Electronic Communications of the EASST
Volume 65 (2014)

Proceedings of the International Workshop on Software Quality and Maintainability (SQM 2014)

An Initial Quality Analysis of the Ohloh Software Evolution Data

Magiel Bruntink

15 pages

Guest Editors: Lodewijk Bergmans, Tom Mens, Steven Raemaekers
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

An Initial Quality Analysis of the Ohloh Software Evolution Data

Magiel Bruntink[1]

[1] M.Bruntink@uva.nl
System and Network Engineering research group, Faculty of Science, University of Amsterdam

Abstract:

Large public data sets on software evolution promise great value to both researchers and practitioners, in particular for software (development) analytics. To realise this value, the data quality of such data sets needs to be studied and improved. Despite these data sets being of a secondary nature, i.e., they were not collected by the people using them, data quality is often taken for granted, casting doubt on conclusions drawn from those data. This paper reports on an intial investigation of the quality of the software evolution data available on Ohloh, and further describes steps taken to cleanse the data set. Our goal is that other researchers, practitioners, and parties responsible for data sets such as Ohloh, use the outcomes of the validation and cleansing steps to improve quality of data sets in the public domain.

Keywords: Software evolution, Data quality, Open source software

1 Introduction

In recent years, the software evolution research community has gained access to a growing number of large public data sources on (open source) software projects. These data sources are now the subject of many research projects, as witnessed by papers at several international conferences such as MSR, CSMR, and WCRE. Furthermore, new fields of study and application are emerging around the study of these data sources, such as software development analytics [MZ12].

An overarching problem with software evolution data sources is the fact that many data are of a *secondary* nature, that is, they were not collected by the same party that wishes to study them. For instance, the PROMISE data set [MCH+12] contains data on defects observed on software at various organisations (such as NASA), but researchers external to those organisations are using such data sets in their studies. Instead of dealing with secondary data, one could obtain source artifacts (e.g., source code, version control records, mailing lists) and perform one's own data collection and processing. An example tool to support this approach is GHTorrent [GS12], which basically allows a researcher to obtain a mirror of projects on GitHub. Given that software projects number in the order of millions, and given the great diversity in programming languages and other factors [NZB13][ZMZ+13], such an approach will not be feasible or desirable for every software evolution researcher. Some researchers will therefore have to deal with the secondary nature of their data by, for instance, installing more explicit data quality controls [She11].

This paper reports on our study of Ohloh [Bla13b], which is best described as a large-scale online index and analytics platform for open source projects. At the time of writing about 600,000 projects are indexed on Ohloh. Each indexed project on Ohloh has its own page with project

meta-data, coupled with numerical data on its evolution. Examples of data are monthly numbers of lines of code, churn, number of commits, contributors, and so on. These data are the result of analyses done by Ohloh (hence, secondary data) based on a project's source code and version control records. The data can be accessed on the Ohloh website through pre-defined visualisations, or by using the REST API. The latter method offers access to more fine grained data. The data set used as the starting point in this paper was introduced in our earlier work [Bru13b].

For software evolution researchers and practitioners in software analytics, an easily accessible data source such as Ohloh would be of great value. Other research projects are indeed making use of Ohloh, such as Nagappan et al. [NZB13], Deshpande et al. [San], Arafat et al. [AR09], and Sands [DR08]. However, since from the viewpoint of external researchers, the data on Ohloh is secondary in nature, a necessary first step is to analyse the data from the perspective of quality. Our aim is to raise attention to this particular issue of data quality, and obtain a cleansed data set that is useable to other researchers.

In this paper we make the following contributions:

- First, we follow the approach outlined by Shepperd et al. in [SSSM13] by analysing the data for missing, implausible, and inconsistent values (Section 3). This approach consists of applying internal sanity checks to identify possible quality issues. In the future this approach needs to be complemented by external quality checks that compare to other data sets.

- Second, we use our findings to cleanse the data set of cases that are problematic, and report on the resulting cleansed data set (Section 4).

The paper is structured as follows. Section 2 first covers related work, Section 3 then describes the steps taken to collect and validate the data. Section 4 reports on cleansing the data set based on the validation results. Finally, Section 5 concludes the paper.

All data sets used in our study, and all software developed to handle the data, are available online at our GitHub project OhlohAnalytics project [Bru13a]. Replication details can be found in Appendix A.

