Advances in Engineering, volume 126

5th International Conference on Machinery, Materials and Computing Technology (ICMMCT 2017)

The detection of code smell on software development: a mapping study

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; experiment study; metric; detection; detection tool

Abstract. *Context:* Although Code Smell can't cause problems with execution of project code, Code Smell can cause some potential problems of maintainability and understandability on the software projects. Meanwhile, as for the research of code smell, current research results pay attention to only several specific Code Smells, and then don't have a comprehensive detection on Code Smell. *Objective:* To investigate what the objective of Code Smell study is, and to find what kinds of code smells could the detection tools of code smell detect. *Methods:* According to the Guidelines of Kithenham, we carry out a mapping study about 22 code smells, searching the relevant papers till 2015. *Results:* Through the process of mapping study, 286 papers are finally included and then classified into our data records. *Conclusion:* Referring to detection tools, firstly they only take notice of several specific Code Smells, because these code smells can be easily measured in term of quantification. Secondly, experiment systems of these papers are almost lab projects and industrial open source not the industrial closed source projects. Thirdly, the size of most detected lab projects are under 30 KLOC. In the future, we will focus efforts on detection of Code Smells that can't be easily detected, what's more, we will put our studies under a comprehensive environment, using three types of project: lab project, open source industrial project and closed source industrial project.

Introduction

Code Smells can cause the problems of maintainability and understandability on software projects. Code Smells are not bugs, just can make some difficulties for software developers to understand source code of project. Meanwhile these code smells could cause difficulties to refactor and upgrade source code of projects for software developers and maintainers.

Firstly, Fowler et al. [6] inform 22 kinds of code smell. They give the definitions of these code smells. Secondly, the researchers put forward a doubt that whether the software designers need to detect the code smell or not. Yuepu Guomet et al. [8] describe the Domain-Specific tailoring of Code Smell. They think the metrics of code smell is a little broad, and want to make that metrics of code smell is more appropriate to the specific field. Steffen M. Olbrich, et al. [17] discuss that whether all of code smell is harmful to software projects. They carry out a research whether a project of containing the God Class and Brain Class is under the poor efficiency.

To quickly find code smells, we must develop code smell detection tool to detect code smell automatically. In the two decades (2000-2015), a lot of detection tools are proposed to detect some kinds of code smell. Fontana F A et al. [5] inform a lot of code smell detection tools. There are free or commercial detection tools, for example, Checkstyle[1], Infusion[2], PMD [22], JDeodorant [23] , iPlasma [14] , DECOR [15] and so on. But we can find some points that the most of detection tools only can detect several specially appointed code smell.

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

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

Although code smell has been researched by academic circles through the unambiguous stages, but we still have doubt about some aspects. Firstly, what is the objective of code smell study? What's more, we still uncertain that whether detection tools can detect all code smells or not. Researchers prefer to carry out experiments on lab projects and open source projects. Their experiment study is lack of closed source projects. To explore and explain these questions, we carry out a mapping study about code smell.

The remainder of this paper is as follows: Section 2 introduce the related work. Section 3 is the methodology that we did our mapping study. Our analysis are presented in section 4. In section 5, we come to discuss about our presented research questions. Threats to validity is shown in section 6. Finally, we get our conclusion and our future work in section 7.

Related work

Code Smells are firstly described by Fowler et al. [6], they give the definitions of 22 code smells. Through the symptoms of code smell have been predefined by Fowler, Yamashita [26] study what results several code smells can cause together. They carry out an empirical study to test and verify the relationship among several code smells. AYuepu Guomet et al. [8] describe the Domain-Specific tailoring of Code Smell. They think the heuristics of code smell [6] is a little broad, they tailor the heuristics of domain-specific of code smell to make the heuristics to fit the specific domain.

These above research works mainly concern definitions of code smell. Code smell is an issue that is closely with industrial field, but there are less data to show clearly the attitude of industrial fields about code smell, do they know the term *Code Smell*? Do they need to solve code smell in the project? Aiko Yamashita, Leon Moonen [25] provide a point that we should think about code smell in the industrial view. And they do a survey among the software developers. They explore that view of industrial field to code smell.

Palomba, F et al. [18] provide a method *HIST*(Historical Information for Smell detection) to detect five code smells in the version control system. We know that with the increasing and changing requirements of industrial software, the version of projects can be continually released and the source code of projects can become more and more enormous. The former version maybe have the less quantity of code smell, the newest version can increase the numbers of code smell because of the release of new version of project. They only can detect five code smells. Meanwhile there is also a deficiency that they just carry out their experiment on open source projects : Apache Ant, Apache Tomcat and so on, they don't involve commercial closed source project. F. A. Fontana et al. [5] do comparisons among code smell detection tools. They introduce a lot of detection tools . For example, Checkstyle[3], Infusion[4], PMD [22], JDeodorant [23], iPlasma [14], DECOR [15] and so on. Meanwhile, they carry out studies of precision and efficiency among these detection tools. The research data shows that these detection tools only detect several code smells, most of them only can detect 3 or 4 code smells. These detection tools can together detect no more than twenty code smells. In addition, the detection precision among these detection tools varies enormously.

Zhang et al.[27] carry out a systematic literature review on code smell. They put forward a point : researchers have a preference on some specific code smells. Researchers pay more attention on code smell *Duplicated Code* and have less notice code smell *Message Chains*.

Although the above work had gotten some achievement, we think that the research of code smell should pay more attention on industrial field when we carry out study in lab environment. Because code smell is an issue that closely related with industrial field. We carry out our study on lab projects, industrial open source projects and industrial closed source projects, we will have

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

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

comprehensive understanding on code smell. What's more, there is still a lot of code smells that can't be detected by detection tools.

Methodology

Research Questions.

