THINK INDIA JOURNAL ISSN: 0971-1260

Vol-22-Issue-3-July-2019

On The Statistical Relationships Between The Coding Standards and The Reported Bugs of The Python Software Systems.

Mir Mohammad Yousuf [1] , Bisma Majid[2] , Mamoon Rashid[3] ,Umer Iqbal wani[4]

[1,3,4] Assistant Professor,School of Computer Science and Engineering
Lovely Professional University
[2]Research Scholar, Department of electronics and communication,
N.I.T Srinagar

Abstract---Software maintenance nowadays is identified by the huge costs and slow speed of implementation. The software doesn't get tired, with the introduction of the new requirements, it needs to go through the stabilization phase. The change in one module reflects the change in the other modules. Poor coding skills can lead to the increase in the bug list of a particular software product that affects the maintenance cost. This paper presents the relationship between the better practices of coding and the maintenance cost in terms of the bugs reported. The results show that the bug rate of a software reduces with the increase in the skills of the programming i.e., following the coding standards of the programming language.

Keywords: Bugs, coding, software maintenance, python software project, stabilization, programming language.

I. INTRODUCTION

The activities that are performed throughout the pre-delivery still because the post-delivery stage, wherever pre-delivery activities embrace designing for the post-delivery operations, supportability and provision determination. The post-delivery activities embrace computer code modification, training, and operation at the assistance table.ISO/IEC 12207 defines a maintainer as "an organization that performs the maintenance activities".[IS012207].The primary activities of the software have been defined as: "a process implementation; problem and modification analysis; modification implementation; maintenance review/ acceptance; migration; and retirement". They are further defined by the tasks in ISO/IEC 12207.There are four types of maintenance

1. **Adaptive maintenance** : "The modification of a software product, performed after delivery, to keep a software product usable in a changed or changing environment." Adaptive maintenance accounts 25% of all the maintenance activities. [5]
2. **Corrective maintenance**: "The reactive modification of a software product performed after delivery to correct discovered problems." Corrective maintenance accounts 20% of all the maintenance activities. [5]
3. **Perfective maintenance**"The modification of a software product after delivery to detect and correct latent faults in the software product before they are manifested as failures." Perfective maintenance accounts 50% of all the maintenance activities. [5]

4. **Preventive maintenance** "The modification of a software product after delivery to detect and correct latent faults in the software product before they become operational faults". Preventive maintenance accounts 5% of all the maintenance activities. [5]

Today, software bugs stay never-ending and dear fixture of commercial and OSS development. using the software configuration management (SCM) systems, the software projects rigorously control their changes and capture the bug reports using the bug tracking software like Bugzilla. so recording that changes within the SCM fixes a selected bug within the change tracking system. the key problem with the bug fix knowledge is that it sheds no light-weight on once a bug has been injected into the code and who injected it. oft the person fixing the bug isn't the one who initial created the bug.

It can be predicted that the Software system is subject to failure during execution caused by bugs remaining in the system and bugs remaining in the software equally affect the failure rate of the software.8 Thus we can say that the introduction of the bugs lead to the software maintenance. IEEE defines software maintenance as *"The modification of a software product after delivery to correct faults, to improve performance or other attributes or to adapt the product to a modified environment"[5]*. Robert Glass writes in his 'Facts and Fallacies of Software Engineering', "about 60% of a software's cost is maintenance, and about 60% of the maintenance cost is improvement". The growth rate from the technological point of view in the software field is more as compared to other fields[16] . Many metrics such as size/volume metrics, OO metrics and the complexity metrics are used in software engineering research as surrogates for the maintainability of software systems. As the software ecosystem is continuously changing and there is need of the regular maintenance to help the software system to adapt to the new change. Not only the work patterns but the software platforms, hardware upgrades, compilers etc. all affect the working of the software system[19]. The software is required to be kept fresh so that it can adapt to the changing circumstances and thus making the software adaptable.

A series of quantitative and qualitative analysis has been done in this paper to understand the effects of uplifting a software patch. Overall we analysed 32 versions of Trac and 4 versions of Django. "Trac is an open source web-based project management and bug tracking system that is adopted by various organizations for the use of bug tracking system for both free and open-source software and proprietary projects and products". It is developed in python development framework and is used by Internet Research task Force, JQuery UI, and WordPress.

