammad Jabbari Sabegh, Binny Samuel, and Arturo Castellanos. 2021. From representation to mediation: a new agenda for conceptual modeling research in a digital world. *MIS Quarterly: Management Information Systems* 45, 1 (2021). (cited on Page 59)

[272] David Reid, Mahmoud Jahanshahi, and Audris Mockus. 2022. The Extent of Orphan Vulnerabilities from Code Reuse in Open Source Software. (2022). (cited on Page 59)

[273] Sofia Reis and Rui Abreu. 2021. A ground-truth dataset of real security patches. *arXiv preprint arXiv:2110.09635* (2021). (cited on Page 27 and 59)

[274] Luyao Ren, Shurui Zhou, Christian Kästner, and Andrzej Wąsowski. 2019. Identifying redundancies in fork-based development. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 230–241. (cited on Page 27 and 59)

[275] Sayed Mohsin Reza, Omar Badreddin, and Khandoker Rahad. 2020. Modelmine: a tool to facilitate mining models from open source repositories. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. (cited on Page 59)

[276] Dirk Riehle and Nikolay Harutyunyan. 2019. Open-Source License Compliance in Software Supply Chains. In *Towards Engineering Free/Libre Open Source Software (FLOSS) Ecosystems for Impact and Sustainability*. Springer. (cited on Page 51 and 55)

[277] Gregorio Robles, Truong Ho-Quang, Regina Hebig, Michel R. V. Chaudron, and Miguel Angel Fernández. 2017. An extensive dataset of UML models in GitHub. In *MSR*, Jesús M. González-Barahona, Abram Hindle, and Lin Tan (Eds.). IEEE Computer Society. (cited on Page 19, 24, 25, 32, 37, 42, 47, and 59)

[278] Md Omar Faruk Rokon, Pei Yan, Risul Islam, and Michalis Faloutsos. 2021. Repo2vec: A comprehensive embedding approach for determining repository similarity. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 355–365. (cited on Page 59)

[279] Giovanni Rosa, Luca Pascarella, Simone Scalabrino, Rosalia Tufano, Gabriele Bavota, Michele Lanza, and Rocco Oliveto. 2021. Evaluating szz implementations through a developer-informed oracle. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 436–447. (cited on Page 59)

[280] ARMIN ROTH. 2020. *Faculty of Engineering, Department of Computer Science*. Ph. D. Dissertation. Friedrich-Alexander-Universität Erlangen-Nürnberg. (cited on Page 59)

[281] Guillaume Rousseau, Roberto Di Cosmo, and Stefano Zacchiroli. 2019. Growth and duplication of public source code over time: Provenance tracking at scale. *arXiv preprint arXiv:1906.08076* (2019). (cited on Page 27 and 59)

[282] Guillaume Rousseau, Roberto Di Cosmo, and Stefano Zacchiroli. 2020. Software provenance tracking at the scale of public source code. *Empirical Software Engineering* 25, 4 (2020). (cited on Page 27 and 59)

[283] Mohammad Amin Sadeghi, Shameem Parambath, Ji Lucas, Youssef Meguebli, Maguette Toure, Fawaz Al Qahtani, Ting Yu, and Sanjay Chawla. 2020. Log representation as an interface for log processing applications. (2020). (cited on Page 59)

[284] Ripon K. Saha, Yingjun Lyu, Wing Lam, Hiroaki Yoshida, and Mukul R. Prasad. 2018. Bugs.jar: a large-scale, diverse dataset of real-world Java bugs. In *MSR*, Andy Zaidman, Yasutaka Kamei, and Emily Hill (Eds.). ACM. (cited on Page 20, 24, 28, 32, 37, 38, 42, 47, and 59)

[285] Seemanta Saha et al. 2019. Harnessing evolution for multi-hunk program repair. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*. IEEE, 13–24. (cited on Page 28 and 59)

[286] CM Khaled Saifullah, Muhammad Asaduzzaman, and Chanchal K Roy. 2019. Learning from examples to find fully qualified names of api elements in code snippets. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 243–254. (cited on Page 26 and 59)

