Assessing The Influence Of Software Cloning On Software Maintenance Cost

Neha Kohli
M.Tech
Sri Sai Institute of Engg. & Tech.

Gagandeep Singh
Asst. Prof.
Sri Sai Institute of Engg. & Tech.

Abstract

Software engineers often build new procedures by cloning, copying an existing one with similar requirements, and slightly modifying it. While this may be easier than extracting the common part, and sharing it in a library, it increases the system size and often leads to higher maintenance costs. The occurrence of clones is highly dependent on the system architecture and development model, and has been studied in the past for a few large procedural systems.

In this dissertation a new mechanism is proposed which will find the effect of software clones on the maintenance of open source software. In order to achieve this objective a suitable tool will be selected. Different versions of open source software will be evaluated in the selected tool in the presence and absence of clones. In order to performance comparison, different software engineering's well known metrics will be selected.

Introduction

1.1.Overview

A "clone" in software is a segment of code that has been created through duplication of another piece of code. [1] [2] e.g. Copy and paste. Clones may start to appear for any one of the following reasons: · Development time: A software engineer clones a procedure when needing similar functionality instead of extracting the common reusable part. It looks quicker to achieve, may be faster for the initial implementation, but often leads to more expensive code to maintain.

i. **Communication:** A software engineer borrows code from a colleague but cannot extract the common reusable part. Either he is not sufficiently knowledgeable about the cloned procedure, or he cannot convince the other software engineers involved to include this reusable procedure in the library and modify their code to use it.

ii. **Structural:** A software engineer borrows code from another subsystem but cannot avoid cloning because the other subsystem may not be modified; the other subsystem may belong to a different department or may not be modifiable (stored in non-volatile memory in embedded systems, or frozen after a lengthy testing/qualification procedure).

iii. **Coincidence:** It may happen that two software engineers came up with similar procedures independently, thus leading to look-alikes more than clones. It would be beneficial to replace them with a reusable procedure. These are typically much more difficult to detect as they may achieve the same functionality with somewhat different apparent structures.

iv. **Efficiency:** Considerations of efficiency may make the cost of a procedure call or method invocation, too high a price. 4

1.2 Problems with clones in code

Unjustified duplicated code gives rise to severe problems:

i. If one repairs a bug in a system with duplicated code, all possible duplications of that bug must be checked.
ii. Code duplication increases the size of the code, extending compile time and expanding the size of the executable.
iii. Code duplication often indicates design problems like missing inheritance or missing procedural abstraction. In turn, such a lack of abstraction hampers the addition of functionality.
iv. Errors in the systematic renaming can lead to unintended aliasing, resulting in latent bugs that show up much later.
v. The effect of all of these is a form of "software aging" or "hardening of the arteries" that result when even small design changes become very difficult to make.

1.3 Types of clones

In general, clones may be described using the following typology:

i. Type I: An exactly identical source code clone, i.e. no changes at all.
ii. Type II: An exactly identical source code clone, but with indentation, comments, or identifier (name) changes.
iii. Type III: A functionally identical clone, but with small changes made to the code to tailor it to some new function.
iv. Type IV: A functionally identical clone, developed possibly with the originator unaware that there is a function already available that accomplishes essentially the same function.

1.4 Need of Software Cloning

The copying and duplication of source code has been studied in software engineering under several topic areas. Copy and paste programming is a common activity but it introduces a negative point to reuse by creating clones. Cloning, the copying and duplicating of blocks of code, is the basic means of software reuse [25]. The most prominent research area within software engineering which studies the duplication of source code is clone analysis; other treatments of code copying and duplication include studies of programmer behaviour, code plagiarism detection algorithms, the post-modern programming movement, as well as the development of some specialized programming languages. Clone analysis is, however, the largest area of research related to code duplication.

During the software development cycle, code cloning is easy and inexpensive (in both effort and money). However, this cloning practice can complicate software maintenance and it has been suggested that too much cloned code is a risk, albeit the practice itself is not generally considered harmful [37]. Not only it effects the maintenance phase but also leads to various problems like clones increases Resource Requirements, increases Defect Probability and also increases the Probability of Bad Design.

1.5 Scope of Cloning

Today various programming methodologies are being used in the software development process. The practice of copy and paste code is extensively acknowledged but is rarely explicitly accounted for in models of software

development. The software industry seems to be embracing yet another change in the way it does business. Because of their emphasis on agility and time-to-market, many software shops have made the move to extreme programming and agile methods.

To implement these methods adherents embrace XP practices like pair programming, refactoring and collective code ownership to generate their products. These releases, which are working versions of the product, not prototypes, are used to demonstrate functions and features to stakeholders who help shape their form through refactoring and continuous integration. Programming methodology is accompanied with high degree of reuse. Refactoring is one of the main practices of Extreme Programming and thus refactoring is used in the cloning process. Implementing refactoring patterns and detecting clones helps in improving the code and removing the clones.

Code clones are considered harmful in software development, also provides hindrance in software evolution and maintenance phase. So various techniques are used to detect them and remove them from the software. One of the approaches is to try to eliminate them through refactoring. Various refactoring patterns are used to detect the clones and hence remove them. Other techniques are also used to eliminate them.

1.5 Objective

Code clone detection could be useful in many ways e.g. decreasing the cost of software maintenance activities. Detection of duplicate code fragments increases understand-ability of software systems and may help system maintainers to increase code quality of the existing system.

Detection of duplicate code fragments leads to efficiency on the software maintenance process and decreases maintenance cost. The aim of this paper is to evaluate the effect of software clones on the maintenance of open source software.

In order to achieve this objective, the following objectives must be fulfilled.