At the beginning of mapping study, we propose three research questions of mapping study. With these research questions, we carry out mapping study and find more useful information from the mapping study process.

Question 1: What is the objective of code smell study in empirical software engineering paper?

With different mentions of study, researchers carry out empirical studies on code smell.

Question 2: Have all code smells been detected by detection Tools?

We know that Fowler defines 22 kinds of Code Smell. In recent years, Code Smell researchers are devoted to develop automatic detection tool to detect these code smells. One Detection tool is so easy to detect three or four code smells. Adding up detected code smells of all detection tools, we find that no more than twenty code smells can be detected. Researchers pay more attention on some specific code smells, some code smells are less concerned by researchers.

Question 3: Does the researchers have carried out adequate comprehensive experimental studies?

Code Smell is actual industrial problem. We not only want to know the work of research-based project, but also want to get the detection results on industrial studies. So we should carry an comprehensive experimental study on code smell. An comprehensive experimental study contains study of research lab projects, industrial open source projects and industrial closed source projects.

Search Process.

We carry out our research study through a mapping study. Kithenham et al. [10] give the guidelines to teach us that how to do a mapping study. The main process of mapping study is shown in Figure 1. We firstly carry three searches: *Electronic Search, Manual Search and Snowball Search* to search papers related to code smell. In every step, we make use of Inclusion and Exclusion to judge whether one searched paper should be included or not. We read every paper's Title, Abstract, Introduction and Conclusion to judge the correlation between the paper and code smell. After above three search steps, we use the *Quality Assessments* to further estimate a paper, if the paper get too lower assessment scores, it still will be excluded. In our *Quality Assessments*, the threshold of score is 2 points. After preceding steps, we arrange the paper to a particular form of data records. The form of data record (table 1) will be explained in detail in the following section.

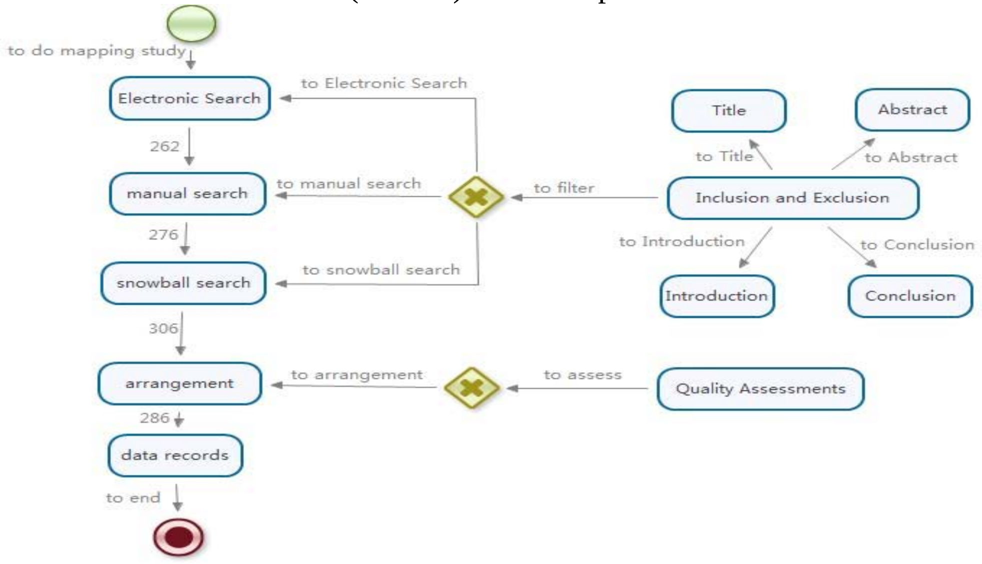

Figure 1: Process of Mapping Study

Firstly, we choose three databases to do our electronic search, and these Data-bases are as follows: IEEE X.plore, ACM Digital library, Citeseer in table 2. Why we choose the three databases to do our research? Because the three databases are available to all researchers and these researchers are much more familiar with the three databases. At last but not least, the three databases can stand for the direction of code smell.

Secondly, We carry out Manual Search through searching two Journals: *TOSEM (ACM Transactions on Software Engineering Methodology) , TSE (IEEE Transactions on Software Engineering)* in table 2.

Thirdly, Snowball Search is done through analyzing the reference of above searched papers, to find papers that are relevant to code smell. The purpose is to replenish the leaky papers that don't searched by above two steps. Through the third step - snowball search, we can be more comprehensive to carry out code smell research.

Table 1: the elements of record

data	explanation
Title	title of the paper
Authors	names of the paper
database	the database stores the PDF of the paper
Conference/Journal	Simply classify the type of the paper to two types: Conference and Journal
the shorter form of Conference/Journal	record the shorter form of the name of Conference/Journal
keywords	record the keywords searched the paper
the year of publication	the time the Conference/Journal publishes the paper
other information	record other extra information

Table 2: source of paper

source of paper	short name
IEEE X.plore	IEEE
ACM Digital library	ACM
Citeseer	Citeseer
Journal of ACM Transactions on Software Engineering Methodology	TOSEM
Journal of IEEE Transactions on Software Engineering	TSE

These keywords we use are to search the papers about code smell: *code smell and empirical* OR *code smell and experiment*

OR *code smell and case study* OR *code smell and case studies OR code smell and maintenance* OR *code smell and maintainability* OR *code smell detection tool* OR *code smell and quality metric.*

Inclusion and Exclusion Criteria.

Inclusion

A mapping study must be able to represent the direction of the research; and then the paper must include code smell and base on the empirical/experience/experiment study; in addition, if a study is published in journal or conference, it can be included.

Exclusion

A paper must be written using English not other language, otherwise it would be excluded. What's more, if a study is a slide or a report or a graduation thesis, it also will be excluded.

Quality Assessments.

