Open Source Evolution Analysis

Izzat Alsmadi
Kenneth Magel
Department of computer science
North Dakota state university
{izzat.alsmadi, kenneth.magel}@ndsu.edu

ABSTRACT

Source code analysis is important for software management. It enables us recognize strengths and weaknesses in our earlier projects or releases. We developed a source code analysis tool. This tool gathers several metrics from C/C++, C# or Java source codes. In this paper, we will use the tool to analyze some of the open source code projects. We will study the selected projects release evolutions and compare some characteristics between the same project releases, as well as among different projects. Different programming language code and development styles will be studied through those open source projects.

General Terms
Source code analysis.
Keywords
Open source, code metrics.

1. INTRODUCTION

The knowledge gathered by software metrics plays an important role in software management. This knowledge can be used to build classification or proxy models that can be used toward future projects or releases. A software metric tool helps us know the required information to build such models. The metrics that are gathered need to be compiled to make some hypothesis and assumptions about the model. Using rules similar to those rules used in data mining classification models, the more we apply this model, the more we can tune its assumptions and gain confident in its results.

The paper analyzes the results of some source code charts from selected open source available projects. Those projects will be studied through selected number of their releases.

2. RELATED WORK

Godfrey et al. studied the LOC releases evolution for Linux[18]. Capiluppi et al. suggested that understandability decreases as time passes by, and focused on code and module sizes [19]. Stroulia et al. used CVSChecker to study temporal source code activities [20]. Marjanovic proposed a meta model framework for code release history Systems [21]. Scacchi studied the game open source development practices[22]. Raja et al. explored some important software characteristics that contribute to consistent software quality in Linux [23]. Moreno, et al. introduced Jeliot3 , as a visual source code evaluation tool[13]. Graver evaluated object oriented refactoring process and evolution of a compiler[14].

3. SWMETRIC TOOL

SWMetric is a tool we developed to gather metrics on the function and the class level. The tool can analyze code of C/C++/C# or Java. It is originally

22nd IEEE International Conference on Software Maintenance (ICSM'06)
0-7695-2354-4/06 $20.00 © 2006 IEEE

developed as part of a student research project for Honeywell aviation division.

4. GOALS AND APPROACHES

1. Study the relation between project releases and LOC size variation. We first studied how different projects LOC changes with the releases.

In traditional approaches, we should not see much of new lines of codes produced with the recent releases. The diagram, however, shows an instability of the amount of new LOC produced, most of the projects tested, show an increase in a release and a decrease in the next one. However, this is expected in the case of open source projects where there is much of instability in terms of developers, their abilities, available time and other resources.

2. Study the LOC efficiency and Declaration Percent of the total code size. Most of the projects studied have a steady LOC efficiency. This efficiency is almost the same for the different projects and it also does not vary for most of the time with releases progress. The efficiency percentage for most projects is between 70-80 %.

Initially, in Software automation development scenarios, we should have a low efficiency, where most of the lines are counted in LOC , but not in SLOC, as they are automatically generated by a code generation tool. We should have an increase for the efficiency with releases, or increase for SLOC lines from the total LOC lines.

The Declaration percentage of data declarations to the total code size indicates how much of the total code size is used for declarations, method headers and global variable.

3. Release vs. LOC/function

Typically, programs should have a fixed size of functions of no more than (20-30) LOC's[9].

Some of the open source projects may have been developed by different individuals. An increase of the functions sizes with time indicates problems in planning that are causing functions to expand or inflate. Of the studied projects, two show a relatively fixed amount of LOC/function (~ 20) which may indicate stable coding, better predictions and management. The diagram shows a good percent of projects that had and sustained a fixed LOC/function.

4. MCDC and nesting per function

MCDC and nesting are indications of how many decisions and nodes are there in each function. These are important values for software testing.

5. Code distribution. We studied code distribution through dividing code into three parts : comment lines, declaration and global variable lines, and the last part is the rest of the source code.

5.CONCLUSION AND FUTURE WORK

We are willing to explore more of the available open source codes in order to be able to make some hypotheses and be able to build software classification models. We will make more detailed studies to compare individual to company coding styles, and the common or different characteristics of coding among the different programming languages.

6. REFERENCES

1. Free Software Foundation, 2006, 15 May 2006. <http://directory.fsf.org/libs/c/>.
2. Open Watcom.org, 02-2006, 15 May 2006. Openwatcom.org/ftp/source/>.
3. Sun Microsystems, 05-2006. <http://java.sun.com/products/archive/>.