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 the 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.

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.

* 1. Data sources **ADD MORE AS I GET THEM**

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/>
* …
  1. Search Strategy and Inclusion Process BEEF THIS UP IF NEED BE AT END (only use papers with repository and quality in synopsis?)

Initially a research scope had to be identified, in this case I set a time period restriction of ten years (2005-2015). 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]

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

N4

N

N2

N3

Figure 1 – showing how the papers were filtered

* 1. Data Synthesis - TODO

1. RESULTS
2. DISCUSSIONS
3. CONCLUSIONS
4. REREFENCES
5. 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.
6. Robert Baggen, et al. Standardized code quality benchmarking for improving software maintainability, Springer Science+Business Media, LLC 2011
7. 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

POSSIBLY USEFUL LINKS

<http://dl.acm.org/citation.cfm?id=1083150>

http://www.researchgate.net/profile/Lefteris\_Angelis/publication/220356602\_Code\_quality\_analysis\_in\_open\_source\_software\_development/links/0912f50e858dbb10fb000000.pdf