Preprint of a chapter: Tomasz Lewowski and Lech Madeyski, Integrating Research and Practice in Software Engineering, vol. 851 of Studies in Computational Intelligence, chapter ***"Creating evolving project data sets in software engineering"***, pp. 1–14. Springer, 2020. DOI: 10.1007/978-3-030-26574-8_1 [BibTeX]
Draft: http://madeyski.e-informatyka.pl/download/LewowskiMadeyski20SCI.pdf

Creating evolving project data sets in software engineering

Tomasz Lewowski and Lech Madeyski

Abstract While the amount of research in the area of software engineering is ever increasing, it is still a challenge to select a research data set. Quite a number of data sets have been proposed, but we still lack a systematic approach to creating ones that would evolve together with the industry. We aim to present a systematic method of selecting data sets of industry-relevant software projects for the purposes of software engineering research. We present a set of guidelines for filtering GitHub projects and implement those guidelines in a form of an R script. In particular, we select mostly projects from the biggest industrial open source contributors and remove projects in the first quartile in any of several categories from the data set. We use the latest GitHub GraphQL API to select the desired set of repositories. We evaluate the technique on Java projects. Presented technique systematizes methods for creating software development data sets and their evolution. Proposed algorithm has reasonable precision—between 0.65 and 0.80—and can be used as a baseline for further refinements.

Tomasz Lewowski
Wroclaw University of Science and Technology
Faculty of Computer Science and Management
ORCiD: 0000-0003-4897-1263
e-mail: tomasz.lewowski@pwr.edu.pl

Lech Madeyski
Wroclaw University of Science and Technology
Faculty of Computer Science and Management
ORCiD: 0000-0003-3907-3357
e-mail: lech.madeyski@pwr.edu.pl

1 Introduction

Software engineering (SE) body of knowledge expands at an amazing pace, and a lot of concepts investigated by researchers become incorporated into industrial practice — Test-Driven Development (TDD) [22, 34], Continuous TDD (CTDD) [24], pair programming [5] and Agile [10], just to name a few. Obviously, research requires data sets, and data sets on all kinds of software-related issues are ubiquitous. However, even when data sets are published, they often contain data that was selected manually [39] or do not contain runnable scripts that recreate the data set [30].

While there are often legal restrictions which constrain researchers to publish data sets based only on open source software, a comparison between open source code and proprietary code was already made in [32, 19, 21]. While features of the code do not differ substantially between both types of projects, differences may become substantial for project management-related metrics and processes—open source projects tend to have rather flat structure, with spontaneously formed subcommunities [8].

Still, even mitigating problems with those metrics would not be sufficient to make a data set future-proof. Due to unbelievable rate of industry evolution, data sets become obsolete incredibly quickly. There are many reasons for that—including research being successful in solving some problem, abandoning a specific technology or simply changing industrial practices [40]. Regardless of the reasons, this is neither an issue that can be ignored, nor one that can be addressed in a setup with a fixed data set.

Some data cannot be obtained automatically and require manual inspection. In particular, manual data labeling or classification is often needed for data sets meant for machine learning. In data sets meant for machine learning we generally cannot avoid manual work, simply because if we had an algorithm that classifies data and is able to create the data set, we would not need to create this algorithm any more[1]. Since human input is the core of the data set, said data set cannot be recreated automatically. However, even for this kind of data, the data set should evolve together with the domain.

Yet there is also another type of data set—one that is created purely from static data. Examples of such data sets include excerpts from existing source code repositories or issue trackers. This kind of data set does not require any additional modification except API calls to fetch proper records. We argue that this kind of data sets should not be published as static data sets but as regularly updated snapshots, preferably together with tools for automatic refreshing. This would allow researchers to analyze the evolution of the data set (and thus—represented context) and evolve techniques used to tackle their problem. We still perceive value in static data sets, but we believe that their main purpose should be to guarantee research reproducibility.

