Impact Factor (SJIF): 4.542

International Journal of Advance Research in Engineering, Science & Technology

e-ISSN: 2393-9877, p-ISSN: 2394-2444
Volume 4, Issue 4, April-2017

Description of tools for the detection of bad smells for enhancing quality of the software

Gunish Katar
Student
Department of Information Technology
HMR Institute of Technology
New Delhi, India

Sandhya Tarwani
Assistant Professor
Department of Information Technology
HMR Institute of Technology
New Delhi, India

Abstract— **Refactoring is a strategy to make a PC program more lucid and viable. A bad smell means that some misfortune is in the code, which requires refactoring to do. Many devices are accessible for identification and detection of these code smells. These devices change significantly in discovering philosophies and procure diverse abilities. In this study, we have concentrated on distinctive code notice recognition devices minutely which identifies different bad smells automatically that helps software developers and maintenance team to remove them in order to enhance software quality and attempt have been made to fathom our examination by framing various features of these tools.**
Keywords—refactoring; bad smells; robusta; tools;

I. INTRODUCTION

Programming upkeep is the most costly period of the product advancement life cycle (SDLC). It is the last period of the lifecycle and starts simply after the arrival of the item. Amid the whole life expectancy of the product, it gets changed persistently due to advancement in the innovation and developing rivalry in the business. According to the perceptions by Pressman [1], 70% of the aggregate endeavors are being used in upkeep period of the current framework. As per his review, 17% of the work is to right blames, 18% is utilized for the transportability and the real slice of 65% is utilized to actualize the progressions made in the necessity. The effect of changing business requests in programming can be surely knew with the assistance of Lehman's law [2] which clarifies that for protecting fulfillment persistent changes are required and without a doubt important however in the event that extra work is not done than the intricacy of the product will increment with each adjustment in the source code of the product. The many-sided quality of the product brings about low practicality, less understandability and exceedingly influenced usefulness. Programming viability is how much programming is comprehended, repaired and upgraded. As per Institute of Electrical and Electronics Engineers (IEEE) guidelines [3], it is the simplicity with which a product framework or segment can be adjusted to right blames, enhance execution or different traits.
In computer science field, smell is considered as a symptom of the poor quality of the source code. Fowler and Beck [4] initiated the concept of bad smell. They described smells as surface indication which usually means deeper problem in the software. 22 bad smells were described by them in their study that correlates software design with maintenance of the system. Bad smells are the symptoms that define deeper problem in the source code. It negatively affects the design principles and quality of the software. Martin [5] calls a list of code smells as value system for software craftsmanship. Often this deeper problem is resolved when code is subjected to a short feedback cycle where it is restructured in small as well as controlled steps. Hence, from the programmer point of view, bad smells are the indication of refactoring.
Refactoring has turned into an outstanding procedure for the product building group to remove the bad smells in the source code. It is a procedure to enhance the inner structure of a program without changing its outer conduct. Visit refactoring of the code helps software engineer to make the code more justifiable, discover bugs and make it reasonable for the expansion of new elements and to program speedier. Over all that, it enhances the plan of the product and along these lines the general nature of the product also optimises. Refactoring should be possible physically and consequently. Broad writing is accessible on refactoring of the question situated projects and various devices are accessible for the programmed refactoring of the code. Refactoring has an uncommon association with the ideas of figuring out and lithe programming improvement. One of dexterous programming improvement models, eXtreme Programming (XP), considers refactoring as one of its fundamental components. Refactoring persistently enhances the outline of the product and helps the advancement and incremental improvement of the product.

Different steps for the activity of refactoring have been proposed by different researchers. Tom Mens *et al.*, [6] described six activities in the process of refactoring. These are: Identifying; where refactoring should be done, determining which refactoring(s) should be applied to the identified places, ensuring that the applied refactoring preserves behaviour, applying the refactoring, assessing the effects of refactoring on the quality of the software and process, assessing the consistency between the code and other software artefacts. Bad smells are design flaws or structural problem of software that can be handled through refactoring.
A variety of software tools have been developed for the automated detection of bad smells and they differ in their capabilities and approaches. Determining whether some piece of code contains bad smell(s) is somewhat subjective and still there is a lack of standards. In this work, a comparative study is carried out regarding bad smell detection tools namely JDeodorant, inCode, Stench blossom, PMD, robusta, Incode, smellsheet detective. Their detection methodology is discussed in greater detail.