[287] Pasquale Salza, Fabio Palomba, Dario Di Nucci, Andrea De Lucia, and Filomena Ferrucci. 2020. Third-party libraries in mobile apps. *Empirical Software Engineering* 25, 3 (2020). (cited on Page 28 and 59)

[288] Jordan Samhi, Tegawendé F Bissyandé, and Jacques Klein. 2022. TriggerZoo: A Dataset of Android Applications Automatically Infected with Logic Bombs. *arXiv preprint arXiv:2203.04448* (2022). (cited on Page 59)

[289] Huascar Sanchez and Briland Hitaj. 2022. Trust in Motion: Capturing Trust Ascendancy in Open-Source Projects using Hybrid AI. *arXiv preprint arXiv:2210.02656* (2022). (cited on Page 59)

[290] Vemuganti Sesha Satvik and Ravikant Gautam. 2022. Summer Internship Report 2022. (2022). (cited on Page 59)

[291] Damien Saucez and Luigi Iannone. 2018. Thoughts and Recommendations from the ACM SIGCOMM 2017 Reproducibility Workshop. *SIGCOMM Comput. Commun. Rev.* 48, 1 (apr 2018). (cited on Page 55)

[292] Damien Saucez, Luigi Iannone, and Olivier Bonaventure. 2019. Evaluating the Artifacts of SIGCOMM Papers. *SIGCOMM Comput. Commun. Rev.* 49, 2 (may 2019). (cited on Page 55)

[293] Gerald Schermann, Sali Zumberi, and Jürgen Cito. 2018. Structured information on state and evolution of dockerfiles on github. In *MSR*, Andy Zaidman, Yasutaka Kamei, and Emily Hill (Eds.). ACM. (cited on Page 19, 23, 25, 32, 37, 47, and 59)

[294] Ramon RH Schiffelers, Yaping Luo, Josh Mengerink, and Mark van den Brand. 2018. Towards Automated Analysis of Model-Driven Artifacts in Industry.. In *MODELSWARD.* (cited on Page 27 and 59)

[295] Marc Schiltz. 2018. Science Without Publication Paywalls: cOAlition S for the Realisation of Full and Immediate Open Access. *PLOS Medicine* 15, 9 (09 2018). (cited on Page 1)

[296] Alexander Schultheiß, Paul Maximilian Bittner, Sascha El-Sharkawy, Thomas Thüm, and Timo Kehrer. 2022. Simulating the evolution of clone-and-own projects with VEVOS. In *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering 2022.* (cited on Page 59)

[297] Abdulkadir Seker, Banu Diri, Halil Arslan, and Mehmet Fatih Amasyalı. 2022. Open Source Software Development Challenges: A Systematic Literature Review on GitHub. *Research Anthology on Agile Software, Software Development, and Testing* (2022). (cited on Page 59)

[298] Yusra Shakeel, Jacob Krüger, Ivonne von Nostitz-Wallwitz, Christian Lausberger, Gabriel C. Durand, Gunter Saake, and Thomas Leich. 2018. (Automated) Literature Analysis - Threats and Experiences. In *SE4Science.* ACM. (cited on Page 11)

[299] Tushar Sharma, Vasiliki Efstathiou, Panos Louridas, and Diomidis Spinellis. 2021. Code smell detection by deep direct-learning and transfer-learning. *Journal of Systems and Software* 176 (2021). (cited on Page 59)

[300] Tushar Sharma and Marouane Kessentini. 2021. QScored: A Large Dataset of Code Smells and Quality Metrics. In *MSR.* IEEE. (cited on Page 15, 22, 25, 30, 37, 38, 47, and 59)

[301] Elena Sherman and Robert Dyer. 2018. Software engineering collaboratories (SEClabs) and collaboratories as a service (CaaS). In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* (cited on Page 59)

[302] Jari Soini, Markku Kuusisto, Petri Rantanen, Mika Saari, and Pekka Sillberg. 2020. A study on an evolution of a data collection system for knowledge representation. In *Information Modelling and Knowledge Bases XXXI.* IOS Press. (cited on Page 59)