2 Related work

Software evolution data sets. The (quantitative) study of (open-source) software repositories has been going on for quite some time, leading to a rich body of literature. Surveys of the field have been provided by Kagdi et al. in 2007 [KCM07] and Hassan in 2008 [Has08]. A recent overview (2011) of the OSS ecosystem is being provided by Androutsellis-Theotokis et al. in [ASKG11]. Examples of large software engineering data sets that are publicly available are GHTorrent [GS12], PROMISE [MCH+12], FLOSSmetrics [GRD10], FLOSSmole [HCC06], Qualitas Corpus [TAD+10], and Ohloh [Bla13b], but even more are available. A recent overview of public data sets is given by Rodriguez et al. in [RHH12].

Ohloh. Other researchers are also studying the data offered by Ohloh. Recently, Nagappan et al. [NZB13] used the Ohloh data set for defining a method to quantify the representativeness of empirical studies (on OSS projects). In 2008, Deshpande and Riehle reported on an investigation

into the total size and growth of the OSS ecosystem using Ohloh [DR08]. Also using Ohloh, Arafat and Riehle reported on the distribution of commit sizes [AR09].

Data quality in software engineering. Recent work that concerns data quality of software engineering data sets (the NASA software defect data in this case) has been done by Shepperd et al. [SSSM13]. In this paper we follow the categories of quality checks (missing, implausible, and inconsistent values) described in Shepperd's work. He also calls to action on the topic of data quality in software engineering in general [She11]. Liebchen (a student of Shepperd's) wrote a dissertation on the topic of data quality in software engineering, in particular from the perspective of identifying and handling noise [Lie10]. Rodriguez et al. discuss quality issues in public software data sets, in particular from the perspective of machine learning [RHH12].

Software analytics. Software analytics is a term recently introduced by Zhang et al. [ZDL+11] (among others) to label research aimed at supporting decision making in software [MZ]. Our work in this paper can be seen as supporting the field of software analytics. Work that is closely related to ours has been done by Herraiz. He studied the statistical properties of software evolution data available on SourceForge and the FreeBSD package base [HT08]. We see our work as a follow-up in terms of scope and diversity, since by studying Ohloh, we use a larger and more diverse data source (which does not primarily focus on C). Zhang et al. [ZMZ+13] recently studied the influence of context variables such as programming language, project age, and so on, on software evolution metrics. In our exploration of the Ohloh data, we use programming language as the main context variable to group the data. Zhang et al. identify programming language as the context variable influencing the most metrics, supporting our choice. Nevertheless, other context variables investigated by them are also influential and should be investigated further in the future.

3 Data collection and quality analysis

This section describes the three steps we performed in order to investigate the quality of our Ohloh data set: collection, validation and cleansing. For each step, we present the resulting numbers on the data. Appendix A describes how the research results can be obtained and replicated.

3.1 Data collection

The data set was collected in July 2013 from Ohloh [Bla13b], an open-source software index and analytics platform. Generally these data consist of a monthly aggregate resulting from analysis (done by Ohloh) of the source code and the version control system(s) of an OSS project. In total, data were collected for 12,360 OSS projects, resulting in a grand total of 828,312 project months. The projects have been selected according to Ohloh's measure of project popularity in descending order (number of users of a project as declared by users of Ohloh). The cut-off point at 12,360 was imposed on our data collection process due to time constraints and reaching a file count limit in our programming environment (Eclipse). Future data collection could collect more projects given some changes in the collection tools.

The data collection was initiated by obtaining a list of project names from Ohloh's REST API.

Feature	Description
Identification	
Project name	The name of the project for identification purposes.
Year	The calendar year that the data belong to.
Month	The calendar month that the data belong to.
Core features	
Lines Of Code (LOC): Total, Added, Deleted	Lines of code total at the end of Month, based on all source text configured for the project in any programming language. Added and Deleted counts are derived from commits in Month.
Blanks: Total, Added, Deleted	As LOC, but for lines containing only blanks (whitespace).
Comments: Total, Added, Deleted	As LOC, but for lines containing only comments.
Comment Ratio	The proportion of comment lines from the total non-blank lines (code + comments) in Month.
Commits: monthly and Cumulative	The number of commits performed in Month, and the cumulative number of commits since the start of the project.
Contributors	The number of people that made at least 1 commit in Month.
Man Months	An estimate of the effort applied since the start of the project. This estimate is the running total of the monthly contributors number.
Meta data	
Main programming language	The most used programming language for the project to which this Month belongs, as measured using LOC in the month of data collection, i.e. July 2013. In other words, all the months of a project are assigned the same main programming language, being the last main programming language Ohloh identified for that project.
Last update time	The date at which a project was last analysed by Ohloh.