As for Quality Assessments, we firstly look through the guideline [10], there is a summary about quality assessment. Meanwhile, we draw on the experience of questions that li et al.[12] puts forward in Quality Assessments. They use five questions to assess their searched papers. In addition, T. Dybå, T. Dingsøyr [2] also put forward their own quality assessment criteria, and give the code of points. From the foregoing contents, we choose 4 questions from the checklist of quality assessment. We also define the score for Yes (Y)=1, Partly (0.5), No (N)=0.

We sum up scores of 4 questions as the Quality Assessments scores. If a paper gets too low scores: the scores is less than 2 points, we will delete it from our mapping study. The 4 questions are as follows:

How well is data collection carried out?

-Yes: The study has collected ample data

-Partly: The study has collected partial data

-No: The study doesn't collect the data

How defensible the research design?

-Yes: The research design is very reasonable

-Partly: Although the research can solve the problems they put forward, the evidence and data is insufficiency

-No: The design is very disordered, and confused

How clearly is the research process established?

-Yes: The study clearly describes the research process

-Partly: The study describes the research process, while it loses some details

-No: The study doesn't describes the research process

How clear are the links between data , interpretation and conclusions?

-Yes: The study explicitly explains the links

-Partly: Although the study describes the links, it doesn't explain very clearly in some details

-No: The study doesn't explain the links

Outcome.

In Fig.1, we use these above keywords to electric search in three databases, meanwhile we exclude the searched papers according to preliminary inclusion and exclusion criteria. Thirdly, we look through these remained papers title, abstract, introduction and conclusion to filter papers again. In the electronic search, we find 262 papers about code smell.

As for manual search, we look through above Journals: *TOSEM*, *TSE* to find relative papers in Fig.1. Meanwhile, we also use the inclusion and exclusion criteria. When we finish the step, we also further to read these Journals title, abstract, introduction and conclusion to filter papers. 14 papers meet the requirements in manual search.

In snowball search, we look through the reference of papers that are searched by above two steps, we do the same criteria as snowball search. 30 papers are found in snowball search.

After the three search: electric search, manual search and snowball search, we preliminary find 306 papers about code smell. In addition, through fourth step Quality Assessments, the score of 20 papers is below 2 scores. So we delete the 20 papers from our data. we finally decrease our data records from 306 to 286 papers about Code Smell.

Tendency of Code Smell Research.

Through counting publication year, each year's publication histogram is shown in Fig.2. The term Code Smell firstly informed by Fowler, there are very little papers released about code smell between 2001 and 2003, fours years only have 9 papers about code smell. But after 2003, more and more researchers pay attention to code smell, the numbers of papers about code smell rise year by year until 2013.

Fig.2 shows the tendency of publication year, it witnesses that the tendency of code smell study become more and more popular. These researchers argue hundreds of thought and idea about code smell, they want to find specific and feasible thought to do more deep research in code smell field so that the number of paper shows an upward trends.

As to recent two years 2014 and 2015, the numbers seem descended. But there are still 8 (15%) papers of top level. The 8 papers are published by top level Conference and Journal that internationally accreditated. These are top level Conferences and Journals: TSE[5], ICSE[6] and TOSEM[7]. It illustrates that code smell researchers gradually don't pursue the number of paper, instead pursue high quality paper.

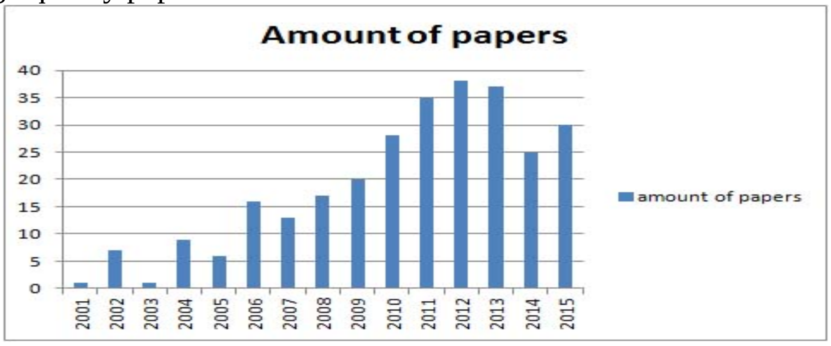

Figure 2: publication year

Research Hot Spots of Code Smell.

Fig.3 shows the percentage of keywords, these keywords are used in the papers of mapping study. These keywords not only can be used to search code smell relevant papers, but also can be used to analyze the research hot sports of Code Smell. In addition, " code smell " and " code smell detection tool " is almost involved by each paper, so put it aside.

Keyword "code smell and empirical" contains 37%, 26% are the " code smell and maintenance ", and " code smell and quality metric" accounts for 14%, other key words hold 23%. These percentages inform us where the research hot spots about code smell are. We firstly see that "code smell and empirical" accounts for the largest proportion. It means that Code Smell researchers want to do more research in empirical field [20], not only put forward theory about code smell. And as for " code smell and maintenance ", the definition of code smell is relative to maintenance, what's more, the industrial applications of code smell also focus on maintenance [21]. What's more, researchers use "code smell and quality metric " to find the detection metric of code smell.

In addition, these keywords that occupies less proportion are also important, these fields that researchers pay less attention to maybe a breakout for researchers to issue high quality papers. In the recent years, a lot of emerging technology hot spots is coming out or coming hot again. For example, machine learning, data mining, big data and so on. Fu S, Shen B. [7] use the thought of data mining to detect code smell. Although these new thoughts now only account for a small proportion , but we believe these maybe become new research hot spots in code smell field.