II. LITERATURE REVIEW

The software bugs and their effect on the software quality and maintenance has gradually increased the interest of the researchers for which they were interested lately. Although there has been a less amount of research done in analyzing the python software system as compared to other systems of other platforms. Matteo Orrú et al (2015) [9] presented a dataset of metrics of python systems. They designed the corpus, mentioned the most problems in making it, and provided its description and limitation.. They conjointly urged the utilization of the info set for the empirical studies of the python systems that permits dependableness of so lowers the price of the experiments.. Zhifei Chen et al (2016) [2] introduced 11 smells and described the detection strategy. They conjointly enforced the detection tool specifically Pysmell that was accustomed establish the code smells in 5 real world python systems. Their results showed that Pysmell could

detect 285code smell instances in total with the average precision of 97.7%, which revealed that large classes and large methods are more prevalent. Mortiz et al (2014) [7] empirically explored the problems fixed through the modern code reviews (MCR) in open source software systems (OSS) and tried to increase the understanding the practical benefits that the MCR process produces on the code reviews. they manually classified one,1400 changes going down within the reviewed code from the 2 OSS comes into a valid categorization theme[10][13]. The results showed that the types of due to the modern code review process in OSS were very similar to those in the academic systems and the industry from the literature[11]. Their main contribution enclosed the amendment classification i.e, a valid amendment classification for the java and C comes to higher perceive the particular effects of code reviews on 1400 changes exhausted reviews. The similarity quantitative relation of maintainability-related to the practical issues being 75:25. They conjointly discovered that 7-35% of the code review comments were discarded and 10-22% of the changes weren't triggered by a precise comment. additionally to the present found that the bug-fixing tasks cause fewer changes and also the tasks with a lot of altered files and the next code churn have a lot of changes. Their results reveal that seventy-fifth changes ar associated with the evaluability of the system. Then a lot of code churn or the upper range of touched files during a task, the a lot of they expected the review on the average.

Nicole Vavrova et al (2016)[8] developed a tool called the design defect detector, which parses a python module, creates a code model of it and reports on the presence of various design defects found there. They however compared the design defect density in python to the design defect density in java and found that the density measured in python (6.07 defects per 10K LOC) was slightly lower than the density measured in java (8.37 defects per 10,000 lines of code)[16]18].

Marco castallucioo et al (2017)[12] conducted a series of qualitative and quantitative chemical analysis to know the choice creating the method of patch uplift at Mozilla and characteristics of elated patches that introduce regressions. They analyzed 33664 issue reports in seventeen versions of Mozilla over a amount. They examined the characteristics of the elated patches that introduced regressions within the code and located that they're a lot of complicated than clean uplifts.

Uttamjit Kaur and Gagandeep Singh suggests some issues and problems encountered by code maintenance method. There ar some problems with code maintenance i.e.: info size, system age, maintenance budget, system size, employee's size or restructuring for amendment[12][14]. They presents many ways that to scale back value and efforts concerned in code maintenance. code maintenance prices may be reduced considerably if the code design is well outlined, clearly documented, it creates the environment that promotes style consistency through the utilization of pointers and testing quality events, by up to seventy two. additionally, the model enhanced the prediction accuracy by up to twenty three share points compared to ancient bug tracking approaches[11][19].

III. METHODOLOGY AND APPROACH

The source code of a project is an essential component for any type of analysis. The characteristics of a source code determine the types of results we can infer for them. However, the bug list for such projects can be essential component for ensuring better quality assurance of the product. If we select a project, big enough to represent an industry sized software, the inferences can hold true for the industry software as well as the academicians. Therefore, a person needs to be wise while selecting an open source project .We selected an open source

project with some stability and proper documentation., all the bugs with the priority being highest, high, normal, low and lowest and type of bug being defect, enhancement of 32 versions of the Trac are collected. Table 1 and 2 shows the release dates of different versions of Django and Trac that were analyzed.