Table 1: Definition of the features of the data set collected from Ohloh, as described in Section 3.1.

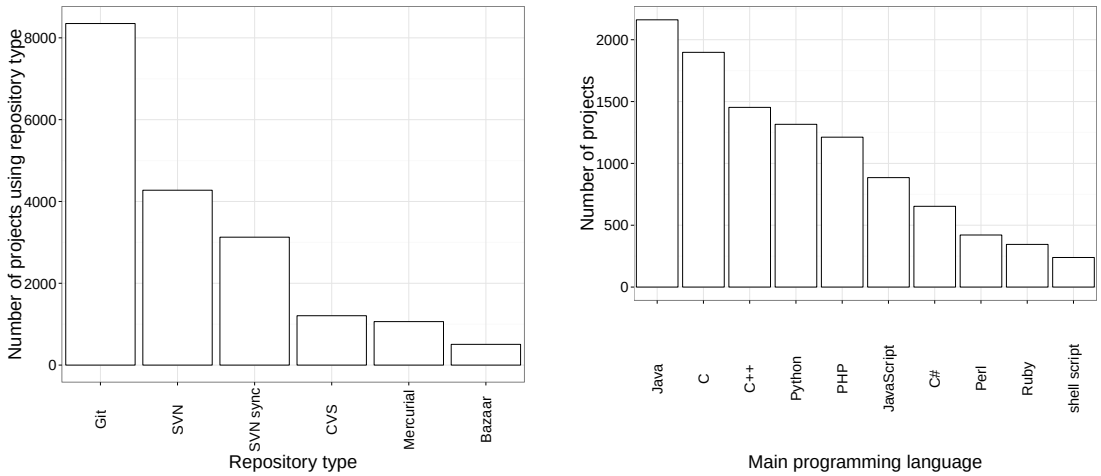

Figure 1: Histograms of the projects in the data set before cleansing. The left-hand side histogram shows the usage of the 6 version control systems supported by Ohloh. The right-hand side histogram shows the the 10 most frequent (out of 76) main programming languages.

Subsequently, per project 4 queries are done to obtain 4 separate XML files that contain data. The XML files from Ohloh contain the following data:

- **MetaData.xml**: Includes a project description, site URLs, creation and update dates, user count, user supplied tags (keywords), programming language use, among others.
- **Enlistments.xml**: The projects's configuration of version control systems that should be included in Ohloh's code analysis. In July 2013, Ohloh supported the following version control systems: Git, Mercurial, Bazaar, Subversion, and CVS. All types of version control systems occur in the data set.
- **SizeFacts.xml**: Monthly statistics on size of the projects source code: lines of code (source text excluding comments and white space), lines of comments, blank lines. Ohloh includes all of a project's source code in calculating its statistics: code in all programming languages recognised by Ohloh's code counting tool Ohcount. This tool is available as an open source project [Bla13a]. Normally, every month of data is available since the project started using the enlisted version control system(s).
- **ActivityFacts.xml**: Monthly statistics on changes to the project's source code: added or removed lines of code, comments and blanks, number of commits to the version control systems used by the project, and the number of contributors (users performing the commits). Normally, every month of data is available since the project started using the enlisted version control system.

When talking about the data set in the remainder of this paper we will use the following (common) terminology:

Projects	
Projects collected from Ohloh API	12,360
Projects with an improper version control configuration	1,377
Projects with non-consecutive monthly data	0
Projects without any data	29
Projects remaining	10,954
Values in core features (see Table 1)	
Total number of values	11,000,640
Missing values	115,078
Implausible values	2,694
Cases (project-months)	
Total number of cases	785,760
Cases with missing values	20,375
Cases with implausible values	1,213
Cases with inconsistent features	452,730

Table 2: Overview of the data validation steps described in Section 3.2.

- **Feature.** A dimension of the data set, also typically called a variable or column. A case (see below) either holds a value for a feature in the data set, or the special identifier 'NA', which indicates a missing value.
- **Case.** A data tuple, or row, of the data set. In our data set each case is uniquely identified by its combination of its project name, year and month features. A case thus represents a calendar month of development for a project in a calendar year.
- **Value.** An individual number or other type of value that a case contains for a feature.