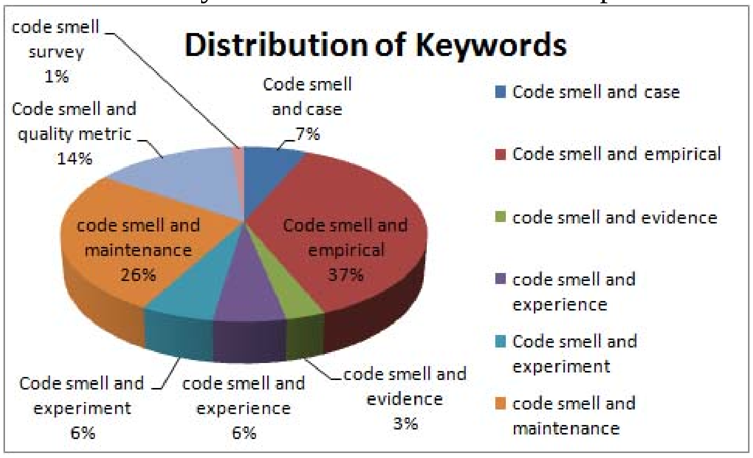

Figure 3: keywords

[5] http://www.computer.org/portal/web/tse/home

[6] http://www.icse-conferences.org/

[7] http://www.acm.org/pubs/tosem/

Analysis

Through above process of collecting and arranging data, we have got 286 data records about code smell papers. Before analysis, we should introduce the composition of records in table 1. Each paper has one record. Each record is combined by 8 elements. Of course, every data record of paper should have title, authors, Conference/Journal and publication year. We explain other elements---*database*: we record it to judge which database the paper locates in; the use *shorter form of Conference/Journal* to make quickly location of paper when we check and use the data records; *other information* is remained to record extra information.

Now we must analyze and find useful information from these records. This section is divided to four parts: Detection tool of code smell, Detection metric, Detection of code smell and Empirical Experiment.

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 in table 3.

Checkstyle is a well-known static code analysis tool, it can detect 4 code smells: Large Class, Long Method, Long Parameter List and Duplicated Code. Meanwhile the user of Checkstyle can operate XML file to modify parameters to detect code smell.

JDeodorant is mentioned by Tsantalis N et al. [23] JDeodorant is an eclipse plug-in. Through using the detection tool JDeodorant, code smell research field has produed lot of papers [4,3]. It can automatically detect 4 code smells : Feature Envy, God Class, Long Method and Switch Statement. What's more, it can achieve high detection accuracy. In addition, JDeodorant can achieve a good visualization of detection results. But at present it only can detect 4 code smells.

PMD is also a well-known detection tool and an Eclipse plug-in. PMD can detect 4 code smells: Large Class, Long Method, Long Parameter List and Duplicated Code. It can get good scores on the detection of duplicated code.

InFusion can detect more than 20 code smells and code defects. It can detect 7 code smells: Duplicated Code, Data Class, Data Clumps, Feature Envy, Large Class, Refused Bequest and Short gun Surgery. Meanwhile it's a charge software, so it has not been widely used.

iPlasma is a sub version of InFusion. What's more, it's free and available, It can detect 4 code smells: Duplicated Code, God Class (Large Class), Feature Envy and Refused Bequest and so on. But iPlasma has lower detection precision.

Stench Blossm is also an Eclipse plug-in, it can detect 6 code smells: Data Clumps, Feature Envy, Large Class, Long Method, Message Chains and Switch Statement. It use the style of petals to show the code smell, The strength of Code Smell is in proportion to size of Petal. In other words, if a file has more code smell, the petal is more bigger. But if you want to find out the location of the code smell, you should to read the source code by hand.

Detection Metrics.

In former section, we introduce detection tools. In this section, we introduce detection metrics integrating with section *Detection tools*. Code smell researchers put forward these detection metrics. They can transfer these detection metrics into programming of detection tools. When they use the detection tool to detect code smell in projects, they can find the imperfection of detection tools, these researchers will go to review these quality metrics. Through this process, they can make more accurate detection of code smell. In addition, these quality metrics are the core of detection tool.

1)LOC

If a class or method body has too much code, it violates the thought of Objected-oriented Programming. It seems that the class or method carries too much duty that other classes or methods should take. LOC is the number of code lines per file. Munro M J et al. [16] consider the metric LOC to assure the quality of code.

Detection tool *Checkstyle* and *PMD* adopt this metric : LOC. It just has one different place: they use the different threshold. As for detecting class, Checkstyle use threshold 2000 LOC, PMD use threshold 1000 LOC. As for detecting method, Checkstyle use threshold 150 LOC, PMD use threshold 100 LOC.

Table 3: Comparison of Detection Tool

Detection Tool	Advantage	Disadvantage	Numbers of Code Smell	Code Smell
Checkstyle	a static code analysis tool, Detection threshold can be operated in a XML fille of Checkstyle	It can't filter the detection results, it has lower detection precision	4	Large Class Long Method Long Parameter List Duplicated Code
PMD	An Eclipse plug-in. It's so easy to operate, meanwhile it has high precison to detect Duplicated Code	It has lower precison to detect other Code Smell except Duplicated Code.	4	Large Class Long Method Long Parameter List Duplicated Code
Jdeodorant	An Eclipse plug-in. It has higher detection precison to 4 Code Smells	It only can detect 4 Java Code Smells at present.	4	Feature Envy God Class Long Method Switch Statement (Type Checking)
InFusion	It can detect more than 20 Code Smells and Code defects, It can detect Java/c/c++ project.	It's a business software, so it' charged.	7	Duplicated Code Data Class Data Clumps Feature Envy Large Class Refused Bequest Short gun Surgery
iPlasma	the sub version of InFusin	It has lower precison	4	Duplicated Code God Class Feature Envy Refused Bequest
Stench Blossom	An Eclipse plug-in. It can make a better visualization of Detection	If you want to find out these detected code smell, you should look through source code by hand	6	Data Clumps Feature Envy Large Class Long Method Message Chains Switch Statement

2) Abstract Syntax Tree

Lanza and Marinescu et .al [11] adopt the metric of node. Firstly they introduce *Access to Foreign Data (ATFD)* to measure that the method of one class directly use the attributes of other classes. Secondly, they introduce *Locality of Attribute Accesses (LAA)* to measure that whether the method of one class calls too many attributes of other classes than its own class.