[303] Mauricio Soto, Zack Coker, and Claire Le Goues. 2017. Analyzing the Impact of Social Attributes on Commit Integration Success. In *MSR.* IEEE. (cited on Page 28)

[304] César Soto-Valero, Thomas Durieux, and Benoit Baudry. 2021. A longitudinal analysis of bloated java dependencies. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering.* (cited on Page 59)

[305] César Soto-Valero, Thomas Durieux, Nicolas Harrand, and Benoit Baudry. 2022. Coverage-Based Debloating for Java Bytecode. *ACM Computing Surveys (CSUR)* (2022). (cited on Page 59)

[306] Rodrigo Souza and Bruno Silva. 2017. Sentiment Analysis of Travis CI Builds. In *MSR*. IEEE. (cited on Page 28)

[307] Josef Spillner. 2019. Quantitative analysis of cloud function evolution in the AWS serverless application repository. *arXiv preprint arXiv:1905.04800* (2019). (cited on Page 59)

[308] Diomidis Spinellis, Zoe Kotti, Konstantinos Kravvaritis, Georgios Theodorou, and Panos Louridas. 2020. A Dataset of Enterprise-Driven Open Source Software. In *MSR*, Sunghun Kim, Georgios Gousios, Sarah Nadi, and Joseph Hejderup (Eds.). ACM. (cited on Page 17, 22, 25, 26, 30, 37, 38, 47, and 59)

[309] Diomidis Spinellis, Zoe Kotti, and Audris Mockus. 2020. A Dataset for GitHub Repository Deduplication. In *MSR*, Sunghun Kim, Georgios Gousios, Sarah Nadi, and Joseph Hejderup (Eds.). ACM. (cited on Page 18, 23, 26, 31, 37, 47, and 59)

[310] Diomidis Spinellis, Panos Louridas, and Maria Kechagia. 2021. Software evolution: the lifetime of fine-grained elements. *PeerJ Computer Science* 7 (2021). (cited on Page 26 and 59)

[311] Daniel Strüber, Mukelabai Mukelabai, Jacob Krüger, Stefan Fischer, Lukas Linsbauer, Jabier Martinez, and Thorsten Berger. 2019. Facing the truth: benchmarking the techniques for the evolution of variant-rich systems. In *Proceedings of the 23rd International Systems and Software Product Line Conference-Volume A*. (cited on Page 59)

[312] Matúš Sulír, Michaela Bačíková, Matej Madeja, Sergej Chodarev, and Ján Juhár. 2020. Large-scale dataset of local java software build results. *Data* 5, 3 (2020). (cited on Page 59)

[313] Xin Tan and Minghui Zhou. 2019. How to communicate when submitting patches: An empirical study of the Linux kernel. *Proceedings of the ACM on Human-Computer Interaction* 3, CSCW (2019). (cited on Page 25 and 59)

[314] Haoye Tian, Kui Liu, Abdoul Kader Kaboré, Anil Koyuncu, Li Li, Jacques Klein, and Tegawendé F Bissyandé. 2020. Evaluating representation learning of code changes for predicting patch correctness in program repair. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 981–992. (cited on Page 28 and 59)

[315] Christopher S. Timperley, Lauren Herckis, Claire Le Goues, and Michael Hilton. 2021. Understanding and Improving Artifact Sharing in Software Engineering Research. *Empirical Software Engineering* 26, 67 (2021). (cited on Page 55)

[316] Nitin M. Tiwari, Ganesha Upadhyaya, Hoan A. Nguyen, and Hridesh Rajan. 2017. Candoia: A Platform for Building and Sharing Mining Software Repositories Tools as Apps. In *MSR*. IEEE. (cited on Page 55)

[317] Alexander Trautsch, Fabian Trautsch, Steffen Herbold, Benjamin Ledel, and Jens Grabowski. 2020. The smartshark ecosystem for software repository mining. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Companion Proceedings*. (cited on Page 55)

[318] Tomoaki Tsuru, Tasuku Nakagawa, Shinsuke Matsumoto, Yoshiki Higo, and Shinji Kusumoto. 2021. Type-2 Code Clone Detection for Dockerfiles. In *2021 IEEE 15th International Workshop on Software Clones (IWSC)*. IEEE, 1–7. (cited on Page 59)