The XML files obtained from Ohloh are processed resulting in the data features defined in Table 1. Note that the size and activity features have been joined together (on the primary data key formed by the Project name, Year and Month). If some features are missing for a case, special identifiers ('NA') are inserted instead. If all size and activity facts are missing, the project is reported missing altogether (see the data validation process in Section 3.2. All processing and storage of data is handled by software developed using the meta-programming language Rascal [KSV09] and the statistics environment R [R D08].

3.2 Data validation

Before actual validation started, the cases per project were truncated to the cases that represent complete months. This was done by comparing the 'Last update time' to the 'Year' and 'Month' features of each case. When 'Last update time' falls within a case (or theoretically, lies before a case), that case does not represent a completely analysed month yet, and the case is excluded from the data set.

Several steps of data validation were performed to study the quality of the data. Table 2 gives a numerical overview of the outcomes. Here we describe the validation steps we performed, inspired by the data quality analysis done by Shepperd et al. in [SSSM13], and further guided by known issues of the Ohloh data set. Instead of cleansing the data set of problematic data once we identify them, we only mark the data as such. This allows us to first report on the data quality

Feature	Missing values	Implausible values
LOC	18,060	639
Blanks	18,060	699
Comments	18,060	942
Comment ratio	19,746	414
LOC Added	629	0
LOC Deleted	629	0
Comments Added	629	0
Comments Deleted	629	0
Blanks Added	629	0
Blanks Deleted	629	0
Commits	629	0
Cumulative commits	18,060	0
Contributors	629	0
Man months	18,060	0

Table 3: Outcomes of the missing and implausible checks described in Section 3.2. The numbers presented here are for the 10,954 projects that have a proper version control configuration and a non-empty set of consecutive monthly data.

Inconsistent features check (shown as logical formula)				Failing cases	Uncheckable cases
LOC $\neq 0$	$\vee$	Blanks $\neq 0$	$\vee$ Comments $\neq 0$	1,645	0
LOC	$=$	LOC (prev. month)	$+$ Added $-$ Deleted	87,984	29,503
Blanks	$=$	Blanks (prev. month)	$+$ Added $-$ Deleted	82,525	29,503
Comments	$=$	Comments (prev. month)	$+$ Added $-$ Deleted	78,566	29,503
Commits $= 0$	$\wedge$	(LOC Added $\neq 0$	$\vee$ LOC Deleted $\neq 0$)	0	629
Commits $= 0$	$\wedge$	(Comments Added $\neq 0$	$\vee$ Comments Deleted $\neq 0$)	0	629
Commits $= 0$	$\wedge$	(Blanks Added $\neq 0$	$\vee$ Blanks Deleted $\neq 0$)	0	629
Comment Ratio	$=$	Comments	/ (Comments + LOC)	0	19,746
Cum. commits	$=$	Cum. commits (prev. month)	$+$ Commits	92,866	29,503
Man months	$=$	Man months (prev. month)	$+$ Contributors	451,304	18,689

Table 4: Outcomes of the consistency checks described in Section 3.2. The numbers presented here are for the 10,954 projects that have a proper version control configuration and a non-empty set of consecutive monthly data. The Comment Ratio check first rounds the numbers to 10 decimal places.

issues of the raw data set, and then later provide a cleansed version. All versions of the data set, i.e. the raw version, the version with marked data, and the cleansed version are available at our GitHub project *OhlohAnalytics*.

Problematic SVN configurations. The first step consists of checking for a known issue with the Ohloh data: Projects that use the SVN version control system[1], while instructed by Ohloh to only configure those code directories that represent active and unique developments, sometimes list the projects's top-level SVN directory. The consequence is that all project branches and tagged versions are also included in Ohloh's analysis, in turn leading to many potential duplications and inflated metrics. We use a heuristic approach to check the SVN enlistments (Ohloh's term for code locations) of a project against a list of patterns that are 'safe' to use. This list is provided in Appendix B.

Missing data. Second, data missing from the data set are identified. This problem can occur at multiple levels. On the project-level, some projects can be missing all size or activity data despite being accessible through the Ohloh API. These are easily identified by our processing code because the XML files are missing or empty. On the case-level this issue is more subtle. If some features are missing for a case, the processing code will have inserted 'NA' identifiers instead of normal data values. More troublesome is the issue that entire cases could be missing for a project. We utilise the following definition: a project's data set is complete if it contains consecutive cases in time from its earliest case up to its last case. In other words, no months are allowed to be missing in between a project's appearance and disappearance. The weakness of this definition is that we do not validate if the project has been active before (or after) it appeared in the Ohloh data. This could only be done accurately by including data from other data sources on the same projects.