In other words, their main thought is to use the Abstract syntax tree (AST) to express the source code of project. AST transfer source code to nodes of syntax. For instances, *ClassInstanceCreation* which is an AST node as one of detection metric. ClassInstanceCreation are class instances that are created in the possess of run of project.

MethodInvocation is also a node of AST. MethodInvocation is that a method call other methods. Detection tools *iPlasma* and *inFusion* use former metrics.

3) Parameter List

If a file has a lot of parameters, it's very difficult to understand and maintain the code. Meanwhile this file has the inclination of process-oriented, *Object Oriented Programming (OOP)* should capsulate these too many parameters.

These researchers consider one element: the numbers of parameters of method (NPM). Danphitsanuphan et al. [1] use NPM thought, they give a threshold 5. Meanwhile these researchers detect long parameter list through detecting through thought of Abstract Syntax Tree (AST). Counting the numbers of the SingleVariableDeclaration (LPL) which is one node of Abstract Syntax Tree (AST), they can judge that whether a method has code smell long parameter list or not. This simplified formula is as follow:

IF (NPM > 5) Long Parameter List;

PMD and *Checkstyle* use the detection metrics, they just use the different threshold.

4) Block of Duplicated Code

Duplicated Code is that the sample codes appear more than one place. These researchers consider the length of block of Duplicated Code or the distance of two Duplicated Codes [13]. Detection tools *iPlasma* and *inFusion* use former two detection metrics.

Detection tool *checkstyle*, it use the detection metric: the block of Duplicated Code, and it uses a sample realization: it just accounts the size of the block of Duplicated Code. This threshold of size is 12.

In addition, we should introduce one point that if the two compared codes have a little difference, it doesn't matter. These metrics are not using one char as a component unit, instead use a code string as a component unit if the two compared code line have 80% similarity, they can judge they are duplicated. For example, a very sample Java code : int a = 1; if we just change the variable name a to variable name b. Although the two codes is not same, two codes are duplicated.

Of course, they can't think the code block has code smell only by one or two lines of duplicated code. They just use former detection metrics to give comprehensive judge the two compared code block.

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. In table 4, each row records the relevant information of one code smell. For example, as for the row *Duplicated Code*, it has 10 detection tools can detect it, the 10 detection tools are listed in the third column.

Table 4 not only records the detection work of several well-known Detection tools: Checkstyle, JDeodorant, PMD, inFusion and iPlasma, but also records a lot of detection tools that other researchers developed and mentioned in papers. From table 4, we find that most code smell researchers pay attention to Code Smells: long method, large class, Data Class, Duplicated Code and Feature envy. These code smells are more easily measured by quantitative metrics. For other code smells in table 4 that detection tool seldom can detect, even 7 code smells can't be detected by any one detection tool in table 4, the most important reason is that it has difficulties to convert the definition of code smell into detection metrics.

Table 4: Code Smell

Name of Code Smell	Numbers of Detection tool	Detection Tool
Duplicated Code	10	Checkstyle, inFusion, JCCD, PMD, InFusion, FP-Growth, JBuilder, CloneDR, SHINOBI, CloneDetective
Long Method	14	Checkstyle, JDeodorant, PMD, InsRefactor, J48, Random Forest, Na´l ve Bayes, Rip, SMO, LibSVM, StenchBlossom, iPlasma, DECOR, iPlasma
Large Class	15	Checkstyle, inFusion, JDeodorant, PMD, InsRefactor, J48, Random Forest, Na´l ve Bayes JRip, SMO, LibSVM, InFusion, iPlasma, DECOR, Stench Blossom, iPlasma, CodeVizard
Long Parameter List	5	Checkstyle, PMD, InsRefactor, iPlasma, DECOR
Shotgun Surgery	2	InFusion, iPlasma
Feature Envy	10	J48, Random Forest, Na´l ve Bayes, JRip, SMO, LibSVM, inFusion, JDeodorant, iPlasma, Stench Blossom
Data Clumps	1	Stench Blossom
Primitive Obsession	1	JSmell
Switch Statements	3	InsRefactor, Stench Blossom, JDeodorant
Parallel Inheritance Hierarchies	1	Depth Inheritance Hierarchy (DIT)
Lazy Class	1	PMD
Speculative Generality	4	JSmell , DECOR, iPlasma, PMD
Message Chains	2	Stench Blossom, DECOR
Data Class	12	J48, Random Forest, Na´l ve Bayes, JRip, SMO, LibSVM, InsRefactor, JSmell, InFusion, iPlasma, Stench Blossom
Refused Bequest	4	InFusion, iPlasma, DECOR, PMD
Temporary Field, Middle Man, Inappropriate Intimacy, Divergent Change, Comments, Incomplete Library Class, Alternative Classes with Different Interfaces	0	No One Tool can detect them

Large Class & Long Method: There are 15 detection tools supporting the detection of Large Class in table 4. 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.

Duplicated Code: Duplicated Code is the sample codes appear more than one place. It can be detected by 10 detection tools in table 4. It is research hot spot. It's not only studied in Code Smell field, but also researched by other researched areas, Duplicated Code is called Code Clone in these areas. Duplicated Code is not easy to be detected by detected like *Large class* any more. The detection metric is that we count the similar lines of code block among software project. If the lines of code block are large than a threshold that is given by experts, we judge that the two compared code blocks are duplicated code.