II. RELATED WORK

Many reviews have been led to examine which segment of the product needs updating medications utilizing OO measurements. In 2005, Slinger [7] in his postulation proposed an overshadowing module, CodeNose, which utilizes the Eclipse JDT parser to manufacture digest language structure trees that speak to the source code. For a situation study, the module was tried by playing out the code notice recognition prepare on a current programming framework. Trifu et al. [8] proposed a quality-guided way to deal with distinguish and evacuate adversely affecting plan defects. For every sort of configuration blemish, every option arrangement were predesigned and their effects on quality properties were predefined. Its option arrangements were checked and the one with most noteworthy positive effect on programming quality was chosen. Shrivastava and Shrivastava [9] exhibited a contextual investigation in which a stock was considered and endeavors were made to enhance the nature of the framework by refactoring. Distinctive renditions of the product were constructed and their change over quality was mulled over. Spinellis [10] dissected distinctive rendition of open source programming and inspected that how refactoring influences the measurements and the nature of the product. It was found that refactoring system not generally enhances the nature of the product or designers were not ready to apply it viably. Awful stenches are the side effects to show further issue in the code which should be expelled with the assistance of refactoring for improving practicality and nature of the product. Different works have been done in the field of refactoring till now; some of them have been examined here. Zibran and Roy [11] clarify the vital need of refactoring scheduler and proposed a limitation programming approach for strife mindful ideal planning of code clone refactoring. Liu et al. [12] in their paper proposed a contention mindful booking approach with the assistance of heuristic calculation. Liu et al. [13] proposed identification and determination grouping for various sort of terrible stenches that will help in refactoring. Tourwe and mens [14] utilized the strategy of rationale meta programming to identify terrible stench. Piveta et al. [15] proposed a way to deal with decrease the quantity of refactoring by disposing of those that semantically doesn't bode well and keep away from those that gives same outcomes. Jensen and Cheng [16] proposed an approach that utilizations hereditary programming and programming building measurements to recognize the arrangement of refactoring to apply to a product outline. Dexun et al. [17] proposed a substantial class awful discovery approach in light of class length dissemination model and attachment measurements. Rao and Reddy [18] proposed quantitative technique that rolls out utilization of idea plan improvement proliferation likelihood lattice to distinguish two sorts of awful stenches i.e. shotgun surgery and unique change terrible stench.

III. TOOLS

There are various types of tools developed to identify and detect bad smells in the source code automatically. In this study, we have focused on the tools that are easily available and flexible to use. Figure 1 shows the type of tools we have used in this study.

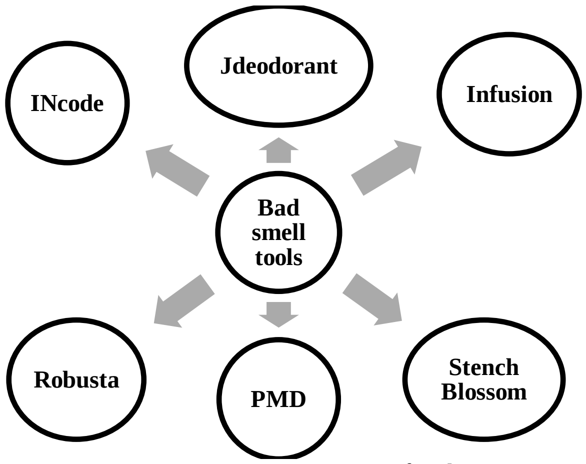


Figure 1 Various types of tools

A. Jdeodorant

JDeodorant is an Eclipse plugin that identifies design problems in software, and resolves them by applying appropriate refactoring like God class, long method etc. JDeodorant only detects five types of bad smells. They are, God Class, Long method, Type checking, Feature Envy and **Duplicated Code**. JDeodorant employs a variety of novel methods and techniques in order to identify code smells and suggest the appropriate refactoring that resolve them. Feature Envy problems are resolved through Move Method refactoring where as Type Checking problems are resolved by Replace Conditional with Polymorphism and Replace Type code with State/Strategy refactoring. While other three, Long Method problems through appropriate Extract Method refactoring, God Class problems by Extract Class refactoring, Duplicated Code problems are resolved by appropriate Extract Clone refactoring. This tool uses ASTParser API of eclipse IDE to detect bad smells from code and ASTRewrite API for refactoring. JDeodorant encompasses various unique and innovative features like Transformation of expert knowledge to fully automated processes, Pre-Evaluation of the effect for each suggested solution, User guidance in comprehending the design problems, User friendliness through one click approach.