Implausible data. Third, we check whether any of the individual data values are implausible. Since most of the values in our data set are counts (except the Comment Ratio, which is a ratio between two counts and hence also expected to be positive), an obviously implausible value would be a negative number. Other types of implausible values have not been checked for at this point. Shepperd et al. [SSSM13] suggest also checking whether counts have non-integer values, and checking whether LOC is 0. In our case non-integer values for count features would be flagged by the type systems of both Rascal and R. There are no such occurrences to report. We also check for 0 values for LOC, but combined with 0 values for both Blanks and Comments. This check is a bit of a special case since it does not check an individual value. We choose to list it as a cross-feature consistency check (discussed next).

Inconsistent data. Finally, a number of cross-feature consistency checks have been applied:

- Cases where the LOC, Blanks, and Comments features are all 0. These are cases where a project may still exist on Ohloh, but no actual source text seems to be present (anymore). Either the project has been terminated, or the data analysis done by Ohloh is erroneous.

- Cases where this month's values for LOC, Blanks, and Comments do not equal the sum of previous month's values and the monthly churn as measured by the respective Added feature minus the Deleted feature.

[1] Indicated by either 'SvnRepository' or 'SvnSyncRepository' in Ohloh's data on the enlistments –or source code locations– supplied for a project.

- Cases where there are no commits in a month, but still changes are visible in LOC, Comments or Blanks.
- Cases where the Comment Ratio value does not equal the proportion of Comments from the sum LOC and Comments (all numbers rounded to 10 decimal places).
- Cases where this month's cumulative value for Commits does not equal the sum of previous month's value and this month's Commits value.
- Cases where this month's Man Months value does not equal the sum of previous month's value and this month's Contributors value.

The results of the data validation steps are reported in Tables 2, 3, and 4. The numbers in the tables are for the 10,954 projects that have a proper version control configuration and a non-empty set of consecutive monthly data. Note that cases may appear more than once among the missing and implausible value counts and the inconsistent features counts.

In some cases inconsistent feature checks could not be performed, as reported under 'Uncheckable cases.' This is due to either missing values (e.g., if LOC is missing, the consistency checks for LOC or the Comment Ratio cannot be done), or the presence of 0 values (e.g., where the Comment Ratio would be infinite). Where the consistency checks include a reference to a previous month's value, another reason for uncheckable cases exists. The very first month of each project has no previous month, by definition. Since 'Projects with non-consecutive monthly data' is 0, this means the number of uncheckable cases due to this reason is equal to the number of projects (10,954).

3.3 Data quality observations

The validation outcomes in Tables 2, 3, and 4, show that the Ohloh data set is not without flaws. Some of these problems were known to exist beforehand, however without a quantitative estimate of their impact. First, improper SVN configuration for some projects was a problem raised earlier on Ohloh's user forums. From our data we learn that 11% of the projects could have this problem, which we therefore excluded from our collected data. Since the number of projects involved forced us to use a heuristic approach, this problem calls for a fix at the source. Automatic verification of user submitted version control configurations, e.g., detection of duplicated directory trees, would probably prevent many instances of this problem.

Second, the occurrence of 2,694 (0.02%) negative values for features that are counts (i.e. LOC, Blanks, Comments and the derivative feature Comment ratio), is rather surprising. A closer look reveals that these implausible values occur in 65 projects, in 12 different programming languages, and ranging in time from 1999 to 2013. It thus appears that the cause of this problem has not been repaired yet.

Looking at missing values (1%), we see that in particular values in the size features (i.e. LOC, Blanks, Comments, and Comment ratio) are missing often, together with the features Cumulative commits and Man months. Missing values are problematic in their own right (see [Moc08] for an overview) due to coping strategies possibly introducing selection biases, and furthermore, missing values result in cases where consistency checks could not be applied. Another reason

consistency checks sometimes cannot be applied is that data is missing by definition, i.e. in the first month of a project, there is no data on the previous month. Other missing values are likely due to faults in automatic analyses, like failing code collection or analysis.

Moving on to consistency issues, we first observe that the Man months and Contributors features seem particularly problematic, as more than 50% of the cases hold an inconsistent value. We therefore decided to drop this feature entirely. Furthermore, we observe the high frequency of cases (around 10%) that fail the consistency checks for the deltas on LOC, Comments and Blanks (from SizeFacts.xml) compared to the difference between Added and Deleted features (from ActivityFacts.xml). At his point we can only speculate on the cause(s) of these failed checks. It is however clear that implementing consistency checking in the Ohloh analyses would identify such issues earlier if they appear again in the future.

