Jincheng He jinchenh@usc.edu jincheng@alumni.usc.edu

Summary of the Paper

This paper is focused on using static code analysis tools to get software metrics and revealing the relation between commits and software quality in order to better serve software development. By defining the conception of "**main module**" and "**impactful commit**", it analyzes the connection between "impactful commits" and the different aspects of software metrics as well as within the aspects.

In the first part, the subjects, **commits** from **Open Source Software**, the approach, **Static Code Analysis** are introduced while in the part two, more specific details are given. Three SCA tools used in the research, metrics selected for researching and git where the commits data comes from are shown.

In the third part, the important conception of "**main module**" and "**impactful commit**" are specifically defined which is used throughout the whole research.

In part four raises four main questions which is going to be researched on and solved in part five as well as illustrates the system filter applied on the original dataset and how to determine main modules in the real projects/ **revisions**.

After applying the approach defined in previous parts onto the dataset, results are reached relative to those mentioned in part four:

1. Prove the centrality of the defined "main module" while pointing out the exceptions result from **small modules, agents, plugins**, size of **test suite**, diversity of **programming languages**, **multiple subprojects**, **architecturally independent components** and extremely active contributors.
2. Illustrate the multiple reasons resulting in **compiling problems**, which can serve as an alarm for software developers and supervisors. While figuring out the reasons, the paper also gave abundant examples as well as exceptions with their causes.
3. Reveal the relation between "**impactful commit**" and nine selected **software metrics** while giving the exceptions and insight into those exceptions.
4. Reveal the relations **between** different **software metrics**.

Part six lists related work with this research and points out the difference.

Three aspects are listed as part seven before the end of the paper, suggesting the underlying limitations of this research and the approach used to overcome them:

1. Limitation of tools used in the research -- widely used static code analysis tools.
2. Limitation of determination of main modules -- have insights into each project.
3. Limitation of sample selected -- select apache as data source.

In the conclusion part, results are presented, and future aims are listed including software evolution, dynamic analysis, regression tests, machine learning and natural language processes.

My ideas

1. Although the selection of programming language, CVS, and static code analysis tools is refined. Still, a wider selection of each aspects may lead to, in some degree, different or more convincing results.

   Choosing Python projects may be a good choice, since Python is also popular and it's a kind of dynamic language, different from java, which has more possibility to get a surprising result.

   As for version control, perhaps SVN can be used as another choice, but the feasibility needs further confirmations.

   For SCA, further research may lie in other aspects beside the nine metrics listed in the paper, such as the density of commits. But this may need further reading in order to decide the most valuable metrics.

2. For the "main modules". Although in the paper, a lot of effort is applied on defining (part three) and proving (RQ1) this. There may be another way to make the determination more convincing. If possible, we can cooperate with development teams and set up a list of prefixes to comments while committing, marking the changes to main modules. For example, while committing, the comment should be like "[type_of_module] [type_of_change] content". Again, feasibility should be reconsidered since it will be easier to gain data but the cooperation itself might be difficult to realize.

3. As nine types of software metrics are listed and analyzed in the paper, more might be available. In addition, some other static code analysis could also be used especially those which still have new releases. (According to Wikipedia®, the latest release of FindBugs lied in 2015 which might mean, to some extent, out of date, further information should refer to jdk release notes to see whether in 2015 to 2017 Java^{TM} have significant changes.) This may also get us more software metrics for selection.

4. In RQ4 result, internal relations of nine metrics are find and put into a table. Still, in the paper reveals only one-to-one relation between metric. A further insight of n-to-1 relation may be valuable to the analysis of software quality. For example, what is the percentage of changes in complexity within the dataset in which all three basic aspect changes? Something like this may lead to new conclusions.

5. In RQ1 result, CXF-Fediz is introduced as an exception resulting from its special architecture (two large architecturally independent components). A research going deeper into the relation between commits, architecture and software quality might also reveal some new points. For example, commits may have special characteristics in projects of different architectures, and the approaches may also differs to upgrade the software quality.