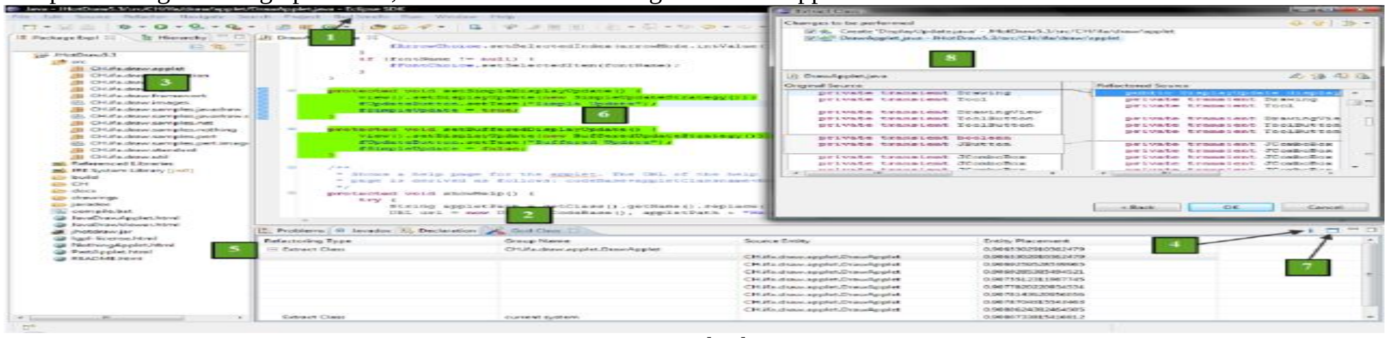


Figure 2 Jdeodorant

B. Infusion

InFusion at present is current, commercial evolutionary version of iPlasma. InFusion supports the analysis, diagnosis of a system at the architectural, as well as at the code level and covers all the important phases of the analysis process. InFusion allows users to detect more than 20 design flaws and code smells namely Code Duplication, classes that break encapsulation (Data Class, God Class), methods and classes that are heavily coupled or illdesigned class hierarchies and other code smells like Cyclic Dependencies, Brain Method, Data Class, Feature Envy, God Class, Intensive Coupling, Missing Template, Method, Refused Parent, Bequest, Significant, Duplication. InFusion has its roots in iPlasma, and then extended with more functionality.

C. Infusion

InFusion at present is current, commercial evolutionary version of iPlasma. InFusion supports the analysis, diagnosis of a system at the architectural, as well as at the code level and covers all the important phases of the analysis process. InFusion allows users to detect more than 20 design flaws and code smells namely Code Duplication, classes that break encapsulation (Data Class, God Class), methods and classes that are heavily coupled or illdesigned class hierarchies and other code smells like Cyclic Dependencies, Brain Method, Data Class, Feature Envy, God Class, Intensive Coupling, Missing Template, Method, Refused Parent, Bequest, Significant, Duplication. InFusion has its roots in iPlasma, and then extended with more functionality.