4 Data cleansing

The data validation step resulted in an annotated version of the data set, where boolean features were added to indicate whether a quality problem exists for a each case. Given an annotated data set like this, cleansing the data is a straightforward process. The strategy of cleansing applied here is described by Mockus in [Moc08] casewise deletion, and as a "quick-fix technique that may yield biased or inconclusive results" (abridged). We therefore consider this a speculative step that requires further investigation from a statistical perspective.

The following steps are applied:

1. **Removing features.** Inspecting the results in Table 4 it is clear that the Man Months feature has a problem, as for 57% of the cases the values are not consistent. We decided to drop the Man Months feature from the data set.

2. **Deleting cases.** Cases that were inconsistent because of the Man Months feature have not been deleted (for that reason), as we decided first to remove the Man Months feature from the data set entirely. After removing features, cases that have missing values, implausible values, or inconsistent features are deleted.

4.1 The cleansed data set

Cleansing has resulted in a data set that holds 85% of the original cases, as is shown by Table 5. Note that due to the deletion of cases during the cleansing process, 4,393 projects now no longer have consecutive monthly data. 143 projects disappeared completely due to the removal of all their cases.

Figures 2 and 3 show the impact of cleansing on the number of projects per version control system, and main programming languages, respectively. In particular projects using SVN (and SVN sync) repository types have been removed from the data set. In the validation process we found that for 1,377 (see Table 2) projects the SVN configuration could not be trusted to exclude branches and tagged versions, potentially leading to double counts.

Figure 3 shows that cleansing removed projects for all of the 10 most frequent main programming languages. It does not appear that a particular language is more prone to quality issues, but more analysis would be required to confirm this.

Projects		**% of initial**
Projects in the cleaned data set	10,811	87%
Projects with non-consecutive monthly data	4,393	-

Cases and values		**% of initial**
Number of cases after cleansing	671,570	85%
Number of values after cleansing	8,730,410	79%

Table 5: The data set after the cleansing process, as described in Section 4. Due to the deletion of cases during the cleansing process, 4,393 projects no longer have consecutive monthly data.

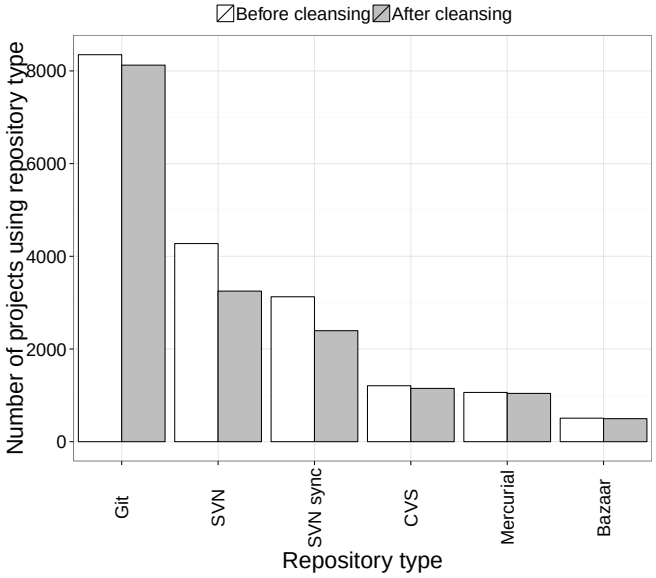

Figure 2: Histogram of the usage of the 6 version control systems supported by Ohloh by projects in our data set, before cleansing (left-hand side bars) and after cleansing (right-hand side bars).

Mockus [Moc08] recommends utilising other cleansing techniques (than casewise deleting) when more than 10% of cases are suffering from missing data. In our data set, we are dealing not only with missing data, but also data that is suspect from a consistency point of view, or clearly wrong (e.g., negative counts). Our foremost goal is the improvement of this data set, therefore our findings have been shared with the Ohloh development team. Most data quality issues reported here are probably caused by flaws in automatic analyses, calling for a fix at the source instead of more elaborate cleansing techniques. However, in future work we also aim to study the use of other cleansing techniques on software evolution data sets in general.

