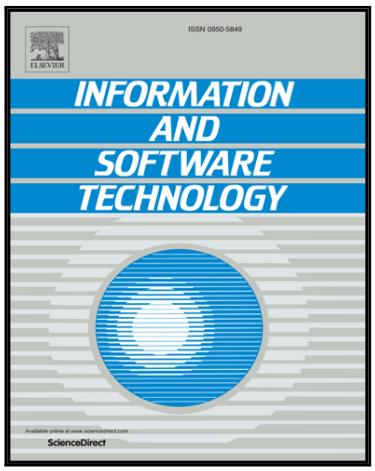

Understanding metric-based detectable smells in Python software: a comparative study

Zhifei Chen , Lin Chen , Wanwangying Ma , Xiaoyu Zhou , Yuming Zhou , Baowen Xu

PII: S0950-5849(16)30169-0
DOI: 10.1016/j.infsof.2017.09.011
Reference: INFSOF 5884

To appear in: *Information and Software Technology*

Received date: 22 September 2016
Revised date: 13 August 2017
Accepted date: 22 September 2017

Please cite this article as: Zhifei Chen , Lin Chen , Wanwangying Ma , Xiaoyu Zhou , Yuming Zhou , Baowen Xu , Understanding metric-based detectable smells in Python software: a comparative study, *Information and Software Technology* (2017), doi: 10.1016/j.infsof.2017.09.011

Understanding metric-based detectable smells in Python software: a comparative study

Zhifei Chen[#1], Lin Chen[#2+], Wanwangying Ma[#3], Xiaoyu Zhou[*4], Yuming Zhou[#5], Baowen Xu[#6+]

[#](State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, China)

[*](School of Computer Science and Engineering, Southeast University, Nanjing 210096, China)

{[1]chenzhifei, [3]wwyma}@smail.nju.edu.cn, {[2]lchen, [5]zhouyuming, [6]bwxu}@nju.edu.cn, [4]zhouxy@seu.edu.cn

Abstract

Context: Code smells are supposed to cause potential comprehension and maintenance problems in software development. Although code smells are studied in many languages, e.g. Java and C#, there is a lack of technique or tool support addressing code smells in Python.

Objective: Due to the great differences between Python and static languages, the goal of this study is to define and detect code smells in Python programs and to explore the effects of Python smells on software maintainability.

Method: In this paper, we introduced ten code smells and established a metric-based detection method with three different filtering strategies to specify metric thresholds (Experience-Based Strategy, Statistics-Based Strategy, and Tuning Machine Strategy). Then, we performed a comparative study to investigate how three detection strategies perform in detecting Python smells and how these smells affect software maintainability with different detection strategies. This study utilized a corpus of 106 Python projects with most stars on GitHub.

Results: The results showed that: (1) the metric-based detection approach performs well in detecting Python smells and Tuning Machine Strategy achieves the best accuracy; (2) the three detection strategies discover some different smell occurrences, and Long Parameter List and Long Method are more prevalent than other smells; (3) several kinds of code smells are more significantly related to changes or faults in Python modules.

Conclusion: These findings reveal the key features of Python smells and also provide a guideline for the choice of detection strategy in detecting and analyzing Python smells.

Key words Python; code smell; detection strategy; software maintainability

I. INTRODUCTION

Code smells [2, 14] are particular bad patterns in source code which violate important principles of software design and implementation issues. Particularly, code smells indicate when and what refactoring can be applied [38, 43-44]. It does not mean that no code smells are allowed to appear, but rather that code smells are essential hints about beneficial refactoring. Various studies have confirmed the effects of code smells on different maintainability related aspects [54, 61, 62], especially changes [4-6, 57-58], effort [7-9], modularity [55], comprehensibility [10, 11], and defects [12, 46, 56-58].

Existing approaches of detecting code smells include metric-based [1, 16-18, 26], machine learning [19-21], history-based [22-23], textual-based [60], and search-based [41] approaches. A large group of code smells can be measured by software metrics to quantify their characteristics, hence metric-based detection technique becomes the most common way of detecting code smells. Measuring code smells requires proper quantification means of design rules and practices [16], which raises a set of challenge. Above all, metric values leave the engineer mostly clueless concerning the ultimate cause of the anomaly that it indicts. Credible thresholds are established to promote the use of metrics as an effective measurement instrument, which is a prominent challenge for metric-based detection technique [13, 17].