Feature Envy: A function is more inclined to other class than the class that the function belongs to. In other words, the method calls too much fields and methods of other classes than own class. Although it needs to detect the Method Call and use the thought Abstract Syntax Tree (AST), but it has practical application to eliminate the phenomenon that is high coupling and low cohesion. So that there are 10 detection tools to support detection.

Lazy Class: Lazy Class is only detected by one detection tool. Lazy class is a class which has too low value of existence. Although the definition of *Lazy class* is more easier to be followed, there has more difficulty to quantify the definition to detection metrics. So that there has little tool to support detection of Lazy Class. Even there are 7 Code Smells that can't be detected by any one tool in table 4.

Empirical Experiment.

We search 86 experiment papers from 286 records of data and then classify them to 60 records of experiment. The search standard is that if one paper use *software projects* to detect and analyze Code Smell, we collect it to the records of experiment. Each experiment record is combined by the those elements: project, participants of project, LOC, experimental objective, related tool and result.

1) *Type of Project*

Why we choose *type of project* as an considerable element? Because the purpose of research is to detect and find code smell in project. In *project* element, we record that what projects the paper used and these projects are lab project (type A), industrial projects in table 5. Industrial projects are divided to two types: industrial open source projects (type B) and industrial closed source project (type C) in table 5.

Lab projects(type A) are smaller research projects. These projects only have one aim to test detection ability of these detection tools without any other function. Type A account for 13% projects.

Industrial open source projects (type B) have been involved 151 times (64%) in the 86 experiment papers, it accounts for the largest proportion. These projects are open source, so that each developers can download them easily from Internet. But most industrial projects are some software platform, for example eclipse, Aapche Ant, Log4J, ArgoUML, JFreeChart and so on. The type of project can't stand for the whole industrial field.

In addition, there are 55 industrial closed source projects (type C). These projects only account for 23%. These projects are developed by software companies. These projects are private, these source code can't be opened to everyone. But these projects are true business projects, for example , legacy system, E-business Website Resource Management System and so on. After detection of lab project and industrial open source project, detection of industrial closed source project can give us a more comprehensive view to Code Smell.

2) LOC

It is not enough that we only consider *project*, Kilo Line of Code, for short KLOC is also an important criterion. As we all known, some projects are enormous, specially industrial projects. The bigger the project, the more difficult Code Smell is difficult to detect. So that we exploit KLOC to further analyze the empirical experiment.

Analyzing table 6, we can find that size of most of projects is under 10 KLOC. Meanwhile we find that type A only appear in 0~10 KLOC and

10~30 KLOC projects in the third column *project type*. Type A is lab project , as we mention in part **Type of Project**. The size of lab project is under 30 KLOC. Lab project seems too small to illustrate questions.

Some industrial projects involved in 0~10 KLOC and 10~30 KLOC projects. Because writers of these papers choose an assembly of a larger projects, for example, Hall, Tracy, et al. [9] use 6 assemblies of Apache Common : Apache Common Codec (5KLOC) , Apache Common DBCP (10KLOC), Apache Common DbUtils (3KLOC), Apache Common IO (12KLOC) , Apache Common Logging (7 KLOC), Apache Common Net (33KLOC). Although each assembly is small, but as an entirety, Apache Common has at least 70 KLOC. It is so huge open source project. Above all, we roughly choose 30 KLOC as a cut-off point of lab project and industrial project. It can guide researchers to choose appropriate size of research project.

Table 5: Project Type

project	Lab projects	Industrial open source projects	industrial closed source projects
type	type A	type B	type C
count	31(13%)	151(64%)	54(23%)

Table 6: KLOC

KLOC	number of project	project type
0~10	65	type A, type B, type C
10~30	23	type A , type B, type C
30~50	22	type B, type C
>50	44	type B, type C

3) *Participants of Experiment*

In our paper, we also have an interest on *participants of experiment*. We want to know participants: writer of paper, industrial developer in these experiment papers. Doing these experiments, some participants is only the writers of paper (participant A), some participants may be familiar with some software developers, they can invite these software developers together to do experiment and analyze symptoms of code smell (participant B) to enhance the practicality of their work.

In table 7, we find that only 18 experiments that involved software developers. It only accounts for 30%. The data illustrates that although code smell is both research and industrial problem, they don't together put attention on code smell.

Table 7: Extensional Research Questions

No.	Research Question(s)
DT1 [21]	How the relation between code smell and maintenance through detecting an industrial project?
DT2 [24]	What code smells are we going to detect? How are we going to detect these smells?
DT3 [19]	Does their proposed detection tool have a good detection result in experiment?
DT4 [18]	To test the detection precision of a new detection tool that they put forward.
DT5 [5]	How the accurate of the different detection tools is?

Discussion

Through mapping study about code smell, we have analyzed the data records in Section **Analysis**. Now we give a specific discussion of code smell with *Question 1- Question 3*. In addition, We want to further expound our Research questions through extensional questions in table 7. We choose some questions of detection tools as ***DT1 to DT5*** in table 7.

Table 9: purposes of experiment paper

purpose	Maintainability	Evaluating detection tool	Definition of code smell
count	11(18%)	27(45%)	22(37%)

Question 1: What is the objective of code smell study in empirical software engineering paper?

We arrange the purposes of 60 experiment records in table 9. We roughly divide the aims of these experiments to three purposes: Maintainability, Evaluating detection tool and Definition of code smell.

11(18%) experiment papers are relevant to *Maintainability*. Code Smell doesn't bring any bugs to the execution of software project. *DT1* discusses the relationship between code smell and maintenance in table 7. Code Smell can cause some difficulty for software developers and maintainers to understand source code of project. From this, Code Smell can give rise to the increased maintenance costs. Research on maintainability is a practical research, it can produce practical beneficial result: decreasing the maintenance costs.