[319] Leigh Turnbull, Zhiyuan Tan, and Kehinde O Babaagba. 2022. A Generative Neural Network for Enhancing Android Metamorphic Malware Detection based on Behaviour Profiling. In *2022 IEEE Conference on Dependable and Secure Computing (DSC)*. IEEE, 1–9. (cited on Page 26 and 59)

[320] Anderson Uchôa, Caio Barbosa, Daniel Coutinho, Willian Oizumi, Wesley KG Assunçao, Silvia Regina Vergilio, Juliana Alves Pereira, Anderson Oliveira, and Alessandro Garcia. 2021. Predicting design impactful changes in modern code review: A large-scale empirical study. In *MSR*. IEEE, 471–482. (cited on Page 27 and 59)

[321] Anderson Uchôa, Caio Barbosa, Willian Oizumi, Publio Blenílio, Rafael Lima, Alessandro Garcia, and Carla Bezerra. 2020. How does modern code review impact software design degradation? an in-depth empirical study. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 511–522. (cited on Page 27 and 59)

[322] Farhan Ullah, Amjad Alsirhani, Mohammed Mujib Alshahrani, Abdullah Alomari, Hamad Naeem, and Syed Aziz Shah. 2022. Explainable Malware Detection System Using Transformers-Based Transfer Learning and Multi-Model Visual Representation. *Sensors* 22, 18 (2022). (cited on Page 59)

[323] Dheeraj Vagavolu, Vartika Agrahari, Sridhar Chimalakonda, and Akhila Sri Manasa Venigalla. 2021. GE526: A Dataset of Open-Source Game Engines. In *MSR*. IEEE. (cited on Page 15, 22, 25, 30, 37, 47, and 59)

[324] Frenk CJ van Mil, Ayushi Rastogi, and Andy Zaidman. 2021. Promises and Perils of Inferring Personality on GitHub. In *Proceedings of the 15th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. (cited on Page 59)

[325] Bogdan Vasilescu, Vladimir Filkov, and Alexander Serebrenik. 2013. Stackoverflow and github: Associations between software development and crowdsourced knowledge. In *2013 International Conference on Social Computing*. IEEE, 188–195. (cited on Page 29 and 59)

[326] Bogdan Vasilescu, Daryl Posnett, Baishakhi Ray, Mark GJ van den Brand, Alexander Serebrenik, Premkumar Devanbu, and Vladimir Filkov. 2015. Gender and tenure diversity in GitHub teams. In *Proceedings of the 33rd annual ACM conference on human factors in computing systems*. (cited on Page 29 and 59)

[327] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. 2015. Quality and productivity outcomes relating to continuous integration in GitHub. In *Proceedings of the 2015 10th joint meeting on foundations of software engineering*. (cited on Page 29 and 59)

[328] Carmine Vassallo, Sebastiano Panichella, Fabio Palomba, Sebastian Proksch, Andy Zaidman, and Harald C Gall. 2018. Context is king: The developer perspective on the usage of static analysis tools. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 38–49. (cited on Page 29 and 59)

[329] Sruthi Venkatanarayanan, Jens Dietrich, Craig Anslow, and Patrick Lam. [n. d.]. VizAPI: Visualizing Interactions between Java Libraries and Clients. ([n. d.]). (cited on Page 25 and 59)

[330] Elaine Venson, Ting Fung Lam, Bradford Clark, and Barry Boehm. 2021. Analyzing Software Security-related Size and its Relationship with Vulnerabilities in OSS. In *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*. IEEE, 956–965. (cited on Page 26 and 59)

[331] Roberto Verdecchia, Patricia Lago, and Carol de Vries. 2021. VU Research Portal. (2021). (cited on Page 59)

[332] Renan Vieira, Antônio da Silva, Lincoln Rocha, and João Paulo Gomes. 2019. From Reports to Bug-Fix Commits: A 10 Years Dataset of Bug-Fixing Activity from 55 Apache's Open Source Projects. In *Proceedings of the Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering*. (cited on Page 59)