Table1: Release dates of different versions of Django

S.NO	Django version	Release Date
1.	Django 1.8	*(April 1,2015)*
2.	Django 1.9	*(December 1,2015)*
3.	Django 1.10	*(August 1,2016)*
4.	Django 1.11	*(April 4,2017)*

Table 2: The release dates of different versions of Trac

S.NO	TRAC Version	Release Date
TRAC 0.0.x releases		
1.	0.12 'Babel'	(June 13, 2010)
2.	0.12.1	(October 9, 2010)
3.	0.12.2	(January 31, 2011)
4.	0.12.3	(February 6, 2012)
5.	0.12.4	(September 7, 2012)
6.	0.12.5	(January 15, 2013)
7.	0.12.6	(October 23, 2014)
8.	0.12.7	(July 12, 2015)
Trac 1.0.xx releases		
9.	1.0 “Cell”	(September 7, 2012)
10.	1.0.1	(February 1, 2013)
11.	1.0.2	(October 23, 2014)

12.	1.0.3	(January 13, 2015)
13.	1.0.4	(February 8, 2015)
14.	1.0.5	(March 24, 2015)
15.	1.0.6	(May 20, 2015)
16.	1.0.7	(July 17, 2015)
17.	1.0.8	(July 24, 2015)
18.	1.0.9	(September 10, 2015)
19.	1.0.10	(February 20, 2016)
20.	1.0.11	(May 7, 2016)
21.	1.0.12	(July 4, 2016)
22.	1.0.13	(September 11, 2016)
23.	1.0.14	(June 9, 2017)
24.	1.0.15	(June 16, 2017)
TRAC Versions1.1.x releases		
25.	1.1.1	(February 3, 2013)
26.	1.1.2	(October 23, 2014)
27.	1.1.3	(January 13, 2015)
28.	1.1.4	(March 24, 2015)
29.	1.1.5	(May 18, 2015)
30.	1.1.6	*(July 17, 2015)*
TRAC Versions1.2.x releases		
31.	1.2 “Hermes”	(November 5, 2016)
32.	1.2.1	(March 29, 2017)

The degree of the impact that a bug has on the event or operation of a part system is termed the bug severity. Higher impact on the severity can cause assignment of the upper to the bug. the standard assurance engineer typically determines the severity level of the bug. The bug severity classification in many sorts, the particular terminologies and their that means will vary relying upon the folks, projects, organizations, or defect following tools. The bugs of the preceding versions were extracted relying upon the severity, type, priority of the bugs.

The versions of the Django were selected on the basis if the substantial difference in the release dates and were examined for the presence of bugs. The bugs were selected depending upon the type and severity of the bug.

After the mapping of the bugs was done with the parameters mentioned above, the resultant files contained the distribution of the defects per version as:

Table3: The total number of bugs in different Trac versions

S.NO	TRAC version	Defects
1	TRAC 0.12.x	453
2	TRAC 1.0.x	334
3	TRAC 1.1.x	54
4	TRAC 1.2.x	25
TOTAL		866

Table4: Total number of bugs in each version of Django

S.NO.	Django Version	No. of bugs
1	1.8	1132
2	1.9	891
3	1.10	864
4	1.11	818
TOTAL		3705

IV. RESULTS AND DISCUSSION

The 5 versions of Django and 32 versions of the Trac were tested on the pylint. A script was written in python that analyzes the source code of the entire python project looking for the signs of poor quality. Pylint is a source code, bug and the quality checker for the python programming language that follows the style recommended by PEP8 [13] (the official style guide of python core). The script tests each and every file and rates the code of the entire project depending upon the standards like if the variable names are well formed according to the project's coding standards or if declared interfaces are truly implemented [14] or not, etc. followed in the code. The above figures show the average rate of the code and the total number of bugs reported in each version.