27(45%) experiment papers are to carry out their experiment to *evaluate their detection tools*. In table 7, *DT2* gives us an example what code smell should we detect? and how to detect? The answer is that we take advantage of detection tool to detect code smell. After that, *DT3* carry out extensional research question: Does their proposed detection tool have a good detection result in experiment? They do experiment on project to check precision of their own detection tool. Meanwhile, they not only put forward detection tools, but also give the quality metrics. Quality metrics is the core thought of detection tools. The true purpose of evaluate detection tools is to evaluate quality metrics.

37% experiment papers pay attention to *Definition of code smell*. AYuepu Guomet et al. [8] describe the Domain-Specific tailoring of Code Smell. They think the heuristics of code smell [6] is a little broad, they want to carry out a study to give more accurate definition of code smell. Meanwhile, Yamashita [25] carry out a survey to further investigate this question in industrial field. Through realizing detection tool, they test their own detection metrics. After then, they put forward more efficient quality metrics to detect code smell.

As we analyze the objective of code smell study in empirical paper. We find three objective: Maintainability, Evaluating detection tool and Definition of code smell. *Maintainability* is an objective that closely relative with actual field. *Evaluating detection tool* and *Definition of code smell* is two academic research problems. Academic research still accounts for the larger proportion.

Question 2: Have all code smells been detected by detection tools?

Firstly, some Researchers put forward their own detection tool to detect code smell, these detection tool are new and the accuracy is unsettled, *DT4* has involved one new detection tools 'HIST', they show their excellent detection results of their own tool, but their detection tool doesn't be used widely to detection kinds of projects.

Secondly, some detection tools has withstand rigorous testing in the decades. They are broadly used in Detection work. These well-known detection tools are exhibited in table 3: Checkstyle , JDeodorant, PMD, InFusion, iPlasma and Stench Blossom. *DT5* carries out a comparison between those detection tools. Code Smell researchers are devoted to develop automatic detection tool. Tools are so easy to detect three or four code smells. It has much difficulties to detect Code Smell as much as possible.

In table 3, **Checkstyle** that is a static code analysis tool can detect four code smells: *Large Class, Long Method, Long Parameter List and Duplicated Code.* **PMD** only can detect the four same code smells too. As for **JDeodorant**, although it has higher detection precision, it only can detect four code smells: Feature Envy, God Class (Large class), Long Method and Switch Statement (Type Checking). Although **Infusion** can detect more than 20 code smells and code defects, it can detect 7 code smells: Duplicated Code, Data Class, Data Clumps, Feature Envy, Large Class, Refused Bequest, Short gun Surgery. **iPlasma** is sub version of *Infusion*, it can detect four code smells:

Duplicated Code, God Class (Large Class), Feature Envy and Refused Bequest. In addition, **Stench Blossom** can detect six code smells: Data Clumps, Feature Envy, Large Class, Long Method, Message Chains, Switch Statement.

Above all, most of these famous detection tools only can detect four code smells. Meanwhile, we can find the six famous detection tools can detect 12 code smells: Large Class, Long Method, Long Parameter List, Duplicated Code, Feature Envy, Switch Statement, Data Class, Data Clumps, Refused Bequest, Short gun Surgery, Message Chains. 9 code smells is not still detected by these famous detection tools. For example, Temporary Field, Middle Man and so on. There is no tool to support to detect them.

In Table 4, we can clearly see that Duplicated Code, Long Method, Large Class, Feature Envy and Data Class have higher attention by researchers. Specially, there are 15 detection tools to support to detect *Large Class*. In addition, we also should concern a phenomenon that some code smell don't give rise to enough attention. Some code smell aren't detected by any one detection Tool, for example, *Temporary Field, Middle Man, Inappropriate Intimacy, Divergent Change, Comments, Incomplete Library Class, Alternative Classes with Different Interfaces* and so on.

Question 3: Does the researchers have carried out adequate comprehensive experimental studies?

We find that the projects of these studies are lab projects and industrial open source projects and closed source project from Table 5. lab project and open source projects account for 77%. But commercial closed source projects only account for 23%. It only occupies a smaller proportion in experimental studies.

Although the open source projects belong to industrial projects, for example Log4J, Apache Commons and so on, they are not strictly commercial projects. Code Smell is closely relative with industrial field. So if we want to get comprehensive experimental study, we should carry out our study on the three types of project: lab projects, open source projects and closed source projects, In addition, we find out that lab projects have a threshold of KLOC: 30 KLOC. It can guide researchers to choose appropriate size of research project.

Threats to Validity

Search process.

Our research are executed using relative keywords on three digital databases: IEEE Xplore, ACM Digital library, Citeseer. Although we search relative papers only on three databases, maybe some papers about code smell can't be searched, the three databases store the vast majority of code smell papers and can represent the research direction of code smell.

Selection of the Studies.

When we filter the papers through reading paper's title, abstract, keywords, introduction and conclusion to judge whether the paper is relative to code smell or not. But during the period, we maybe record duplicated papers. To solve the duplicated problem, we review the list of paper repeatedly and carefully, and at the end the result will be checked again by our supervisor.

Quality Assessments.

The Quality Assessments (QA) contain some subjective ingredients. But these questions of quality assessments reference checklist of QA given by Staffs Keele [10]. This checklist has been referenced by most of researchers. These questions are also representative and comprehensive. In addition, evaluation standards is reasonable. The subjective effects can be eliminated.

Conclusion & future work

The main work of this paper is to carry out a mapping study about Code Smell. We classify and analyze 286 papers about code smell, firstly the amounts of papers are roughly increased per annum. The field of code smell has lots of research hot spots, for example, “ code smell and empirical ” and “ code smell and maintenance ”.