[333] Anna Vlasova, Maria Tigina, Ilya Vlasov, Anastasiia Birillo, Yaroslav Golubev, and Timofey Bryksin. 2022. Lupa: a framework for large scale analysis of the programming language usage. In *MSR*. (cited on Page 27 and 59)

[334] Johannes Wachs, Mariusz Nitecki, William Schueller, and Axel Polleres. 2022. The geography of open source software: Evidence from github. *Technological Forecasting and Social Change* 176 (2022). (cited on Page 59)

[335] Qingye Wang, Bowen Xu, Xin Xia, Ting Wang, and Shanping Li. 2019. Duplicate pull request detection: When time matters. In *Proceedings of the 11th Asia-Pacific Symposium on Internetware*. (cited on Page 27 and 59)

[336] Yuqing Wang, Mika V Mäntylä, Zihao Liu, and Jouni Markkula. 2022. Test automation maturity improves product quality—Quantitative study of open source projects using continuous integration. *Journal of Systems and Software* 188 (2022). (cited on Page 25, 26, and 59)

[337] Kevin Wellenzohn. 2022. *Robust and scalable content-and-structure indexing of semi-structured hierarchical data*. Ph. D. Dissertation. University of Zurich. (cited on Page 26 and 59)

[338] Kevin Wellenzohn, Michael H Böhlen, Sven Helmer, Antoine Pietri, and Stefano Zacchiroli. 2022. Robust and scalable content-and-structure indexing. *The VLDB Journal* (2022). (cited on Page 26 and 59)

[339] Jinfeng Wen, Zhenpeng Chen, and Xuanzhe Liu. 2022. A literature review on serverless computing. *arXiv preprint arXiv:2206.12275* (2022). (cited on Page 59)

[340] Tyler Wendland, Jingyang Sun, Junayed Mahmud, S. M. Hasan Mansur, Steven Huang, Kevin Moran, Julia Rubin, and Mattia Fazzini. 2021. Andror2: A Dataset of Manually-Reproduced Bug Reports for Android apps. In *MSR*. IEEE. (cited on Page 16, 22, 25, 30, 37, 38, 47, and 59)

[341] Dominik Wermke, Noah Wöhler, Jan H Klemmer, Marcel Fourné, Yasemin Acar, and Sascha Fahl. 2022. Committed to Trust: A Qualitative Study on Security & Trust in Open Source Software Projects. In *Proceedings of the 43rd IEEE Symposium on Security and Privacy*. IEEE Computer Society. (cited on Page 59)

[342] Anna-Katharina Wickert, Lars Baumgärtner, Krishna Narasimhan, Michael Schlichtig, and Mira Mezini. 2022. To Fix or Not to Fix: A Critical Study of Crypto-misuses in the Wild. *arXiv preprint arXiv:2209.11103* (2022). (cited on Page 25 and 59)

[343] Stefan Winter, Christopher S. Timperley, Ben Hermann, Jürgen Cito, Jonathan Bell, Michael Hilton, and Dirk Beyer. 2022. A Retrospective Study of One Decade of Artifact Evaluations. In *ESEC/FSE* (Singapore, Singapore) *(ESEC/FSE 2022)*. ACM, New York, NY, USA. (cited on Page 55)

[344] Claes Wohlin. 2014. Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. In *EASE*. ACM. (cited on Page 11)

[345] Lingfei Wu, Peng Cui, Jian Pei, Liang Zhao, and Le Song. 2022. Graph neural networks. In *Graph Neural Networks: Foundations, Frontiers, and Applications*. Springer. (cited on Page 28 and 59)

[346] Xiuheng Wu, Chenguang Zhu, and Yi Li. 2021. Diffbase: A differential factbase for effective software evolution management. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. (cited on Page 26 and 59)

[347] Yiwen Wu, Yang Zhang, Tao Wang, and Huaimin Wang. 2020. Characterizing the occurrence of dockerfile smells in open-source software: An empirical study. *IEEE Access* 8 (2020). (cited on Page 59)

[348] Yi Xie, Wen Li, Yuqing Sun, Elisa Bertino, and Bin Gong. 2022. Subspace Embedding Based New Paper Recommendation. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 1767–1780. (cited on Page 59)