[1] except for specific cases like insufficient performance or for reverse-engineering, which we ignore in this discussion

Since researchers are interested in various properties and populations, it is understandable that data set creation rules will vary between studies. Some researchers will be interested in all Java projects, others only in Android projects, Python or JavaScript. Some will only want projects of certain scale, others only ones that adhere to some predefined architecture (e.g. MVC) or using some specific constructions or tools. We notice the need for various data sets created for various populations. At the same time, we believe that publishing a static data set for further research (as opposed to providing it for study reproducibility) is not a sound strategy. Instead of providing only the data set, one should also provide a procedure for creating this data set, preferably in form of a runnable script. Only this will provide reproducibility level required to really advance software engineering and enable solid, scientific cross-checks. This is not to say that this script will always generate the same data set—this is never possible to guarantee when using external data sources. However, data sets generated using same method share some characteristics, and these characteristics should be the core concept of an evolving data set, not the records alone.

In this study we present a technique for selecting projects which we believe are relevant for industrial usage in software development companies. This study builds upon one of the internal reports [23] in a research project (supported by National Centre for Research and Development) conducted in *code quest sp. z o.o.* software development company where, e.g., selection of industry relevant projects was proposed. We focus purely on the code repository, which means that data sets created using this method may be used for mining code changes, co-changes, code smells and anti-patterns, but are unlikely to be sufficient for defect prediction or effort estimation. We present and evaluate a data set for Java, but presented technique is sound and, with minor adjustments, can be used equally well for Python, C, Ruby or any other programming language.

The rest of the work is structured as follows: Section 2 describes work in this area already done by other authors. Section 3 contains details of the problem statement and solution. Section 4 describes evaluation of example data set obtained using presented algorithm, then in Section 5 we analyze details of achieved solution. In Section 6 we discuss threats to validity of the study. Section 7 contains the work that will be done next. We conclude the paper in Section 8

2 Related work

There were many attempts to provide a reasonable project data set for software engineering research. Published data sets generally include code repositories [39, 33], issue trackers [20, 16, 29, 30] or only associated metrics [12]. They present a snapshot of projects (or their history), sometimes together with the method used to obtain those. There are also open source tools for experiment management that allow also tracking of data set (e.g. DVC [3]) or ones that work on versioning data alone (e.g. Dat [2]).

GitHub itself is a common platform for creating project data sets—for example, a complete dump of one year activity of GitHub was published as a result of GHTorrent project [14]. Using GitHub as main data source has a number of limitations, which were discussed in detail in [9, 18]. GitHub platform is used not only for analyzing code but also, for example, for analysing sentiment [15, 31], API usage [35] and contribution patterns [7].

A comprehensive list of software engineering-related data sets and resources can be found in [1]. Not mentioned there, but often cited repository software defect prediction data sets can be found in [17].

There were several attempts at defining recommended uses of GitHub for data mining for software engineering projects. For example, authors of [28] manually evaluated 200 repositories and used those to create a machine-learning based tool that extracts repositories that contain common software development practices. Technique for selecting open source projects for teaching software engineering in presented in [37], while authors of [38] focused on community patterns—how developers interact and what social structures they create. Finally, a tool for selecting projects matching certain characteristics from both product and process perspective, integrated not only with Git but also with Jira, is proposed in [11].

In [13] authors propose delivering data sets for machine learning purposes together with data sheets that contain standardized metadata. Technically speaking, this metadata includes also maintenance techniques and evolution rules—however, the paper does not focus in detail on this way of data sheet usage. Presented data sheet is quite informal and meant mostly for human readers, not for automation. Regardless, it is still a sound concept that should make its way into the industry.

3 Research setup

Initially, we aimed to provide another static data set for software engineering research. However, after creating such a data set we found out that in 6 months about 1% of projects were either deleted or moved, thus no longer accessible for reproduction. As a result, the data set cannot be used again to reproduce whole research. Since forking all repositories is not an option due to sheer volume, the data set rusts with time.

We also discovered that many well-known data sets (such as Qualitas Corpus [39]) contain data that is largely irrelevant now—for example it contains projects written in Java 5. The Long Term Support versions now are Java 8 and 11, thus Java 5 patterns should not be used to assess quality of Java code any more. That is simply because the language has evolved so much (generics, lambdas and streams just to name a few new features) that previously used patterns became irrelevant.