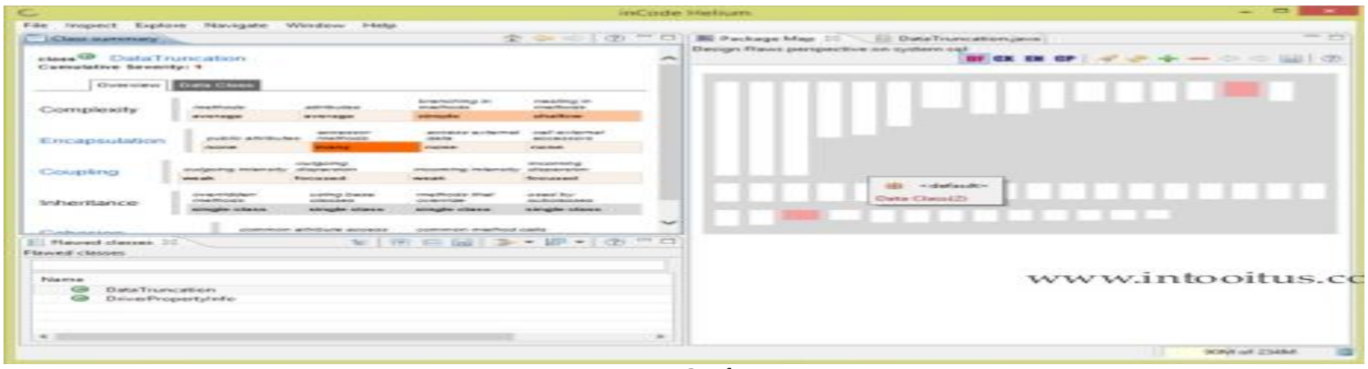
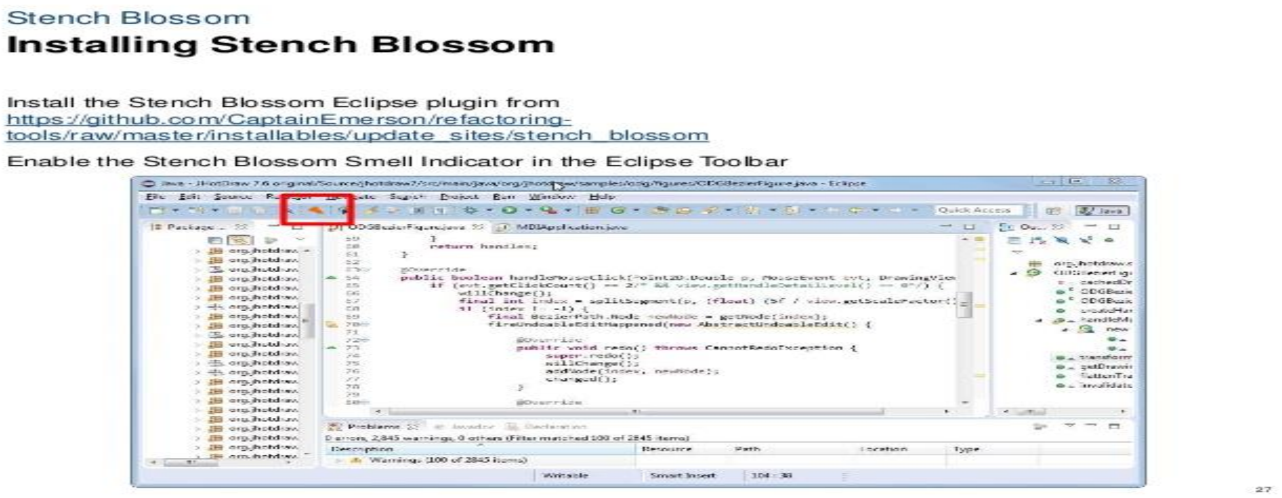


Figure 3 Infusion

D. Stench Blossom

StenchBlossom is a smell finder that gives an intuitive encompassing perception intended to first give software engineers a snappy, abnormal state diagram of the smells in their code, and after that, to help comprehend the wellsprings of the code smells, yet is not reasonable for figuring out. It does not give numerical qualities, but rather just a visual limit: the span of a petal is specifically relative to the element of the code smells. The main conceivable methodology to discover code odours is to physically peruse the source code, searching for a petal whose size is sufficiently huge to make us assume that there is a code smell. It is a module for the for the Overshadow condition that furnishes the developer with three distinct perspectives, which logically offer more data about the smells in the code being envisioned. The apparatus can identify 8 smells like Data Clumps, Feature Envy, Message Chain, Switch Statement, Typecast and so on.



Stench Blossom

Installing Stench Blossom

Install the Stench Blossom Eclipse plugin from https://github.com/CaptainEmerson/refactoring-tools/raw/master/installables/update_sites/stench_blossom

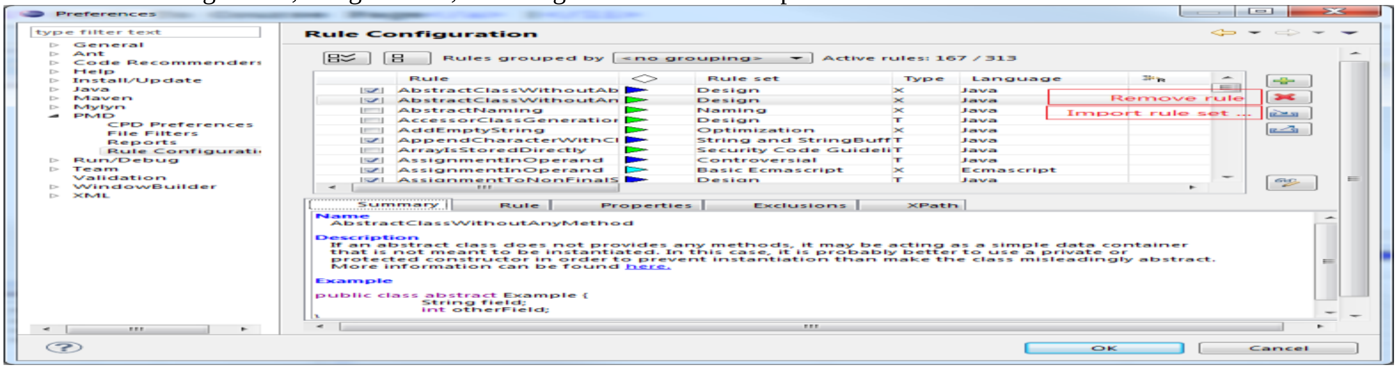
Enable the Stench Blossom Smell Indicator in the Eclipse Toolbar

