Do Bad Smells Indicate "Trouble" in Code?

Min Zhang
University of Hertfordshire
Hatfield, UK
m.1.zhang@herts.ac.uk

Tracy Hall
Brunel University
Uxbridge, UK
tracy.hall@brunel.ac.uk

Nathan Baddoo
University of Hertfordshire
Hatfield, UK
n.baddoo@herts.ac.uk

Paul Wernick
University of Hertfordshire
Hatfield, UK
p.d.wernick@herts.ac.uk

ABSTRACT
In 1999 Fowler et al. identified 22 Bad Smells in code to direct the effective refactoring. These are increasingly being used by software engineers. However, the empirical basis of using Bad Smells to direct refactoring and to address 'trouble' in code is not clear. Our project aims to empirically investigate the impact of Bad Smells on software in terms of their relationship to faults.

Categories and Subject Descriptors
D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement – *restructuring, reverse engineering, and reengineering.*

General Terms
Design, Reliability, Experimentation, Human Factors.

Keywords
Bad Smells, Faults, Open Source

1. INTRODUCTION
Bad Smells are structures in source code informally identified by Fowler et al. [5]. Fowler et al. [5] state that Bad Smells can give "indications that there is trouble that can be solved by a refactoring". Bad Smells are widely used for detecting refactoring opportunities in software [7].

Although Bad Smells make common sense, no empirical evidence has been provided by Fowler et al. to support the efficacy of using Bad Smells. Indeed, two recent studies [8, 10] report that Bad Smells may not indicate problems that significantly affect software. This may mean that Bad Smells do not need to be refactored. Furthermore, while Fowler et al. [5] claim that Bad Smells cause problems that should be fixed by refactoring, they are not specific about the problems caused by Bad Smells. A review of the literature shows that there are many interpretations of the consequences of Bad Smells. However most researchers focus on Bad Smells in terms of increased faults and reduced maintainability [8, 10]. Faults are widely cited as an important indicator of software quality [3]. Consequently we will investigate the relationship between Bad Smells and faults. Our research approaches and plans are described in this paper.

DEFECTS'08, July 20, 2008, Seattle, Washington, USA.

2. SYSTMATIC LITERATURE REVIEW
The first thing that we have done is to systematically investigate the current literature and summarise the current state of knowledge of Bad Smells. A systematic literature review [6] is a particularly effective approach for this.

A systematic literature review (SLR) is a methodology to identify and evaluate all available research relevant to particular research questions and is a useful approach to "summarise the existing evidence concerning a treatment or technology" [6]. It is a research methodology used extensively in medical research. However, Woodall et al. [11] observe that conducting a full SLR is too time-consuming, and it can easily extend beyond the time schedule of a research project. To use the SLR approach efficiently we developed a light-weight SLR protocol [12].

We have applied our light-weight SLR protocol to review all studies of Bad Smells published by IEEE in the last 5 years (2002-2006) [12]. Our findings show that the Duplicated Code Bad Smell has attracted most attention, and has a research profile different from that of other Bad Smells. Our results also show that the status of knowledge varies between different Bad Smells. In particular the Feature Envy, Long Method and Large Class Bad Smells have different research features. For example, the motivation for studies of these Bad Smells is mainly focused on improved understanding, while the motivation for studies of other Bad Smells focuses on enhancing the tools or methods for detecting them. The reasons for these differences need further investigation. We also found that studies of Bad Smells mainly use objective research data. Very little subjective data was used in previous studies of Bad Smells. A better balance of objective and subjective analysis would also be valuable in future studies of Bad Smells. Finally, only a few empirical studies have been conducted to examine the effects of Bad Smells. This evidence suggests that the impact of Bad Smells remain far from being fully understood.

3. METHODOLOGY
Based on the results of our SLR, an empirical study will be conducted to further examine the impact of Bad Smells. In particular, we focus on addressing the following research question:

- *What is the relationship between Bad Smells and faults?*

3.1 Experiment Design
We will address the above research question using both objective and subjective data. Our SLR suggests that relatively few current studies of Bad Smells use subjective data. However, the resultant dependence on objective data does not give us the full picture of Bad Smells. Pfleeger [9] indicates that "software

development is as much an art as a science"; software engineers' instinct plays an important part in software development. While objective data can provide evidence that a new technique is important, subjective data uncovers whether the technique is useable. Both aspects need to be addressed in software engineering studies. Hence, two modes of investigation will be employed in this research project.

