Code quality and mining repositories related to code quality: a systematic review

Jordan McDonald

1. ABSTRACT

…

Keywords – Code Quality, Open source, repository

1. BACKGROUND

An Overview of code quality is presented briefly in this section, this shall range from how code quality is defined to identifying select tools utilized to classify code. This information will then be contextualized in terms of repositories and the motivations behind using this technology as a form of storage and version control as an individual or a global software development team. In addition to this it will be briefly surmised how data is obtained from repositories and how these metrics relate to defining the quality of code.

* 1. Code Quality

Code quality in software and how to measure it is often a case of dispute, elements to be considered include how well tested, loosely coupled, efficient and maintainability. Aspects of these points are often very difficult to measure, how would you measure how readable code is without personal opinion interfering is a key example, this is often a subset of good design which has its own abstractions. The technical quality of source code (how well written it is) is an important determinant for software maintainability. When a change is needed in the software, the quality of its source code has an impact on how easy it is: (1) to determine where and how that change can be performed; (2) to implement that change; (3) to avoid unexpected effects of that change; and (4) to validate the changes performed. [2] Measurements may be used to obtain a picture of the quality both of a single component and of an entire program. Typical software metrics are the size of the code (measured in lines of code, number of statements, and so on) and the code complexity (measured through complexity figures such as the cyclomatic complexity). [3]

* 1. Code Quality Tools

There are numerous tools available from which static code can be analyzed, SonarQube is multi language application that provides in depth statistics and visualization. In addition to this SonarQube focuses on potential bugs, complexity measures, duplication and comments which are analyzed against the built in rule sets. Aside from the production level quality of the previous tool there are numerous additional options, some often language specific such as JTest, Clang, Jslint and TOAD.

* 1. Why perform a systematic literature review on this topic?

Research papers on this topic are present with stark degrees of variation, there are multiple papers that tackle the challenge of classifying code in terms of quality which still has merit as a resource for this paper. However there are a reduced number that focus on repositories, which can be classed in two categories – open source or academic, research papers in the context of industry in my search could not be acquired. The goal of my review to fill a gap in the market and draw the results from these sources and present a coherent analysis of the findings in the topic of repository code quality. This should assist in generating more research questions and provide a framework from which prior hypotheses can be compared, in order to provide support or contradict theories.

1. REVIEW METHODS AND CONDUCT

I have devised this system literature review according to the guidelines outlined by Kitchenham and Charters[1]. This paper documents all the steps that have been taken in order to generate the final results, while exploring in depth the intermediary processes which led to my conclusions.

* 1. Research Questions

RQ.1 – What metrics have peer-review literature used to measure software code quality in repositories?

To answer this question, peer review papers will have to be analyzed and metrics extracted to determine if

Patterns exist.

RQ.2 – What repository sources are most suitable for analyzing code quality?

To answer this question the volume, quality and depth of data from each source will have to be assessed.

RQ.3 – To what extent are peer reviewed studies successful in using repository data to estimate code quality?

* 1. Data sources

The extraction of data related to the topic will be source from digital libraries and databases, to facilitate this process I will browse the websites directly or use portals such as google scholar or the Queens University Belfast library search tools to identify sources of interest. The individual libraries I will be analyzing are listed below.

* <http://ieeexplore.ieee.org>
* <http://dl.acm.org/>
* <http://www.researchgate.net/>
* <http://link.springer.com/>
* <http://flosshub.org/>
  1. Search Strategy and Inclusion Process

Initially a research scope had to be identified, in this case I set a time period restriction of fifteen years (2000-2015). The next step was to designate research questions that would be suitable for the topic, this led to the formulation of keywords extracted from each relevant component of the research questions, synonyms and abbreviations would also be accounted for where appropriate. In addition to this the keyword ‘repository’ was extrapolated to account for popular products such as GitHub and Bit Bucket. At the end of the search phase the results were evaluated and modifications were made to the keywords and Boolean combinations utilized. The final set of keywords and Boolean searches performed are shown below.