27

Figure 4

E. PMD

PMD checks Java source code and searches for potential issues like: conceivable bugs, for example, dead code, empty try/catch/finally/switch statements, unused local variables, parameters and copy code. Additionally, PMD can distinguish three smells that are Large Class, Long Method, and Long Parameter List and permits to set the limits values for the measurements.



Rules grouped by <no grouping> Active rules: 167 / 313

	Rule	Rule set	Type	Language
✓	AbstractClassWithoutAb	Design	X	Java
✓	AbstractClassWithoutAn	Design	X	Java
	AbstractNaming	Naming	X	Java
	AccessorClassGeneratior	Design	T	Java
	AddEmptyString	Optimization	X	Java
✓	AppendCharacterWithCl	String and StringBuff	T	Java
	ArrayIsStoredDirectly	Security Code Guideli	T	Java
✓	AssignmentInOperand	Controversial	T	Java
✓	AssignmentInOperand	Basic Ecmascript	X	Ecmascript
✓	AssignmentToNonFinalS	Design	T	Java

Name
AbstractClassWithoutAnyMethod

Description
If an abstract class does not provides any methods, it may be acting as a simple data container that is not meant to be instantiated. In this case, it is probably better to use a private or protected constructor in order to prevent instantiation than make the class misleadingly abstract. More information can be found here.

Example
```
public class abstract Example {
        String field;
        int otherField;
```

Figure 5 PMD

F. Robusta

Robusta, an automatic bad smell identification instrument has been created to handle exceptionally handling bad smells. Robusta is equipped for recognizing special case related to bad smells. Utilizing Robusta client can include markers the alter see for data of comparing awful stenches. It can likewise be used to refactoring certain sorts of awful stenches or, all the more imperatively, to produce reports for code quality representation. It is a programmatic bad smell detection tool which has been developed to help Java developers deal with 7 types of exception handling bad smells including careless cleanup, dummy handler, empty catch block exception thrown in finally block, nested try statement, unprotected main.

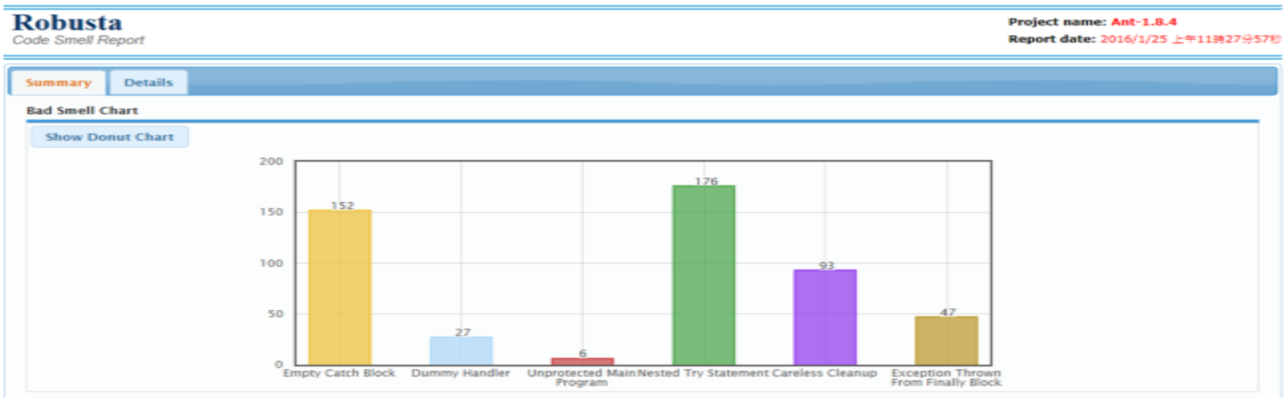
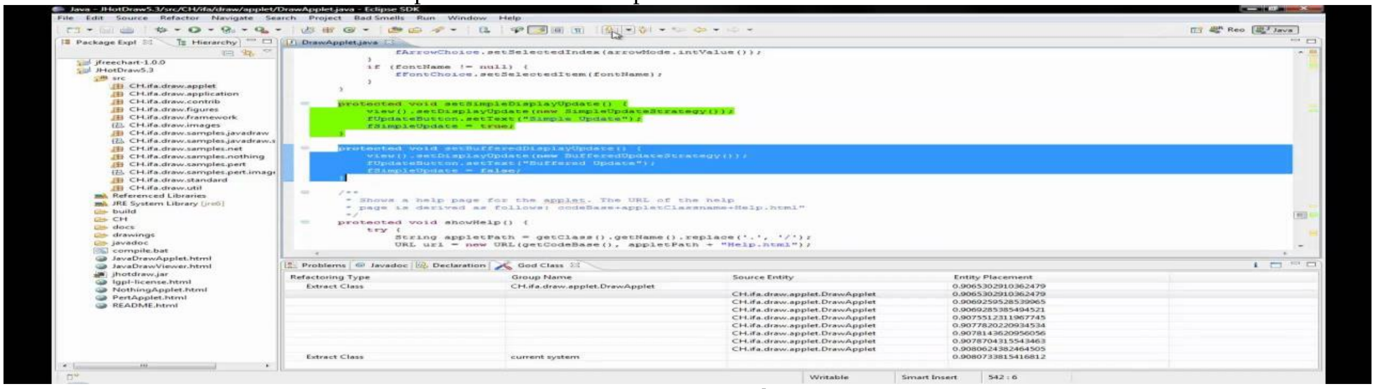