After we discovered that providing a static data set is set for failure (or at least providing a very short usefulness period), we concluded that we must be able to easily update the created data sets. However, to do this, data set has to be created in a systematic way, preferably via an automated script. With throughput of contem-

porary servers any reasonable cloning for research purposes can be easily handled, therefore loading the data and calculating all needed metrics does not need to be cached for any reasons other than archiving and providing reproducibility.

We decided to design such a script in a way that would be maximally flexible and would allow researchers to regenerate the data set easily. We concentrate solely on source code management and ignore any data related to project releases, issue management, licensing etc.

It is important to note that, while providing such a script will substantially improve study reproducibility, it is not a full substitute for including data set used during the research. That is because any script will refer to external APIs and databases which are out of researcher's control. Due to that, their content may (and will) change, and responses will typically yield different results, especially months or years after the original study is completed.

Our main requirement was making the rules completely self-tuning, so that it would not be important whether they are applied to languages with massive communities or to ones with modest communities. Therefore, we decided to base nearly all filters on the data.

We decided to use the following numerical parameters for filtering:

- number of stargazers for a repository as a measure of popularity,
- number of forks of the repository[2] as a measure of popularity,
- number of commits in the main branch of a repository as a measure of maturity,
- total size (in bytes) of code in chosen language as a measure of project scale.

To simplify development we restricted ourselves to using GitHub as project source. Since we believe that most research should be performed only on active projects, we also added two conditions that assert that: the project is `not archived` (archived projects are no longer developed) and last push was during previous year and a half. We also use one more condition, this one related to project maturity—since we wanted to examine only projects which have some background, it became necessary to filter out ones that were created only recently. We—arbitrarily—decided that a "mature" project should be available online for at least one full year, from the beginning to the end. As a result, in our study we selected 15 months (search was performed in March) to be the minimal lifetime of a project to be included in result data set. We also decided that it would be a reasonable assumption that companies provide industry-quality projects and therefore restricted our search for repositories to 30 biggest open source contributors amongst companies. This list of companies was created by [6] and we did not attempt to further validate it. We did manually extract organizations that belong to given companies—for example, both *amzn* and *aws* belong to Amazon. We also added two big open source organizations - Apache Software Foundation and Eclipse Software Foundation.

For all numerical parameters we decided to remove the first quartile from the data set. Quartile is auto-tuned by existing projects, so we believe it is sufficient to filter out the most irrelevant repositories.

[2] this also means that we rejected all forks and only left the main repository

A separate issue is the data actually provided in the data set—we firmly believe that no data that is easy to obtain should be included in a public data set for reasons other than study reproduction. Only fields that should be included in such a data set should be ones that either require lengthy computation, substantial expenses to obtain (for example proprietary tools or even manual inspection) or are likely to be gone or change location after some time. This means that a data set of classes should contain full paths to classes and their versions (commit SHA in Git) rather than their individual metrics, provided that classes are expected to be accessible during whole lifetime of a data set. On the other hand, number of forks, watchers or stargazers for a repository is likely to differ on another query, so it should be included in the data set.

There are several reasons why we believe that this is the right approach: first, such data sets are smaller and easier to maintain. Second, other researchers do not replicate data that could be wrong (for example due to a defect in metric calculation software). Third, it allows researchers to use arbitrary metrics. Of course, there are also downsides to this: data sets on proprietary data cannot be published, some data may not be accessible any more and researchers need to put more effort into gathering data. The last problem can be mitigated if data set-creating script contains also scripts that are used for metrics calculations.

In the next section we are going to answer the following research question:

RQ How efficient is our algorithm is finding industry-relevant Java projects?

To do that, we are going to create a data set for Java projects and manually assess the amount of them that can be classified as *industry-relevant projects*.

The performance metric that we will use to assess quality of created data set will be precision, defined as:

$$\frac{P_r}{P_a} \tag{1}$$

Where P_r is the number of industry-relevant projects in the data set and P_a is the number of all projects in the data set.