I. The main objective is to evaluate the effect of software clones on the maintenance of open source software.

II. The effect of code cloning v/s clone free open source software on the “code metrics” like Method Level Metrics, Class Level Metrics, File Level Metrics and Package Level Metrics.

III. Quantitative measurement of Maintainability Index Metric

Quantitative measurement of an operational system's maintainability is desirable both as an instantaneous measure and as a predictor of maintainability over time. Efforts to measure and track maintainability are intended to help reduce a system's tendency toward "code entropy" or degraded integrity, and to indicate when it becomes cheaper and/or less risky to rewrite the code than to change it.

Software Maintainability Metrics Models in Practice is the latest report from an ongoing, multi-year joint effort (involving the Software Engineering Test Laboratory of the University of Idaho, the Idaho National Engineering Laboratory, Hewlett-Packard, and other companies) to quantify maintainability via a Maintainability Index (MI).

Problem Statement

Software Cloning effects software maintenance and other engineering efforts. Cloning at design as well as at code level is seemed as an obstacle so it is needed to be removed. And hence cloning has grown as an active area in software engineering research community yielding numerous

techniques, various tools and other methods for clone detection and removal. Cloning in source code has been reported for different programming languages and application domains.

In this dissertation a new mechanism is proposed which will find the effect of software clones on the maintenance of open source software. In order to achieve this objective a suitable tool will be selected. Different versions of open source softwares will be evaluated in the selected tool in the presence and absence of clones. In order to performance comparison, different software engineering's well known metrics will be selected.

However it has been seen in base paper Meirelles et al. (2010) [1] has studied a significant number of Free Software projects for evaluations. This dissertation extends its work and study for software cloning by using well known metrics not only at code level also at effect of code cloning v/s clone free open source software on the "software metrics" like Method Level Metrics, Class Level Metrics, File Level Metrics and Package Level Metrics and Quantitative measurement of Maintainability Index Metric.

Research methodology

In order to achieve the objectives a software analysis tool will be selected and different software will be tested in the case of clone as well as same software when it is clone free.

The following metrics will be selected for evaluation:

3.2.1 Method Level

a. Number of lines of Code (NLOC_MTD)
b. Percentage of comments (POC_MTD)
c. Number of Variables (NOV_MTD)
d. Number of Unused Variables (NOUV_MTD)
e. Number of comment lines (CL_MTD)
f. Number of Parameters (NOP_MTD)
g. Number of Unused Parameters (NOUP_MTD)

3.2.2 Class Level

a. Number of Lines of Code(NLOC_CLS)
b. Number of Parents (NOPNT_CLS)
c. Number of Fields (NOFLD_CLS)
d. Percentage of Non-Private Fields (NPFP_CLS)
e. Percentage of Non-Private Methods (NPMP_CLS)
f. Number of Inner Classes (NOIC_CLS)

3.2.3 File Level

a. Number of Lines of Code (NLOC_FIL)
b. Halstead Effort / Volume (HE_FIL, HV_FIL)
c. SEI Maintainability Index (MI_FIL)

3.2.4 Package Level

a. Number of Lines of Code (NLOC_PKG)
b. Number of Classes (NOCLS_PKG)
c. Number of Interfaces (NOIFC_PKG)
d. Number of Files (NOF_PKG)

Conclusion and Future work

New mechanism is proposed which will find the effect of software clones on the maintenance of open source software. In order to achieve this objective a suitable tool will be selected. Different versions of open source software will be evaluated in the selected tool in the presence and absence of clones. In order to performance comparison, different software engineering's well known metrics will be selected.

Bibliography

[1] Paulo Meirelles, Carlos Santos Jr., Joao Miranda, Fabio Kon, Antonio Terceiro and Christina Chavez (2010), "*A Study of the Relationships between Source Code*

[2] *Metrics and Attractiveness in Free Software Projects"*, IEEE, 2010.

[3] Sandro Schulze, Sven Apel and Christian K¨astner, *"Code Clones in Feature-Oriented Software Product Lines",*ACM, GPCE'10, October 10–13, 2010.

[4] Debarshi Chatterji1, Beverly Massengill2, Jason Oslin1, Jeffrey C. Carver and Nicholas A. Kraft (2010), *"Measuring the Efficacy of Code Clone Information: An Empirical Study"*, in the proceedings of the Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU) at the ACM and SPLASH Conferences, October, 2010.

[5] Jens Krinke (2011*),*" *Is Cloned Code older than Non-Cloned Code?"*, ACM, IWSC 2011, May 23, 2011.

[6] Liliane Barbour, Foutse Khomh and Ying Zou (2011), *"Late Propagation in Software Clones",* IEEE, 2011.

[7] Foyzur Rahman, Christian Bird and Premkumar Devanbu (2011), *"Clones: what is that smell"*, Springer Science, Business Media, LLC 2011.

[8] Minhaz F. Zibran Chanchal and K. Roy (2011), *"A Constraint Programming Approach to Conflict-aware Optimal Scheduling of Prioritized Code Clone Refactoring",* IEEE, 2011.

[9] Gehan M. K. Selim, Liliane Barbour, Weiyi Shang, Bram Adams, Ahmed E. Hassan and Ying Zou (2010), *" Studying the Impact of Clones on Software Defects"*, IEEE, 2010.

[10] Chanchal K. Roy, James R. Cordy and Rainer Koschke (2009), *"Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach"*, Science of Computer Programming February 24, 2009.

[11] Benjamin Hummel, Elmar Juergens, Lars Heinemann and Michael Conradt (2011), *"Index-Based Code Clone Detection: Incremental, Distributed, Scalable",* IEEE, 2011.