5 Conclusion

To conclude, we summarise the two main contributions of the paper and give suggestions for future work. The paper made the following contributions:

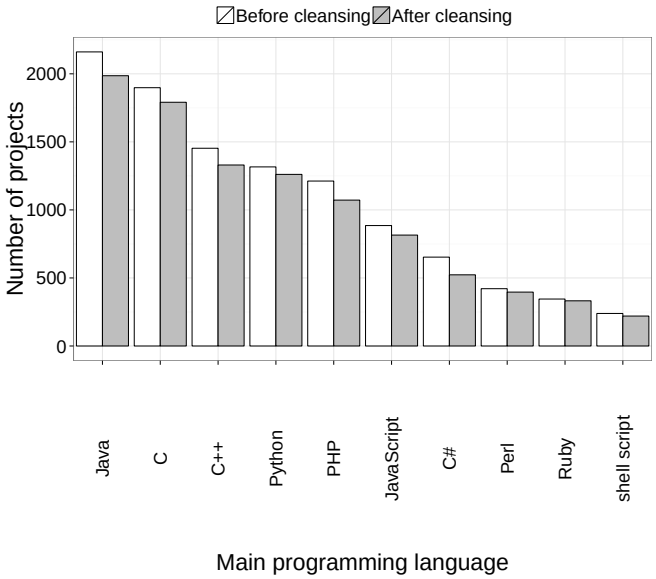

Figure 3: Histogram of the 10 most frequent (out of 76) main programming languages in the data set, before cleansing (left-hand side bars) and after cleansing (right-hand side bars).

- The analysis of data collected from Ohloh from the perspective of data quality (Section 3).
- The description of a cleansed version of the Ohloh data set. (Section 4).

The main goal of this research is to support a community effort to rigorously and transparently document data quality of public data sets. The data sets and processing tools used in this research are therefore available for replication (see Appendix A). The findings of this paper were shared with the Ohloh development team, which resulted in a request for further clarification and an indication that the team would review the results. At the time of camera-ready submission of this paper, no further response from Ohloh has been received.

5.1 Future work

We suggest the following next steps for research. First, since analysing one data set in isolation is limited to internal plausibility and consistency checks, further steps should be made by triangulating, or cross-validating, findings between data sets. Examples of such data sets are FLOSSmole [HCC06], FLOSSmetrics [GRD10], or GHTorrent [GS12].

Second, we recommend data source providers to use our quality findings to improve their data quality, for example by implementing automatic preventative quality checks. Examples of such checks are verification of version control configurations, and plausibility and consistency checks as described in this paper.

Finally, the statistical impact of cleansing data sets needs to be documented concretely to understand the consequences of using such data sets. Different cleansing strategies could be chosen, e.g., deletion, imputation, multiple imputation [Moc08], each with their own impact of

the data set. Ideally, each data set that is available will be accompanied with a bill-of-materials describing, among others, the consequences of the applied cleansing process.

Acknowledgements

Thanks to Tijs van der Storm for critical proofreading of this paper, and several reviewers for their constructive comments.

Bibliography

[AR09] O. Arafat, D. Riehle. The commit size distribution of open source software. In *Proceedings of the 42nd Hawaii International Conference on System Sciences*. Pp. 1–8. IEEE, 2009.

[ASKG11] S. Androutsellis-Theotokis, D. Spinellis, M. Kechagia, G. Gousios. Open source software: A survey from 10,000 feet. *Foundations and Trends in Technology, Information and Operations Management* 4(3-4):187–347, 2011.

[Bla13a] Black Duck Software, Inc. Ohcount. 2013. https://github.com/blackducksw/ohcount

[Bla13b] Black Duck Software, Inc. Ohloh. 2013. http://www.ohloh.net

[Bru13a] M. Bruntink. OhlohAnalytics data set and analysis tools. 2013. http://github.com/MagielBruntink/OhlohAnalytics

[Bru13b] M. Bruntink. Towards Base Rates in Software Analytics. *Science of Computer Programming, to appear, preprint available*, Oct. 2013.

[DR08] A. Deshpande, D. Riehle. The total growth of open source. *Open Source Development, Communities and Quality*, pp. 197–209, 2008.

[GRD10] J. M. González Barahona, G. Robles, S. Dueñas. Collecting data about FLOSS development. In *Proceedings of the 3rd International Workshop on Emerging Trends in FLOSS Research and Development*. Pp. 29–34. ACM Press, 2010.

[GS12] G. Gousios, D. Spinellis. GHTorrent: Github's data from a firehose. In *Proceedings of the 9th Working Conference on Mining Software Repositories (MSR 2012)*. Pp. 12–21. 2012.