[349] Hui Xu, Zhuangbin Chen, Mingshen Sun, Yangfan Zhou, and Michael R Lyu. 2021. Memory-Safety Challenge Considered Solved? An In-Depth Study with All Rust CVEs. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 1 (2021). (cited on Page 59)

[350] Yisen Xu, Fan Wu, Xiangyang Jia, Lingbo Li, and Jifeng Xuan. 2020. Mining the use of higher-order functions. *Empirical Software Engineering* 25, 6 (2020). (cited on Page 59)

[351] Yulin Xu and Minghui Zhou. 2018. A multi-level dataset of linux kernel patchwork. In *MSR*, Andy Zaidman, Yasutaka Kamei, and Emily Hill (Eds.). ACM. (cited on Page 17, 22, 25, 30, 37, 38, 40, 47, and 59)

[352] Aiko Yamashita, S. Amirhossein Abtahizadeh, Foutse Khomh, and Yann-Gaël Guéhéneuc. 2017. Software evolution and quality data from controlled, multiple, industrial case studies. In *MSR*, Jesús M. González-Barahona, Abram Hindle, and Lin Tan (Eds.). IEEE Computer Society. (cited on Page 17, 23, 25, 31, 37, 38, 47, and 59)

[353] Guanqun Yang, Shay Dineen, Zhipeng Lin, and Xueqing Liu. 2021. Few-Sample Named Entity Recognition for Security Vulnerability Reports by Fine-Tuning Pre-trained Language Models. In *International Workshop on Deployable Machine Learning for Security Defense*. Springer, 55–78. (cited on Page 59)

[354] Yanming Yang, Xin Xia, David Lo, Tingting Bi, John Grundy, and Xiaohu Yang. 2022. Predictive models in software engineering: Challenges and opportunities. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 3 (2022). (cited on Page 59)

[355] Kang Yin, Wei Chen, Jiahong Zhou, Guoquan Wu, and Jun Wei. 2018. STAR: a specialized tagging approach for Docker repositories. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 426–435. (cited on Page 59)

[356] Jean-Gabriel Young, Amanda Casari, Katie McLaughlin, Milo Z Trujillo, Laurent Hébert-Dufresne, and James P Bagrow. 2021. Which contributions count? Analysis of attribution in open source. In *MSR*. IEEE, 242–253. (cited on Page 26 and 59)

[357] Ahmmad Youssef. 2019. *Mining software repositories to determine the impact of team factors on the structural attributes of software*. Ph. D. Dissertation. Brunel University London. (cited on Page 59)

[358] Shengcheng Yu, Chunrong Fang, Tongyu Li, Mingzhe Du, Xuan Li, Jing Zhang, Yexiao Yun, Xu Wang, and Zhenyu Chen. 2021. Automated Mobile App Test Script Intent Generation via Image and Code Understanding. *arXiv preprint arXiv:2107.05165* (2021). (cited on Page 26 and 59)

[359] Yue Yu, Zhixing Li, Gang Yin, Tao Wang, and Huaimin Wang. 2018. A dataset of duplicate pull-requests in github. In *MSR*, Andy Zaidman, Yasutaka Kamei, and Emily Hill (Eds.). ACM. (cited on Page 18, 23, 27, 31, 37, 47, and 59)

[360] Ehsan Zabardast, Javier Gonzalez-Huerta, and Binish Tanveer. 2022. Ownership vs Contribution: Investigating the Alignment Between Ownership and Contribution. In *2022 IEEE 19th International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 30–34. (cited on Page 59)

[361] Stefano Zacchiroli. 2020. Gender differences in public code contributions: a 50-year perspective. *IEEE Software* 38, 2 (2020). (cited on Page 27 and 59)

[362] Fiorella Zampetti, Simone Scalabrino, Rocco Oliveto, Gerardo Canfora, and Massimiliano Di Penta. 2017. How open source projects use static code analysis tools in continuous integration pipelines. In *MSR*. IEEE, 334–344. (cited on Page 29 and 59)