ISSN: 0971-1260

Vol-22-Issue-3-July-2019

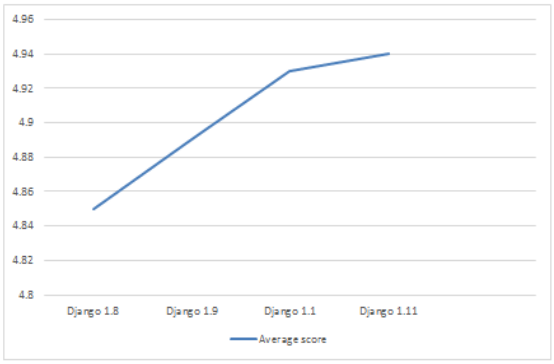

Figure 1: Average rate of code of Django versions

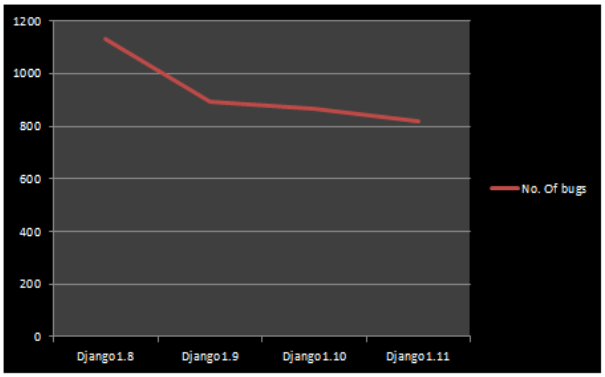

Figure 2: Number of bugs reported in Django versions

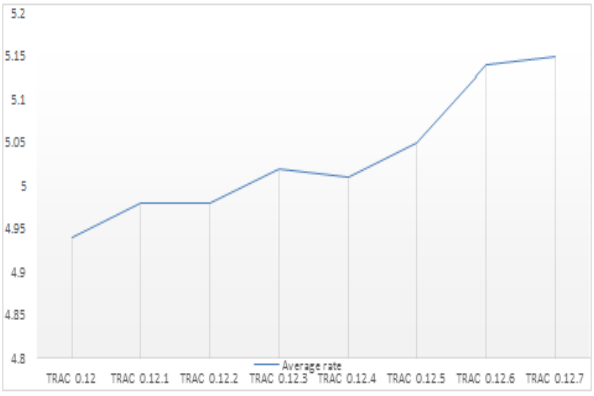

Figure 3 Average rate of code Trac 0.12.xx versions

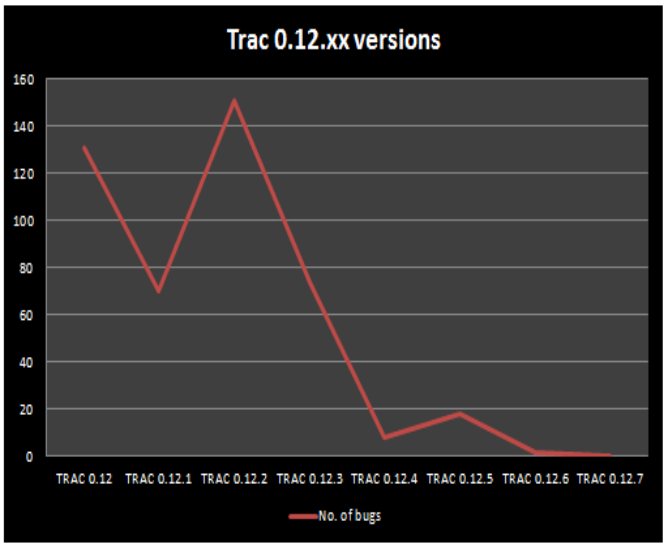

Figure 4: Number of bugs reported in Trac 0.12.xx versions

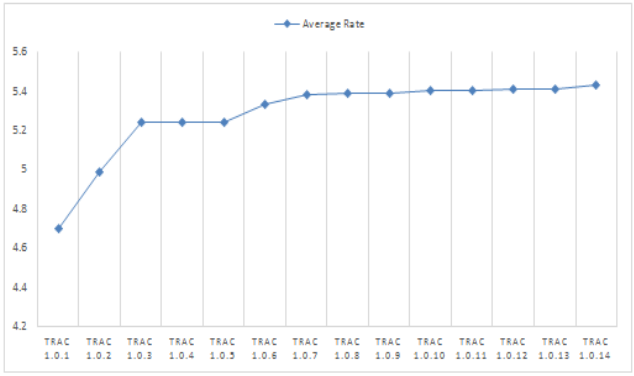

Figure 5 Average rate of Trac 1.0.xx versions