The answers to the three research questions are as follows: One purpose of these empirical study is to decrease the maintenance costs in software development. This purpose is relevant to industrial practical problem. Meanwhile, there are also two academic purposes: Evaluating detection tool and Definition of code smell. Half purpose of experiment is to evaluate detection tool, it indirectly witnesses that to evaluate detection tool is a main academic trend for empirical code smell research. In fact, through evaluating detection tool, they also want to find efficient detection metrics to detect accurately code smell.

There are famous detection tools: *Checkstyle, JDeodorant, PMD, InFusion, iPlasma and Stench Blossom*. most of them can detect four code smells. The famous detection tools totally can detect 12 code smells: *Large Class, Long Method, Long Parameter List, Duplicated Code, Feature Envy, Switch Statement, Data Class, Data Clumps, Refused Bequest, Shot gun Surgery, Message Chains*. All detection tools only can detect 15 code smells. We find that not all code smell can be detected by detection tool, In addition, not only famous detection tools, but also other common detection tools can't support to detect some code smell. For example, *Temporary Field, Middle Man* and so on.

In the code smell research field, there are lots of research using lab projects and industrial open source projects, but lack of research about industrial closed source projects. If we want to get comprehensive view on code smells, we should carry out our study on three types of projects: *lab projects, open source industrial projects and closed source industrial projects*. Meanwhile size of these lab projects are under 30 KLOC. The threshold can give researcher a standard to choose size of research lab project.

In the future, we have lots of work to do. Firstly, we should develop a tool to detect these code smells that have been less attention to, specially these code smells that no detection tool can detect; meanwhile put forward more accurate quality metrics; secondly, we should have more communications with industrial field, and doing a survey about code smell among the software developers and maintainers. Thirdly, we should carry out code smell study using the three types of projects. So we can have an comprehensive study about code smell.

Acknowledgment

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

References

[1] S. T. Danphitsanuphan P. Code smell detecting tool and code smell-structure bug relationship. In Engineering and Technology (S-CET), 2012 Spring Congress on, pages 1–5. IEEE, 2012.

[2] T. Dybå and T. Dingsøyr. Empirical studies of agile software development: A systematic review. Information and software technology, 50(9):833–859,2008.

[3] M. Fokaefs, N. Tsantalis, and A. Chatzigeorgiou. Jdeodorant: Identification and removal of feature envy bad smells. In Software Maintenance, 2007. ICSM 2007. IEEE International Conference on, pages 519–520, 2007.

[4] M. Fokaefs, N. Tsantalis, E. Stroulia, and A. Chatzigeorgiou. Jdeodorant: identification and application of extract class refactorings. In Proceedings

of the 33rd International Conference on Software

Engineering, pages 1037–1039. ACM, 2011.

[5] F. A. Fontana, E. Mariani, A. Morniroli, R. Sormani, and A. Tonello. An experience report on using code smells detection tools. In Software Testing,

Verification and Validation Workshops (ICSTW), 2011 IEEE Fourth International Conference on, pages 450–457. IEEE, 2011.

[6] M. Fowler. Refactoring: improving the design of existing code. Pearson Education India, 1999.

[7] S. Fu and B. Shen. Code bad smell detection through evolutionary data mining. In Empirical Software Engineering and Measurement (ESEM), 2015 ACM/IEEE International Symposium on, pages 1–9. IEEE, 2015.

[8] Y. Guo, C. Seaman, N. Zazworka, and F. Shull. Domain-specific tailoring of code smells: an empirical study. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2, pages 167–170. ACM, 2010.

[9] T. Hall, M. Zhang, D. Bowes, and Y. Sun. Some code smells have a significant but small effect on faults. ACM Transactions on Software Engineering and Methodology (TOSEM), 23(4):33, 2014.

[10] S. Keele. Guidelines for performing systematic literature reviews in software engineering. In Technical report, Ver. 2.3 EBSE Technical Report. EBSE. 2007.

[11] M. Lanza and R. Marinescu. Object-oriented metrics in practice: using software metrics to characterize, evaluate, and improve the design of object-oriented systems. Springer Science & Business Media, 2007.

[12] Z. Li, P. Liang, and P. Avgeriou. Application of knowledge-based approaches in software architecture: A systematic mapping study. Information and Software Technology, 55(5):777–794, 2013.

[13] L. C. Mantyla M V, Vanhanen J. Bad smells-humans as code critics[c]//software maintenance. In Proceedings. 20th IEEE International Conference on, pages 399–408. IEEE, 2004.

[14] 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.

[15] N. Moha, Y.-G. Gueheneuc, L. Duchien, and A.-F. Le Meur. Decor: A method for the specification and detection of code and design smells. Software

Engineering, IEEE Transactions on, 36(1):20–36, 2010.

[16] M. J. Munro. Product metrics for automatic identification of” bad smell” design problems in java source-code. In 11th IEEE International Software Metrics Symposium (METRICS’05). IEEE, 2005.

[17] S. M. Olbrich, D. S. Cruze, and D. I. Sjøberg. Are all code smells harmful? a study of god classes and brain classes in the evolution of three open source systems.

In Software Maintenance (ICSM), 2010 IEEE International Conference on, pages 1–10. IEEE, 2010.

[18] F. Palomba, G. Bavota, M. Di Penta, R. Oliveto, D. Poshyvanyk, and A. De Lucia. Mining version histories for detecting code smells. 2015.

[19] D. Sahin, M. Kessentini, S. Bechikh, and K. Deb. Code-smell detection as a bilevel problem. ACM

Transactions on Software Engineering and Methodology (TOSEM), 24(1):6, 2014.

[20] J. Schumacher, N. Zazworka, F. Shull, C. Seaman, and M. Shaw. Building empirical support for automated code smell detection. In Proceedings of the 2010

ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, page 8. ACM,2010.

[21] D. Sjoberg, A. Yamashita, B. C. D. Anda, A. Mockus,T. Dyba, et al. Quantifying the effect of code smells