[363] Gianluigi Zavattaro, Jacopo Mauro, Dott Saverio Giallorenzo, Dott Giuseppe De Palma, and Matteo Trentin. [n. d.]. Topology-based Scheduling in Serverless Computing Platforms. ([n. d.]). (cited on Page 25 and 59)

[364] Long Zhang, Deepika Tiwari, Brice Morin, Benoit Baudry, and Martin Monperrus. 2019. Automatic Observability for Dockerized Java Applications. *arXiv preprint arXiv:1912.06914* (2019). (cited on Page 59)

[365] Xunhui Zhang, Ayushi Rastogi, and Yue Yu. 2020. On the Shoulders of Giants: A New Dataset for Pull-based Development Research. In *MSR*, Sunghun Kim, Georgios Gousios, Sarah Nadi, and Joseph Hejderup (Eds.). ACM. (cited on Page 17, 22, 25, 30, 37, 47, and 59)

[366] Xin Zhang, Rongjie Yan, Jiwei Yan, Baoquan Cui, Jun Yan, and Jian Zhang. 2022. ExcePy: A Python Benchmark for Bugs with Python Built-in Types. In *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 856–866. (cited on Page 59)

[367] Xunhui Zhang, Yue Yu, Tao Wang, Ayushi Rastogi, and Huaimin Wang. 2022. Pull request latency explained: An empirical overview. *Empirical Software Engineering* 27, 6 (2022). (cited on Page 59)

[368] Yanjie Zhao, Li Li, Haoyu Wang, Qiang He, and John Grundy. 2022. APIMatchmaker: Matching the Right APIs for Supporting the Development of Android Apps. *IEEE Transactions on Software Engineering* (2022). (cited on Page 59)

[369] Yangyang Zhao, Alexander Serebrenik, Yuming Zhou, Vladimir Filkov, and Bogdan Vasilescu. 2017. The impact of continuous integration on other software development practices: a large-scale empirical study. In *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 60–71. (cited on Page 59)

[370] Jiahong Zhou, Wei Chen, Guoquan Wu, and Jun Wei. 2019. SemiTagRec: a semi-supervised learning based tag recommendation approach for Docker repositories. In *International Conference on Software and Systems Reuse*. Springer, 132–148. (cited on Page 59)

[371] Shurui Zhou, Bogdan Vasilescu, and Christian Kästner. 2019. What the fork: a study of inefficient and efficient forking practices in social coding. In *Proceedings of the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*. (cited on Page 59)

[372] Chenguang Zhu, Yi Li, Julia Rubin, and Marsha Chechik. 2017. A dataset for dynamic discovery of semantic changes in version controlled software histories. In *MSR*, Jesús M. González-Barahona, Abram Hindle, and Lin Tan (Eds.). IEEE Computer Society. (cited on Page 18, 23, 26, 31, 34, 37, 47, 49, and 59)

[373] Chenguang Zhu, Yi Li, Julia Rubin, and Marsha Chechik. 2020. GenSlice: Generalized semantic history slicing. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 81–91. (cited on Page 26 and 59)

[374] Wenhan Zhu and Michael W. Godfrey. 2021. Mea culpa: How developers fix their own simple bugs differently from other developers. In *MSR*. IEEE. (cited on Page 25 and 28)

[375] Noa Zilberman and Andrew W. Moore. 2020. Thoughts about Artifact Badging. *SIGCOMM Comput. Commun. Rev.* 50, 2 (may 2020). (cited on Page 55)

[376] Thomas Zimmermann. 2016. Card-Sorting: From Text to Themes. In *Perspectives on Data Science for Software Engineering*. Elsevier. (cited on Page 14)

Thesis: Analyzing Software Evolution Datasets and Their Use Cases

Name: Kittan

Surname: Sebastian

Date of birth: 23.12.1995

Matriculation no.: 207250

I herewith assure that I wrote the present thesis independently, that the thesis has not been partially or fully submitted as graded academic work and that I have used no other means than the ones indicated. I have indicated all parts of the work in which sources are used according to their wording or to their meaning.

I am aware of the fact that violations of copyright can lead to injunctive relief and claims for damages of the author as well as a penalty by the law enforcement agency

Magdeburg, 27. February 2023