We believe that precision is by far the best metric for our use case. First, the research is conducted on open source software, which means that most industrial (i.e., by definition, industry-relevant) projects will not be retrieved. Second, we further restrict ourselves to GitHub, abandoning any potential industry-relevant repositories that are hosted on BitBucket, Gitlab, SourceForge, Savannah or any other server. Since we have no chance of accessing a lot of relevant repositories, the real recall will be low even if we retrieve all relevant repositories from GitHub.

We believe this is not an issue, because in industry we do not expect developers to be familiar with hundreds of projects. Since recall cannot be critical for real developers to learn concepts in software engineering, it also should not be critical for machine learning models. Consequently, we are only interested in two aspects: that the data set would represent what it is supposed to (whatever that means for a specific data set) and that precision of the data set is high (again: whatever that means for any specific case).

To answer our research question we manually inspect a data set of Java projects obtained using our algorithm. The data set contains 792 projects. What is necessary to reproduce or build upon our research is publicly available in the 0.3.0 version of `reproducer` R package accessible from CRAN [27], the official repository of R packages, as our goal is to promote reproducibility of research in software engineering [25] by supporting research papers by the related `R` package (e.g., see [26]). We do not include data acquisition and processing script in the paper, as it would greatly increase the volume of it without adding significant value.

We define industry-relevant projects as ones that fulfill all of the following requirements:

- has a project website with documentation or reasonable in-repository documentation,
- provides an installer, package in a package repository (e.g. Maven Central or npm.org) or detailed installation instructions,
- has a way of reporting defects or providing support (e.g. Google Groups, GitHub Issues, Gitter),
- is not a set of samples, exercises and example code.

We do not set any other requirements for a project to be considered industry-relevant. In particular, we put no constraints on project domain and we do not require any specific development techniques (e.g. usage of Continuous Integration or any specific build system). The rationale for such approach is that, while we do not exactly know what characteristics do industrial projects have when it comes to source code, they definitely are treated differently than proof-of-concepts and pet projects in the area of project and product management. Therefore, we are looking for projects that present a decent level of maturity and newcomer-friendliness. These are projects that have documentation, relatively simple way of trying out and some way of reporting defects. Having a support channel is an additional plus, but not a must.

Some of the projects are graded as half-industrial—for example ones that have poor documentation but are very simple projects (still important, so may be industry-relevant) or ones that have everything else set up, but repository does not mention any support.

4 Results

During research we encountered several interesting repositories: ones that turned out to be copies, ones that were discontinued, ones that are clones of each other. Each such case was clearly marked in the source data and can be inspected at any time.

Our data set contained a total of 792 repositories from 37 GitHub organizations, with a minimum of 1 repository (baidu, greenplum) and maximum of 334 (apache). 365 repositories came from companies, the biggest contributors being

Pivotal, Google, Amazon and Microsoft, while 427 came from biggest software foundations—Apache, Eclipse and Mozilla.

Entire data set with acquired industry-relevance values is published in 0.3.0 version of `reproducer` R package accessible from CRAN [27], the official repository of R packages.

4.1 RQ: How efficient is our algorithm is finding industry-relevant projects?

Out of 792 projects, 517 were assessed entirely industry-relevant and another 118 were assessed semi-relevant. Out of remaining 157 projects 52 were already obsolete—either due to pushing old commits or deprecated during the year— 61 were samples, examples, playgrounds, tests or other kind of non-stable code, and remaining 44 did not contain proper documentation, means of installation and support. Out of the 61 sample repositories, 33 contained at least one of words: ”sample”, ”example”, ”demo” in repository or organization name.

Precision calculated from numbers above is 0.65 for entirely industry-relevant projects and 0.80 for semi-relevant projects.

5 Discussion

The technique we provided lets researchers describe some basic features of a data set without constraining them to static values. While research done on different data sets will obviously yield slightly different results, those can be interpreted as a change in industry trends, which is an extremely important issue for software engineering.

Presented method achieves precision between 0.65 and 0.80 on Java projects, which means that it can be used for creating project data sets and as a baseline technique. This result can still be substantially improved, but it is a reasonable starting point for future research.