Study using code-based metrics data: Firstly, we will conduct an experiment by using objective data. In particular, we will use data from Eclipse open source project. We will monitor Bad Smells using static code analysis techniques in a sequence of software releases from Eclipse. We will investigate:

1. Whether a class containing Bad Smells correlates to faults.
2. How many faults are caused by classes which contain Bad Smells

Study using developer opinion: Our second study will investigate developers' opinions on the relationship between Bad Smells and faults. We will conduct an online survey of developers and researchers. Respondents will be asked to indicate how they think each Bad Smell relates to faults. The results of this study will complement the results of the first study to present a substantive analysis of the impact of Bad Smells.

3.2 Targeting Specific Bad Smells

Because of limited time and resources, this project can not investigate in detail all 22 Bad Smells [5]. Hence prioritizing Bad Smells and selecting a target set of Bad Smells is important. We use the following criteria to select our targeted Bad Smells.

1. Bad Smells which have attracted the least research attention in previous studies.
2. Bad Smells which are relatively easy to identify using static source code analysis techniques.

On this basis, five Bad Smells have been selected. They are the Data Clumps, Switch Statements, Speculative Generality, Message Chains, and Middle Man.

3.3 Formal Definition of Bad Smells

To define what structures in source code indicate Bad Smells is a precondition to using static source code analysis techniques in identifying Bad Smells. We define our target set Bad Smells using a pattern based approach. Each Bad Smell is defined as a set of specific source code patterns.

3.4 Automatic Tools

In order to investigate the relationship between Bad Smells and faults, we need to analyse a large amount of source code. To do this effectively, we are building a tool to assist our analysis. This tool will have two main functions: Capturing data from open source repositories, and identifying Bad Smells and faults from source code. This tool is based on approaches described by Zimmermann et al. [13], Fischer et al. [4], and Counsell et al.[2].

4. ISSUES AND PITFALLS

There are several issues and pitfalls need to be handled in this project. Firstly, our Bad Smells definitions are based on our own interpretation of Fowler et al.'s [5] definitions of Bad Smells. To eliminate bias, we will use an expert panel [1] to validate our definitions. Secondly, we need to find out how accurate our tools are in identifying Bad Smells and faults. A pilot study using a small size of Eclipse data will be conducted to test its accuracy.

5. REFERENCES

[1] Beecham, S., Hall, T., Britton, C., Cottee, M. and Rainer, A. 2003. Validating a Requirements Process Improvement Model, Technical Report 373, University of Hertfordshire.

[2] Counsell, S., Hassoun, Y., Johnson, R., Mannock, K. and Mendes, E. 2003. Trends in Java code changes: the key to identification of refactorings? Proceedings of the 2nd international conference on Principles and practice of programming in Java, Kilkenny City, Ireland.

[3] Fenton, N.E. and Pfleeger, S.L. Software Metrics: A Rigorous & Practical Approach. PWS Publishing Company, Boston, 1997.

[4] Fischer, M., Pinzger, M. and Gall, H., Populating a Release History Database from version control and bug tracking systems. in Software Maintenance, 2003. ICSM 2003. Proceedings. International Conference on, (2003), 23-32.

[5] Fowler, M., Beck, K., Brant, J., Opdyke, W. and Roberts, D. Refactoring: Improving the Design of Existing Code. Addison Wesley, 1999.

[6] Kitchenham, B. 2004. Procedures for Performing Systematic Reviews,TR/SE-0401, Keele University and National ICT Australia Ltd, 1-28.

[7] Mens, T. and Tourwe, T. 2004. A survey of software refactoring. Software Engineering, IEEE Transactions on, 30 (2). 126-139.

[8] Monden, A., Nakae, D., Kamiya, T., Sato, S. and Matsumoto, K., Software quality analysis by code clones in industrial legacy software. in Software Metrics, 2002. Proceedings. Eighth IEEE Symposium on, (2002), 87-94.

[9] Pfleeger, S.L. 2002. What software engineering can learn from soccer. Software, IEEE, 19 (6). 64-65.

[10] Shatnawi, R. and Li, W., An Investigation of Bad Smells in Object-Oriented Design. in Information Technology: New Generations, 2006. ITNG 2006. Third International Conference on (2006), 161-165.

[11] Woodall, P. and Brereton, P. 2006. Conducting a Systematic Literature Review from the Perspective of a Ph.D. Researcher 10th International Conference on Evaluation and Assessment in Software Engineering (EASE), Keele University, UK.

[12] Zhang, M., Hall, T., Wernick, P. and Baddoo, N. 2008. Code Bad Smells: A Review of Current Knowledge, Techical Report 468, STRI, University of Hertfordshire, Hatfield.

[13] Zimmermann, T., Premraj, R. and Zeller, A., Predicting Defects for Eclipse. in Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on, (2007), 9-9.