ADD MORE AS NEEDED

Keywords – [GitHub, Bit Bucket, SVN, CVS, repository, code, quality, repositories]

Search Queries – [code AND quality AND repository, repository AND code OR quality, repository AND quality, repository AND bugs]

\*’repository’ has been interchanged with the repository products (not listed above)

* 1. Included/Excluded Studies

Phase one of the exclusion procedure is to A) determine if the paper is written in the English language B) it fits the ten year window previously mentioned and C) the full paper is available for analysis. The collated papers will then be subjected to the second phase which consists of reading the abstract and title for keywords. The final step will then be using my personal opinion to filter the papers down to the required maximum cap (eight papers) via a process of designating each paper as either relevant or irrelevant. This final step will be supplemented by checks such as excluding opinion based papers in order to focus on a quantitative analysis where possible, this phase will be performed on an ad-hoc basis with preference to papers that will contribute to satisfying this literature review.

Phase 2

1. Abstract?
2. Title?

Phase 3

1. Opinion based?
2. Relevant?

Phase 1

1. English
2. Time period
3. Full paper

8

16

11

10

Figure 1 – showing how the papers were filtered

* 1. Data Synthesis

Each paper that passes the filtering process is the subject to a full analysis, selected pieces of information were then stored in Microsoft Excel, which each paper being associated with seven columns, each representing a data category. The goal in this step was to place all the relevant data in a convenient location, this would allow relationships between metrics and sources to be identified much more rapidly. Fields for the Excel form in include:

* Context – Academic, industry or open source.
* Low level metrics – Information drawn from parsing the code base
* High level metrics – Information drawn from the repository logs
* Tool – Whether a tool was utilized
* Languages – Mixed or one particular language
* Outcomes – Conclusions drawn from the paper
* Miscellaneous – Additional details

1. RESULTS
   1. Initial results of peer reviewed literature

The process of selection papers yielded a final count of eight from which to perform a literature review, the years of publication range from 2002 to 2015. Two papers were present in the first half of the original search range, whereas the other six received publication from 2008 to 2014 would could demonstrate a shift in emphasis to exploring repositories and the code within them. This could be particularly due to the popularity surge of Bit Bucket and GitHub, however the data set is too small to provide concreate evidence to support this. Each paper extracted allowed the MS Excel form outlined previously to be fully populated with the data expected in each column. To show the sources of the research papers, refer to figure 2.

|  |  |
| --- | --- |
| **Source** | **Year** |
| IEEE Xplore | 2015 – 2012 - 2015 |
| ACM | 2005 – 2008 - 2014 |
| Research Gate | 2014 |
| Flosshub | 2002 |

Figure 2 – showing data sources and dates

In order to fulfill the requirements of research question one, the types of metrics need to be considered. From this I categorized the data used in each research paper into two categories, high level metrics and low level metrics. Refer to the vectors below for example data that some of the studies used.

High Level = [total commits, total contributors, mean additions, mean deletions, total LOC, total revisions, open bugs in the last six months, commit message parsing, comment quality, unique committers]

Low Level = [number of statements per method, number of methods, cyclomatic complexity, branch counts, readability, path frequency, lack of method cohesion, code clones, number of unconditional jumps]

Low level metrics are those which revolve around analyzing the code level of the repository, from this data such as the amount of functions, while loops and branches in the source code are evaluated. In contrast high level metrics place emphasis on the repository level, using data centered on commits, forks and contributors. These are often collected using tools or API’s, refer to figure 3 below to see the distribution of which repositories use which category of metric.

Figure 3 – shows the rate of metric usage in each paper

Of the eight successful research papers that fit the criteria seven of the studies used multiple programming languages, while the one paper that focused on a singular language, C, also happened to use a tool to perform the code quality analysis, known as Logiscope, the tool focused on low level metrics. In addition there stood variation in the context.

* 1. Results related to both research questions

To answer research question one, I have composed a list of every mentioned metric that is used by the eight papers in order to judge code quality. Each metric has been stated as well as the frequency it has been mentioned in each distinct paper, this will give a broad overview of popular measurements that have been utilized in the studies, refer to figure 3. \*some wording has been modified to allow for a consistent terminology, while filtering out very general terms