Interestingly, most industry-relevant projects come from Apache Foundation. While at first this seems surprising, many companies donate mature open source projects to open source foundations. For example, Apache Hive was donated by Facebook, and Kafka was donated by LinkedIn. Apache Foundation provides some infrastructure for projects which company-driven ones may lack—for example a public Jira instance and mailing lists. Additionally, both Apache and Eclipse foundations require projects to go through incubation phase, which may also improve overall rating of their projects.

We argue that even though precision of this technique may be lower than precision of simply picking a number of repositories with highest stargazer/fork count, it yields data set with much more variety.

Projects that were not analyzed in this study belong to two main groups:

1. projects developed by a company that is not one of top open-source committers,
2. projects too small or too niche to be included.

While we intentionally omitted group 2, group 1 is something we might want to consider including in future versions of the script. In particular, companies that develop only a single, big product—product like Neo4J, Nexus, OrientDB or ElasticSearch—are excluded from the data set, regardless of the fact that they are absolutely industrial-grade projects, sometimes top of their domain.

6 Threats to validity

Like every research, this one also has its drawbacks.

6.1 Construct validity

We claim that data set obtained with the technique described in this paper should be a reasonable choice for industry-relevant data set. However, it should not be understood as universally the best choice for any research on software engineering—even if it is a good starting point due to industry-relevance. Of course, since the assessment of industry-relevance is performed manually, there is always a risk of misclassification—to address this issue, we provide used data set, together with gathered values for manually assessed fields.

6.2 Internal validity

One may rightly argue that the boundary set on the first quartile is arbitrary the second or third one would be just as good. Perhaps some other percentile might do as well—while each of those approaches would be technically valid, some initial cutoff point had to be selected. We do envision further extending this research with verifying the best cutoff point.

The range of *active projects* is also something that is open for discussion—depending on the range the research is made on, half a year may be either too long or too short period. Obviously, other fixed parts of the query (such as cutoff creation date or cutting off only the first quartile) are also subject to further tuning and are more of an reasonable assumption than a fixed rule.

Some of the conditions we chose may filter out projects that are relevant—for example, *creation date* refers to project, and not repository. This means that if a project is migrated from a different platform recently, it will not be detected. Since GitHub is the biggest open source platform [36] and companies analyzed already do

have GitHub accounts, we believe that amount of projects that fall into this category is negligible.

Manual verification is always a point of failure, omission and mistake, and we probably did not manage to avoid them. However, we performed the process scrutinously and provide full replication package and verification data set, so other researchers can vet us.

As always with dedicated software, there is a risk that the application we wrote contains undiscovered defects. While we did our best to test it and peer-reviewed it, possibility of defects cannot be diminished. For example, one of activity filters we set up—last push to the repository after beginning of 2018—did not work as intended. While it did find relevant pushes from GitHub perspective, the actual commit could have been implemented months or even years earlier. Unfortunately, that is an intrinsic problem which cannot be rectified by using date of last commit—Git commits use local system clock, thus are also not entirely reliable.

6.3 External validity

Obtained data set contains only projects in which Java is the dominant language. However, there are no fixed constraints on language in the method—as long as GitHub API provides all the fields needed and language is recognized (and virtually all are recognized), projects from any language can be fetched.

While we do rely on open source projects, we also constrain ourselves to projects hosted or mirrored on GitHub. While GitHub is the biggest repository of open source projects [36], there are also other significant players like SourceForge, GitLab and BitBucket.

The study utilizes lists of repositories from the companies that have most open source activity. However, this list will change in time, and its size is not set in stone. To acquire the freshest version of this list, the research done in [6] must be redone. In particular, we omit relatively small companies with substantial industry impact, such as JFrog or Neo4J. Our study used Java as language for validation—while we believe that is a reasonable choice, that may unintentionally mask a bias problem—companies that invest most in open source do not necessarily invest most in open source projects matching required profiles. This should not be a big problem for Java—ranked in the TOP3 TIOBE index for many years now [4], but will become a problem once we start creating data sets for less popular technologies, for example Rust or Ruby.

7 Future work

It is still necessary to utilize the technique in real research to prove its usefulness. We encourage all researchers to not only use the example data set but also provided script to create their own data sets.