Figure 6 Robusta

G. InCode

InCode Helium is the cutting-edge quality investigation apparatus outlined by designers. InCode Helium recognizes quality issues and helps you comprehend their basic causes. Some of its features are automated detection of design problems and ranking based on problem severity, interactive visual maps of the code with smart spotlight functionality, metrics-based characterisation of software systems with built-in intuitive interpretation, powerful design and code exploration tools with context-sensitive exploration tips, fast and reliable analysis of very large code bases (Java, C and C++).
The principle highlight of this device is to bolster software engineers to program in code smells mindful programming condition. It works out of sight of overshadowing. Amid programming, if software engineer composes any awful structure, then it demonstrates these smells as, "shroud show blunder" and notices in the state of red shading hinders alongside code. inCode distinguishes 4 bad smells i.e. Highlight begrudge, God class, Duplicate code and Data class. These terrible smells discoveries depend on question arranged measurements. The larger part is that client doesn't need to associate with the basic measurements specifically. Measurements carry out their occupation in the engine, while you zip specifically to the valuable conclusions. Besides, it additionally has functionalities of development of Metrics Pyramid. The pyramid demonstrates the estimations of various measurements and their correspondence inside the product.


Java - JHotDraw5.3/src/CH/ifa/draw/applet/DrawApplet.java - Eclipse SDK
File Edit Source Refactor Navigate Search Project Bad Smells Run Window Help
Package Expl
Hierarchy
DrawApplet.java
Problems
Javadoc
Declaration
God Class
Writable
Smart Insert
542 : 6