[Has08] A. E. Hassan. The road ahead for mining software repositories. In *Proceedings of the 24th IEEE International Conference on Software Maintenance, Frontiers track*. Pp. 48–57. 2008.

[HCC06] J. Howison, M. Conklin, K. Crowston. FLOSSmole: A collaborative repository for FLOSS research data and analyses. *International Journal of Information Technology and Web Engineering* 1:17–26, 07 2006.

[HT08] I. Herraiz Tabernero. *A statistical examination of the properties and evolution of libre software*. PhD thesis, Madrid, Oct. 2008.

[KCM07] H. Kagdi, M. L. Collard, J. I. Maletic. A survey and taxonomy of approaches for mining software repositories in the context of software evolution. *Journal of Software Maintenance and Evolution: Research and Practice* 19(2):77–131, 2007.

[KSV09] P. Klint, T. van der Storm, J. Vinju. RASCAL: A Domain Specific Language for Source Code Analysis and Manipulation. In *Proceedings of the 9th IEEE International Working Conference on Source Code Analysis and Manipulation*. Pp. 168–177. 2009.

[Lie10] G. A. Liebchen. *Data cleaning techniques for software engineering data sets*. PhD thesis, Brunel University, School of Information Systems, Computing and Mathematics, 2010.
http://dspace.brunel.ac.uk/handle/2438/5951

[MCH+12] T. Menzies, B. Caglayan, Z. He, E. Kocaguneli, J. Krall, F. Peters, B. Turhan. The PROMISE Repository of empirical software engineering data. June 2012.
http://promisedata.googlecode.com

[Moc08] A. Mockus. Missing Data in Software Engineering. In Shull et al. (eds.), *Guide to advanced empirical software engineering*. Pp. 1–16. Springer, 2008.

[MZ] T. Menzies, T. Zimmermann. Software Analytics: So What? *IEEE Software* 30(4):0031–37.

[MZ12] T. Menzies, T. Zimmermann. Goldfish bowl panel: software development analytics. In *ICSE 2012: Proceedings of the 2012 International Conference on Software Engineering*. IEEE Press, June 2012.

[NZB13] M. Nagappan, T. Zimmermann, C. Bird. Diversity in Software Engineering Research. In *Proceedings of the 9th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*. ACM, Aug. 2013.

[R D08] R Development Core Team. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, 2008.
http://www.R-project.org

[RHH12] D. Rodriguez, I. Herraiz, R. Harrison. On software engineering repositories and their open problems. In *Proceedings of the First International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*. Pp. 52–56. IEEE, 2012.

[San] R. Sands. Measuring Project Activity. http://meta.ohloh.net/2012/04/measuring-project-activity/

[She11] M. Shepperd. Data Quality: Cinderella at the Software Metrics Ball? In *Proceedings of the 2nd International Workshop on Emerging Trends in Software Metrics*. ACM, 2011.

[SSSM13] M. Shepperd, Q. Song, Z. Sun, C. Mair. Data Quality: Some Comments on the NASA Software Defect Datasets. *IEEE Transactions on Software Engineering* 39(9):1208–1215, 2013.

[TAD+10] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, J. Noble. Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies. In *2010 Asia Pacific Software Engineering Conference (APSEC2010)*. Pp. 336–345. 2010.

[ZDL+11] D. Zhang, Y. Dang, J.-G. Lou, S. Han, H. Zhang, T. Xie. Software analytics as a learning case in practice: Approaches and experiences. In *International Workshop on Machine Learning Technologies in Software Engineering*. Pp. 55–58. ACM, 2011.

[ZMZ+13] F. Zhang, A. Mockus, Y. Zou, F. Khomh, A. E. Hassan. How does Context affect the Distribution of Software Maintainability Metrics? In *Proceedings of the 29th IEEE International Conference on Software Maintainability*. 2013.

A Replication

For replication of the research in this paper, the GitHub repository OhlohAnalytics [Bru13a] should be cloned using the tag 'SQM2014-REVIEW'. All data and developed tools are included. Detailed instructions for replication are provided on the repository's Wiki page.

B Details on validating SVN configurations

The following (case insensitive) regular expressions were used to identify properly configured SVN code directories. If none of the expressions can be matched on the SVN URL enlisted for the project, it is likely all project branches and tags are included in Ohloh's analysis:

- `.*/trunk/?`
- `.*/head/?`
- `.*/sandbox/?`
- `.*/site/?`
- `.*/branches/\w+`
- `.*/tags/\w+`