A huge chunk of work that needs to be done is to provide a reasonable way to provide access to evolving data sets. For source code this function is done by version control systems and binary artifact repositories. Similar techniques are also used in some databases and document stores. However, for data sets we would need to focus on the metadata of the data set. An open repository for evolving data sets is something still to be published.

As for the project data set creation technique we presented here—initial assessment of industry-relevance was done and successful, but further research is still needed. In particular, we will verify whether the industry-relevance property holds for other programming languages and whether precision alone is indeed sufficient metric to make data set usable. While this was our initial assumption, it is by no means obvious and requires further verification.

As for the provided script itself, its big drawback is a hard-coded list of used groups and users. While these groups and users were taken from previous research [6], open-source involvement of companies changes with time, and we should take this into account as well when creating a project data set. List of groups and users from which repositories are analyzed should also be dynamic and decided in run time, or it could be another evolving data set.

8 Conclusions

We firmly believe that the model of evolving data sets presented in this paper is the way to shape the future of software engineering research. Providing this kind of facility would allow us, researchers, to investigate not only state of software development at given point of time, but also its evolution on large scale, in many dimensions. This already happens for source code with version control, it is the right time for it to enter data science. This kind of data sets would have to co-exist with traditional, static data sets—still required for study reproduction.

We also presented a technique for obtaining industry-relevant data sets from GitHub open source repository. We focused only on the source code repository, ignoring anything else related to the project (such as issue tracker, mailing list or support forums). We carefully evaluated returned projects and manually assessed precision of algorithm was between 0.65 and 0.80 for Java repositories, which means that between 0.65 and 0.80 of returned projects are industry-relevant.

Acknowledgements This work has been conducted as a part of research and development project POIR.01.01.01-00-0792/16 supported by the National Centre for Research and Development (NCBiR). We would like to thank Tomasz Korzeniowski and Marek Skrajnowski from *code quest*

sp. z o.o. for all of the comments and feedback from the real-world software engineering environment.

References

1. Awesome empirical software engineering resources. https://github.com/dspinellis/awesome-msr. Accessed: 2019-03-31
2. dat:// — a peer-to-peer protocol. https://datproject.org/. Accessed: 2019-04-23
3. Open-source version control system for machine learning projects. https://dvc.org/. Accessed: 2019-04-23
4. Tiobe index. https://www.tiobe.com/tiobe-index/. Accessed: 2019-04-24
5. Arisholm, E., Gallis, H., Dybå, T., Sjøberg, D.I.K.: Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise. IEEE Transactions on Software Engineering **33**(2), 65–86 (2007)
6. Asay, M.: Who really contributes to open source (2018). URL https://www.infoworld.com/article/3253948/who-really-contributes-to-open-source.html. [Online; posted 7-February-2018; Accessed 23-April-2019]
7. Badashian, A.S., Esteki, A., Gholipour, A., Hindle, A., Stroulia, E.: Involvement, contribution and influence in github and stack overflow. In: Proceedings of 24th Annual International Conference on Computer Science and Software Engineering, CASCON '14, pp. 19–33. IBM Corp., Riverton, NJ, USA (2014). URL http://dl.acm.org/citation.cfm?id=2735522.2735527
8. Bird, C., Pattison, D., D'Souza, R., Filkov, V., Devanbu, P.: Latent social structure in open source projects. In: Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, SIGSOFT '08/FSE-16, pp. 24–35. ACM, New York, NY, USA (2008). DOI 10.1145/1453101.1453107. URL http://doi.acm.org/10.1145/1453101.1453107
9. Cosentino, V., Izquierdo, J.L.C., Cabot, J.: Findings from github: Methods, datasets and limitations. In: 2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR), pp. 137–141 (2016). DOI 10.1109/MSR.2016.023
10. Dybå, T., Dingsøyr, T.: Empirical studies of agile software development: A systematic review. Information and Software Technology **50**(9-10), 833–859 (2008)
11. Falessi, D., Smith, W., Serebrenik, A.: Stress: A semi-automated, fully replicable approach for project selection. In: 2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 151–156 (2017)
12. Filó, T.G., Bigonha, M.A., Ferreira, K.A.: Statistical dataset on software metrics in object-oriented systems. SIGSOFT Softw. Eng. Notes **39**(5), 1–6 (2014). DOI 10.1145/2659118.2659130. URL http://doi.acm.org/10.1145/2659118.2659130
13. Gebru, T., Morgenstern, J.H., Vecchione, B., Vaughan, J.W., Wallach, H.M., Daumé, H., Crawford, K.: Datasheets for datasets. CoRR **abs/1803.09010** (2018)
14. Gousios, G.: The ghtorent dataset and tool suite. In: Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13, pp. 233–236. IEEE Press, Piscataway, NJ, USA (2013). URL http://dl.acm.org/citation.cfm?id=2487085.2487132
15. Guzman, E., Azócar, D., Li, Y.: Sentiment analysis of commit comments in github: An empirical study. In: Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014, pp. 352–355. ACM, New York, NY, USA (2014). DOI 10.1145/2597073.2597118. URL http://doi.acm.org/10.1145/2597073.2597118
16. Habayeb, M., Miranskyy, A., Murtaza, S.S., Buchanan, L., Bener, A.: The firefox temporal defect dataset. In: Proceedings of the 12th Working Conference on Mining Software Repositories, MSR '15, pp. 498–501. IEEE Press, Piscataway, NJ, USA (2015). URL http://dl.acm.org/citation.cfm?id=2820518.2820597

