DT : a detection tool to automatically detect code smell in software project

Xinghua Liu[1,a] and Cheng Zhang[2,b]
[1]School of Computer Science and Technology，Anhui University, China
[2]School of Computer Science and Technology，Anhui University, China
[a]xinghua.liu@ahu.edu.cn，[b]cheng.zhang@ahu.edu.cn

Keywords: code smell; detection; detection tool; metric; threshold

Abstract . *Context:* Code smell can make the decline of code quality. *Code smell* is not a bug, and also can't make system to run exceptionally. It just can make some difficulties for software developers to understand and maintain the source code of projects, and then cause unnecessary maintenance costs. *Objective:* We try to more accurately detect code smell. *Method:* We put forward our smell detection tool: *DT* for short. We use DT to detect eleven code smells through detecting two kinds of projects: lab project, industrial project. *Result:* We get good results by using our Smell Detection Tool (DT), comparing with some famous detection tools: Checkstyle, PMD, JDeodorant and iPlasma. *Conclusion:* Our method Smell Detection Tool (DT) can be used to detect 11 kinds of code smell, In the future, we will go on detecting more code smells that can't be detected, and then do a survey about code smell among the software developers and maintainers.

1.Introduction

Code smell is an irrationality of code of software projects. They impede the evolution of code, so that they will result in an aftereffect that software developers pay more costs on maintenance.

For detecting code smell, researchers develop detection tool to detect code smell automatically in source code of project. Among these detection tools, there are some famous detection tools: Checkstyle[1], Infusion[2], iPlasma [1], PMD [2], JDeodorant [3] InFusion, Stench Blossom and so on. Most detection tools can detect three or four code smells.

The other parts of paper are as follows: Section 2 describes the related work. The Detection Theory is informed in Section 3. Experiment & Discussion is shown in Section 4. Section 5 comes to a conclusion and future work.

2.Related work

Fowler et al. [4] firstly put forward a concept *Code Smell*, which can result in some problems of maintainability. Dag Sjoberg et al. [5] mention the questions of maintenance, if we leave code smell aside, it causes a vexatious loss to the quality of code. Code smell is involved with project code. What's more, the size of industrial project is so enormous. This is not practical to find code smell by hand in project. Therefore code smell detection tools arise gradually.

Researchers gradually put forward some detection tools. The first detection tool is released by Eva van Emden et al. [6]. There are some famous detection tools: *Checkstyle, iPlasma, PMD, JDeodorant* and so on.

Marinescu, Cristina, et al [7] also put forward a detection tool *iPlasma*. *iPlasma* is a sub version of *InFusion* and detect 4 code smells, this tool has lower detection precision. Xinghua et al.[8] also

[1] http://checkstyle.sourceforge.net.

[2] http://www.intooitus.com/inFusion. html

carry out a systematic literature review of code smell. They introduce different detection tools of code smell. They make a comparison among these tools.

We put forward our own code smell detection tool ：*DT*. What's more, we want to more precisely detect these code smells by using our own tool.

3.Detection Theory

3.1Research Questions

Question 1: Do we get higher detection precision than other well-known detection tools?

In the past two decades, lots of detection tools had been proposed. They can support to detect several code smells, but the precision is different among these detection tools. Some detection tool has lower precision.

3.2 Detection Thoughts

We summarize and analyze these detection achievements of other research, we put forward our own detection thought. Through our thought, we put forward our own detection tool named ***DT***.

In our detection tool, we use one main detection thoughts: ***Abstract Syntax Tree (AST)*** .

3.2.1 Abstract Syntax Tree

Abstract Syntax Tree (AST) uses a tree structure to represent the abstract syntax of source code. Each node of the tree corresponds to a structures in source code. The main work of AST is to decompose the sentences of projects.

3.3 Detection Tool of Code Smell

In the past decades, researchers develop a lot of detection tools to detect code smell. There are appearance of several famous detection tools: Checkstyle , JDeodorant, PMD, InFusion, iPlasma and Stench Blossom. We informed these famous detection tool for two reasons: first one is that we want to introduce existing results of detection; the second one is that we will compare our detection tool **DT** with these detection tools in posteriori experiment.

3.4 Detection of Code Smell

One purpose of code smell research is to reduce maintenance costs. If we detect the code smell in project by hand, it's not still reducing the cost, instead increasing the burden of work. So the code smell researchers develop automatic detection tool to detect code smell. The purpose of this section is that we introduce the detection of code smell. Through this section, we will understand the detection of code smell as a whole. Meanwhile, we through analysis of table 3 to choose code smells that we need to detect in projects.

Large Class There are 15 detection tools supporting the detection of Large Class in table 3. Large Class is one class that burdens too much responsibility. During the detection, detection tools often use the Line of Code (LOC) as a metric to measure a project whether it has *Large Class* or not. In the same way, long method also take responsibility that other methods should take. Detection tool can use the LOC or other same metrics to judge whether a method has long method or not. Because of the simple metric, so there are a lot of Detection tool can detect Large Class and Long method.