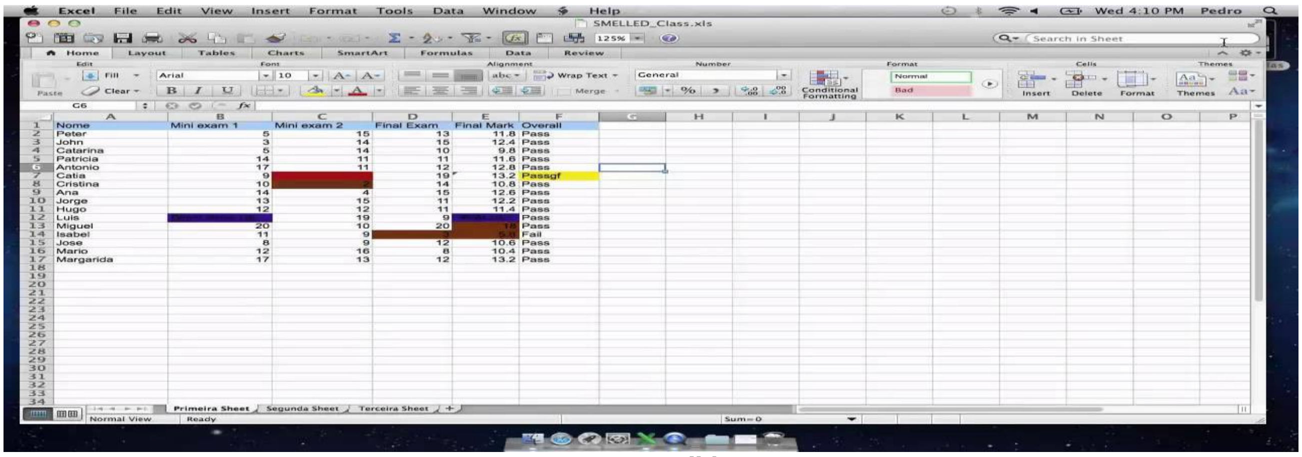
```
            fArrowChoice.setSelectedIndex(arrowMode.intValue());
        }
        if (fontName != null) {
            fFontChoice.setSelectedItem(fontName);
        }
    }

    protected void setSimpleDisplayUpdate() {
        view().setDisplayUpdate(new SimpleUpdateStrategy());
        fUpdateButton.setText("Simple Update");
        fSimpleUpdate = true;
    }

    protected void setBufferedDisplayUpdate() {
        view().setDisplayUpdate(new BufferedUpdateStrategy());
        fUpdateButton.setText("Buffered Update");
        fSimpleUpdate = false;
    }

    /**
     * Shows a help page for the applet. The URL of the help
     * page is derived as follows: codeBase+appletClassname+Help.html"
     */
    protected void showHelp() {
        try {
            String appletPath = getClass().getName().replace('.', '/');
            URL url = new URL(getCodeBase(), appletPath + "Help.html");
```

Refactoring Type	Group Name	Source Entity	Entity Placement
Extract Class	CH.ifa.draw.applet.DrawApplet		0.9065302910362479
		CH.ifa.draw.applet.DrawApplet	0.9065302910362479
		CH.ifa.draw.applet.DrawApplet	0.9069259528539965
		CH.ifa.draw.applet.DrawApplet	0.9069285385494521
		CH.ifa.draw.applet.DrawApplet	0.9075512311967745
		CH.ifa.draw.applet.DrawApplet	0.9077820220934534
		CH.ifa.draw.applet.DrawApplet	0.9078143620956056
		CH.ifa.draw.applet.DrawApplet	0.9078704315543463
		CH.ifa.draw.applet.DrawApplet	0.9080624382464505
Extract Class	current system		0.9080733815416812

Figure 7 InCode

H. Smellsheet Detector

So as to naturally examine spreadsheets SmellSheet Detective that expands on the library to distinguish the smells. This execution joins the Java programming dialect, the Google Web Toolkit (GWT), the Apache POI library and the Google libraries to work with spreadsheets inside the Google Docs condition. We chose to bolster spreadsheets written in the Google Docs stage since it is ending up plainly more utilized, and even the prevalent Microsoft Office suite has likewise its online rendition. In fact, the movement from desktop to online applications is winding up noticeably extremely normal. In any case, we additionally bolster spreadsheets composed utilizing desktop applications. The client can choose a solitary sheet or the full spreadsheet to be investigated and can likewise peruse the spreadsheets in the PC and select one. In the wake of choosing the information, The SmellSheet Detective creates the consequences of terrible stenches in a few organizations (csv, exceed expectations and LATEX tables). The SmellSheet Detective was created on top of a measured and extensible library written in Java.



	A	B	C	D	E	F
1	Nome	Mini exam 1	Mini exam 2	Final Exam	Final Mark	Overall
2	Peter	5	15	13	11.8	Pass
3	John	3	14	15	12.4	Pass
4	Catarina	5	14	10	9.8	Pass
5	Patricia	14	11	11	11.6	Pass
6	Antonio	17	11	12	12.8	Pass
7	Catia	9		19	13.2	Passgf
8	Cristina	10	2	14	10.8	Pass
9	Ana	14	4	15	12.6	Pass
10	Jorge	13	15	11	12.2	Pass
11	Hugo	12	12	11	11.4	Pass
12	Luis	[illegible]	19	9	[illegible]	Pass
13	Miguel	20	10	20	15	Pass
14	Isabel	11	9	3	5.9	Fail
15	Jose	8	9	12	10.6	Pass
16	Mario	12	16	8	10.4	Pass
17	Margarida	17	13	12	13.2	Pass

Figure 8 Smellsheet Detector

IV. Future Work and Conclusion

In this paper, we have discussed about the various tools that have been used at present to detect bad smells. Some tools basically help detecting but most of the tools provide an option for refactoring. In future, we are planning to elaborate this topic by discussing about the pros and cons of these tools along with comparative analysis that will help software developers and maintainers to use these tools wisely. In future, we are also planning to conduct empirical analysis on the open source software to detect smells with the help of these software that will reduce efforts as well as time.