17. Jureczko, M., Madeyski, L.: Towards identifying software project clusters with regard to defect prediction. In: PROMISE'2010: Proceedings of the 6th International Conference on Predictive Models in Software Engineering, pp. 9:1–9:10. ACM (2010). DOI 10.1145/1868328.1868342
18. Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D.M., Damian, D.: The promises and perils of mining github. In: Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014, pp. 92–101. ACM, New York, NY, USA (2014). DOI 10.1145/2597073.2597074. URL http://doi.acm.org/10.1145/2597073.2597074
19. Lamastra, C.R.: Software innovativeness. a comparison between proprietary and free/open source solutions offered by italian smes. R&D Management **39**(2), 153–169 (2009). DOI 10.1111/j.1467-9310.2009.00547.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-9310.2009.00547.x
20. Lamkanfi, A., Pérez, J., Demeyer, S.: The eclipse and mozilla defect tracking dataset: A genuine dataset for mining bug information. In: 2013 10th Working Conference on Mining Software Repositories (MSR), pp. 203–206 (2013). DOI 10.1109/MSR.2013.6624028
21. MacCormack, A., Rusnak, J., Baldwin, C.Y.: Exploring the structure of complex software designs: An empirical study of open source and proprietary code. Management Science **52**(7), 1015–1030 (2006). DOI 10.1287/mnsc.1060.0552. URL https://doi.org/10.1287/mnsc.1060.0552
22. Madeyski, L.: Test-Driven Development: An Empirical Evaluation of Agile Practice. Springer, (Heidelberg, London, New York) (2010). DOI 10.1007/978-3-642-04288-1
23. Madeyski, L.: Training data preparation method. Tech. rep., *code quest* (research project NCBiR POIR.01.01.01-00-0792/16) (2019)
24. Madeyski, L., Kawalerowicz, M.: Continuous Test-Driven Development: A Preliminary Empirical Evaluation using Agile Experimentation in Industrial Settings. In: Towards a Synergistic Combination of Research and Practice in Software Engineering, *Studies in Computational Intelligence*, vol. 733, pp. 105–118. Springer (2018). DOI {10.1007/978-3-319-65208-5\textunderscore8}
25. Madeyski, L., Kitchenham, B.: Would wider adoption of reproducible research be beneficial for empirical software engineering research? Journal of Intelligent & Fuzzy Systems **32**(2), 1509–1521 (2017). DOI 10.3233/JIFS-169146
26. Madeyski, L., Kitchenham, B.: Effect Sizes and their Variance for AB/BA Crossover Design Studies. Empirical Software Engineering **23**(4), 1982–2017 (2018). DOI 10.1007/s10664-017-9574-5
27. Madeyski, L., Kitchenham, B.: reproducer: Reproduce Statistical Analyses and Meta-Analyses (2019). URL http://madeyski.e-informatyka.pl/reproducible-research/. R package version (http://CRAN.R-project.org/package=reproducer)
28. Munaiah, N., Kroh, S., Cabrey, C., Nagappan, M.: Curating github for engineered software projects. Empirical Software Engineering **22**(6), 3219–3253 (2017)
29. Ohira, M., Kashiwa, Y., Yamatani, Y., Yoshiyuki, H., Maeda, Y., Limsettho, N., Fujino, K., Hata, H., Ihara, A., Matsumoto, K.: A dataset of high impact bugs: Manually-classified issue reports. In: 2015 IEEE/ACM 12th Working Conference on Mining Software Repositories, pp. 518–521 (2015). DOI 10.1109/MSR.2015.78
30. Ortu, M., Destefanis, G., Adams, B., Murgia, A., Marchesi, M., Tonelli, R.: The jira repository dataset: Understanding social aspects of software development. In: Proceedings of the 11th International Conference on Predictive Models and Data Analytics in Software Engineering, PROMISE '15, pp. 1:1–1:4. ACM, New York, NY, USA (2015). DOI 10.1145/2810146.2810147. URL http://doi.acm.org/10.1145/2810146.2810147
31. Pletea, D., Vasilescu, B., Serebrenik, A.: Security and emotion: Sentiment analysis of security discussions on github. In: Proceedings of the 11th Working Conference on Mining Software Repositories, MSR 2014, pp. 348–351. ACM, New York, NY, USA (2014). DOI 10.1145/2597073.2597117. URL http://doi.acm.org/10.1145/2597073.2597117