3.5 Detection Metrics

In the former section, we discuss *Detection Thoughts* in macroscopic angle. In this section, we discuss the specific detection metrics according to each code smell. We introduce our detection tool *DT* by detecting code smells *Large Class* .

3.5.1 Large class

Large Class is one class that burdens too much responsibility. For detecting Large Class, we consider two elements: File Length (FL) and percentage (P).

File Length (FL) is the number of code lines per file; Proportion (P) is the percentage that the numbers of code lines of every file take up the total numbers of code lines of project.

We think it is inadequate that detection tool *Checkstyle* only consider the single element: File Length(FL). Because we may encounter some extreme situations. For instance, the file has too much code lines , but it only takes up a very small proportion, so it is not a large class. Secondly,

we can't only think about the percentage singly too, for example, if a small project only have two files, the percentage is larger than 20%, but the file length is too small, we can't set it as a larger class.

Table 1: Weight Distribution

Length	weight	Percentage	weight
< 500	0.4	20%	0.3
>200	0.3	10%	0.1

As the Definition between File Length and weight(Table 1), when the FL is larger than 500, we give the file 0.4 weight; when the FL is larger than 200 and smaller than 500, we give the file 0.3 weight, other situations aren't be considered.

As the Definition between Percentage and weight(Table 1), when the P is larger than 20%, we give the file 0.3 weight; when the P is larger than 10% and smaller than 20%, we give the file 0.1 weight, other situations aren't be considered.

We combine weight of File Length(FL) and weight of percentage(P), and then produce final weight value:

weight of FL + weight of P = total weight

Summing up the above, when file length is greater than 500 , and the percentage is greater than 20%;

when file length is greater than 500, and the percentage is greater than 10%;

when file length is greater than 200. and the percentage is greater than 20%;

If we encounter the three above situations, we can judge the class is a large class.

4 .Experiment & Discussion

4.1 Experiment system

We carry out our comprehensive experiment on t : a research lab project. In table 2, the lab project *PLOW_Code_Smells* has 16,627 LOC. This project has 43 java files. Although it is a small project, it has some classical code smells, we can get more deeply understanding of code smell through carrying out experiment on this research project. Each detection tool can support to detect code smell of lab project. We will carry out comparison among these tools.

Table 2: Weight Distribution

System	Project Name	Project type	LOC
System A	PLOW Code Smells	Lab Project	16 K LOC

4.2 Large Class

We carry out a comparison with detection tools: Checkstyle. Checkstyle support to detect Large Class. Checkstyle use a XML file to set the detection threshold.

In System A, when Checkstyle set the detection threshold as 500 LOC, we can get 9 classes that have the inclination of Large Class in table 3. We can see the two Large Class is DateAxis.java and AbstractCategoryItemRender.java. We can find that DateAxis.java has 1,872 LOC, it accounts for 11.25% size of project. AbstractCategoryItemRender.java has 1,843 LOC, it accounts for 11.07% size of project. The two class is actual Large Class.

If we change the threshold LOC from 500 to 1000 in table 3. Checkstyle and our detection tool have the same detection result: 2 Large Class. The two classes are classes that are mentioned before: DateAxis.java and AbstractCategoryItemRender.java.

Table 3: Large Class

Detection Tool	Threshold	
	500	1000
DT	2	9
Checkstyle	9	2

5.Conclusion and future work

Our detection tool *DT* can get better results and precision than checkstyle, PMD and iPlasma in the detection of code smell.

In the future, we should make our detection tool to detect these code smells that have been less attention to, specially these code smells that no detection tool can detect; secondly, we should have more communications with industrial field, and doing a survey about code smell among the software developers and maintainers.

6.Acknowledgment

This work is supported by National Natural Science Foundation of China(NO. 61402007) and the natural science foundation of Anhui Province (No. 1408085QF108).

7. References

[1] C. Marinescu, R. Marinescu, P. F. Mihancea, and R. Wettel. iplasma: An integrated platform for quality assessment of object-oriented design. In In ICSM (Industrial and Tool Volume. Citeseer, 2005.
[2] S. Slinger. Code smell detection in eclipse. Delft University of Technology, 2005.
[3] N. Tsantalis, T. Chaikalis, and A. Chatzigeorgiou. Jdeodorant: Identification and removal of type-checking bad smells. In Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on, pages 329–331. IEEE, 2008.
[4] M. Fowler. Refactoring: improving the design of existing code. Pearson Education India, 1999.
[5] D. Sjoberg, A. Yamashita, B. C. D. Anda, A. Mockus, T. Dyba, et al. Quantifying the effect of code smells on maintenance effort. Software Engineering, IEEE Transactions on, 39(8):1144–1156, 2013.
[6] E. Van Emden and L. Moonen. Java quality assurance by detecting code smells. In Reverse Engineering, 2002. Proceedings. Ninth Working Conference on, pages 97–106. IEEE, 2002.
[7] D. Sjoberg, A. Yamashita, B. C. D. Anda, A. Mockus, T. Dyba, et al. Quantifying the effect of code smells on maintenance effort. Software Engineering, IEEE Transactions on, 39(8):1144–1156, 2013.
[8] C. Z. Xinghua Liu. the detection of code smell on software development: a mapping study. Submitted.