References

[1] R.S. Pressman, Software Engineering: A Practitioner's Approach, 6 Edition, McGraw Hill, Book Co, 2005.

[2] M.M Lehman. D.e Perry, J.F. Ramin, W. M Turksi and P. Wernick, " Metrics and laws of software evolution-the nineties view." In proc. Of the 4th Int. Symb. on software metrics, Albuquerque, New Mexico, 2000. IEEE

[3] IEEE Std. 610.12-1990.: Standard Glossary of Software Engineering Terminology, IEEE Computer Society Press, Los Alamitos, CA, 1993.

[4] M. Fowler and K. Beck, " Refactoring: improving the design of existing code," in Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA. DOI=10.10007/3-540-45672-4_31.

[5] R. C. Martin "Clean Code: A Handbook of Agile Software Craftsmanship," Prentice Hall, 2009.

[6] T. Mens, T. Tourwe and F. Munoz, "Beyond the Refactoring Browser: Advanced Tool Support for Software Refactoring," in Proc. Int. Workshop on Principles of Software Evolution, Helsinki, Finland. Engineering and Management, ISSN 0975-6809, vol. 5, no. 2, 2002, pp.165-173. DOI= http://dx.doi.org/10.1109/IWPSE.2003.1231207.

[7] S. Slinger, "Code smell detection in eclipse," Ph. D. dissertation, Dept. Soft. Tech., Delft Univ. of Tech., Netherland, 2005.

[8] A. Trifu, O. Seng, and T. Genssler, "Automated design flaw correction in object-oriented systems," in Proc. of CSMR 2004 (the 8th European Conf. on Software Maintenance and Reengineering, IEEE Computer Society, 2004, pp. 174–183.

[9] S. V. Shrivastava and V. Shrivastava, " Imapact of metrics based refactoring on the software quality: a case study," in TENCON 2008 - 2008 IEEE Region 10 Conf., Hyderabad, 2008, pp. 1-6.

[10] Stroggylos, K. and Spinellis, D. 2007. Refactoring – Does it improve Software Quality?, Proc. 5th International Workshop on Software Quality, pp. 1-6. DOI= http://dx.doi.org/10.1109/WOSQ.2007.11.

[11] M. Z. Zibran and C. K. Roy, "A Constraint Programming Approach to Conflict-aware Optimal Scheduling of Prioritized Code Clone Refactoring," in *11th IEEE Int. Working Conf. on source code Analysis and Manipulation*, 2011, pp. 105-114.

[12] H. Liu, G. Li, Z. Y. Ma and W. Z. Shao, "Conflict-aware schedule of software refactorings," in *Institution of Engineering and Technology*, Vol. 2, no. 5, 2008, pp. 446-460.

[13] H. Liu, Z. Ma, W. Shao and Z. Niu, "Schedule of Bad Smell Detection and Resolution: A new Way to Save Efforts," in *IEEE transactions on Software Engineering*, Vol. 38, no. 1, 2012, pp. 220-235.

[14] T. Tourwe and T. Mens, "Identifying Refactoring Opportunities Using Logic Meta Programming," in *Proc. 7th European Conf. on Software Maintainence And Reengineering (CSMR)*, 2003, pp. 91-100.

[15] E. Piveta, J. Araujo, M. Pimenta, A. Moreira, P. Guerreiro and R. T. Price, "Searching for Opportunities of Refactoring Sequences: Reducing the search space," in *Annual IEEE Int. Computer Software and Applications Conf.*, 2008, pp. 319-326.

[16] A. C. Jensen and B. H. C. Cheng, "On the Use of Genetic Programming for Automated Refactoring and the Introduction of Design Patterns," in *Proc. 12th annual Conf. Genetics and evolutionary computation GECCO*, Portland, Oregon, USA, 2010, pp. 1341-1348.

[17] J. Dexun, M. Peijun, S. Xiaohong and W. Tiantian, "Detection and Refactoring of Bad smell caused by large scale," in *Int. Journal of Software Engineering and Applications (IJSEA)*, Vol. 5, no. 5, 2012, pp. 1-13.

[18] A. A. Rao and K. N. Reddy, "Detecting Bad smellsin Object-Oriented Design Using Design Change Propogation Probability Matrix," in *Proc. Int. Multiconference of Engineers and Computer Scientists (IMECS)*, Vol. 1, 2008.