32. Pruett, J., Choi, N.: A comparison between select open source and proprietary integrated library systems. Library Hi Tech **31**(3), 435–454 (2013). DOI 10.1108/LHT-01-2013-0003. URL https://doi.org/10.1108/LHT-01-2013-0003
33. Raemaekers, S., van Deursen, A., Visser, J.: The maven repository dataset of metrics, changes, and dependencies. In: 2013 10th Working Conference on Mining Software Repositories (MSR), pp. 221–224 (2013). DOI 10.1109/MSR.2013.6624031
34. Rafique, Y., Misic, V.B.: The effects of test-driven development on external quality and productivity: A meta-analysis. IEEE Trans. Softw. Eng. **39**(6), 835–856 (2013)
35. Sawant, A.A., Bacchelli, A.: A dataset for api usage. In: Proceedings of the 12th Working Conference on Mining Software Repositories, MSR '15, pp. 506–509. IEEE Press, Piscataway, NJ, USA (2015). URL http://dl.acm.org/citation.cfm?id=2820518.2820599
36. Sharma, A., Thung, F., Kochhar, P.S., Sulistya, A., Lo, D.: Cataloging github repositories. In: Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering, EASE'17, pp. 314–319. ACM, New York, NY, USA (2017)
37. Smith, T.M., McCartney, R., Gokhale, S.S., Kaczmarczyk, L.C.: Selecting open source software projects to teach software engineering. In: Proceedings of the 45th ACM Technical Symposium on Computer Science Education, SIGCSE '14, pp. 397–402. ACM, New York, NY, USA (2014)
38. Tamburri, D.A., Palomba, F., Serebrenik, A., Zaidman, A.: Discovering community patterns in open-source: a systematic approach and its evaluation. Empirical Software Engineering (2018)
39. Tempero, E., Anslow, C., Dietrich, J., Han, T., Li, J., Lumpe, M., Melton, H., Noble, J.: The qualitas corpus: A curated collection of java code for empirical studies. In: 2010 Asia Pacific Software Engineering Conference, pp. 336–345 (2010). DOI 10.1109/APSEC.2010.46
40. Vasudevan, A.R., Harshini, E., Selvakumar, S.: Ssenet-2011: A network intrusion detection system dataset and its comparison with kdd cup 99 dataset. In: 2011 Second Asian Himalayas International Conference on Internet (AH-ICI), pp. 1–5 (2011). DOI 10.1109/AHICI.2011.6113948