|  |  |  |  |
| --- | --- | --- | --- |
| **High Level Metric** | **Frequency** | **Low Level Metric** | **Frequency** |
| Total commits | 4 | Total times a space followed a comma | 1 |
| Number of files | 1 | Number of comments | 3 |
| Total number of changes | 1 | Literal string total | 1 |
| Total transactions | 1 | Operator total | 1 |
| Average changes per file | 1 | Amount of characters in comments | 1 |
| LOC | 3 | Total function definitions | 1 |
| Cyclomatic complexity | 4 | Total while loops | 1 |
| Number of open bugs (6 months) | 1 | Total for loops | 1 |
| Number of closed bugs (6 months) | 1 | Total termination tokens | 1 |
| Number of unique watchers | 1 | Amount of 4 space indents | 1 |
| Rate of developer turnover | 1 | Assertion message violations | 1 |
| Growth in active developers | 1 | Total leaning tabs | 1 |
| Number of contributors | 3 | Number of statements per method (mean) | 1 |
| Commit message parsing | 1 | Total operators per method |  |
| Total bug fixes | 1 | Maximum nest | 2 |
| Total insertions | 1 | Number of paths per method (mean) | 1 |
| Total number of forks | 1 | Number of unconditional jumps | 2 |
| Current issues | 1 | Comment frequency (over statements) | 2 |
| Release count | 1 | Program length | 2 |
| .. | .. | Average statement size | 1 |
| .. | .. | Number of inputs and outputs of a method | 2 |
| .. | .. | Number of lines per method | 1 |
| .. | .. | Number of public methods | 1 |
| .. | .. | Lack of cohesion in methods | 2 |
| .. | .. | Number of statements | 2 |
| .. | .. | Number of protected methods | 1 |
| .. | .. | Coupling between objects | 2 |
| .. | .. | Depth of inheritance tree | 1 |
| .. | .. | Number of child classes | 1 |
| .. | .. | Code churn | 1 |
| .. | .. | Path frequency | 1 |
| .. | .. | Path density | 1 |

Cover repo type if room

* 1. Paper by paper analysis

1. Commit Quality in Five High Performance Computing Projects
2. Mining student CVS repositories for performance indicators
3. Code quality analysis in open source software development
4. Comparing Design and Code Metrics for Software Quality Prediction
5. Evaluating the Quality of Open Source Software
6. A Large Scale Study of Programming Languages and Code Quality in Github

G) Measuring Code Quality to Improve Specification Mining

1. GitHub Projects. Quality Analysis of Open-Source Software

1. DISCUSSIONS // mention no industry context
2. CONCLUSIONS
3. REREFENCES
4. B. Kitchenham, S. Charters. Guidelines for Performing Systematic Literature Reviews in Software Engineering, Technical Report EBSE-2007-01, School of Computer Science and Mathematics, Keele University, 2007.
5. Robert Baggen, et al. Standardized code quality benchmarking for improving software maintainability, Springer Science+Business Media, LLC 2011
6. Ioannis Samoladas et al, Open Source Software Development Should Strive for even greater code maintainability, Communications of the ACM Volume 47 Issue 10, October 2004 Pages 83-87
7. Keir Merle et al, Mining student CVS repositories for performance indicators, MSR '05 Proceedings of the 2005 international workshop on Mining software repositories
8. Loannis Stamelos et al, Code quality analysis in open source software development, Information Systems Journal (Impact Factor: 2.07). 01/2002; 12(1):43-60. DOI: 10.1046/j.1365-2575.2002.00117.x
9. Yue Jiang et al, Comparing Design and Code Metrics for Software Quality Prediction, 05/2008; DOI: 10.1145/1370788.1370793
10. Diomidis Spinellis et al, Evaluating the Quality of Open Source Software, Electronic Notes in Theoretical Computer Science 233 (2009) 5–28
11. Baishakhi Ray et al, A Large Scale Study of Programming Languages and Code Quality in Github, FSE 2014 Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering
12. Claire Le Goues and Westley Weimer, Measuring Code Quality to Improve Specification Mining, IEEE transactions on software engineering, vol 38, no 1, January/February, 2012 175
13. Oskar Jarczyk1 et al, GitHub Projects. Quality Analysis of Open-Source Software, Volume 8851 of the series Lecture Notes in Computer Science pp 80-94
14. Agrawal, K. Commit Quality in Five High Performance Computing Projects, Software Engineering for High Performance Computing in Science (SE4HPCS), 2015 IEEE/ACM 1st International Workshop, 2015