ISSN: 0971-1260

Vol-22-Issue-3-July-2019

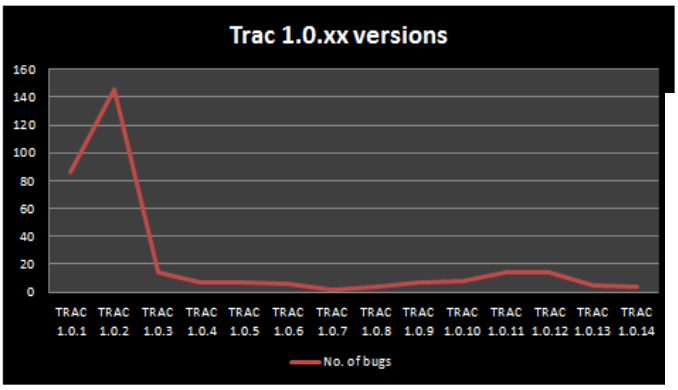

Figure 6: Number of bugs reported in Trac 1.0.xx versions

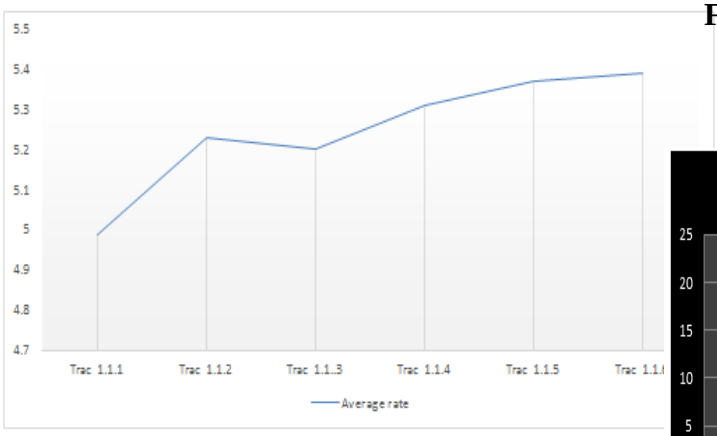

Figure 7: Average rate of code Trac 1.1.xx version

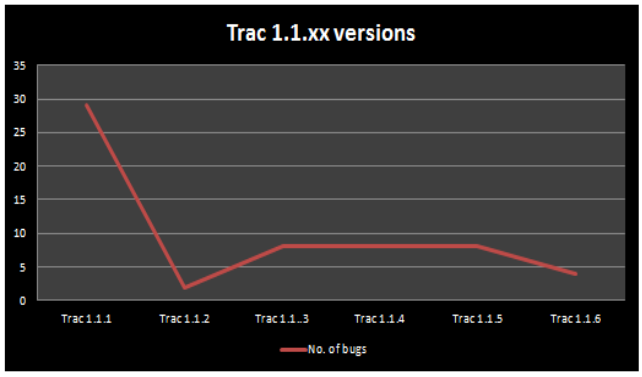

Figure 8: Number of bugs reported in Trac 1.1.xx versions

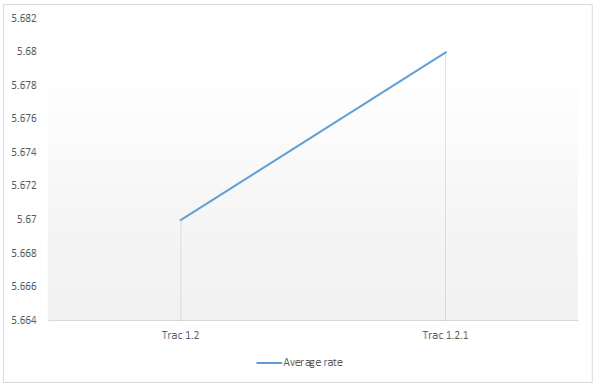

Figure 9: Average rate of code Trac 1.2.xx versions

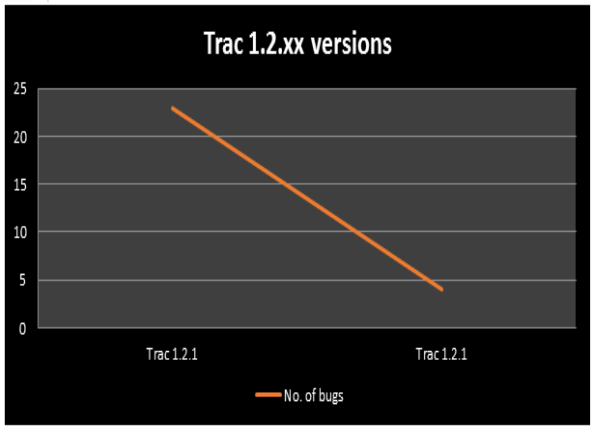

Figure 10: Number of bugs reported in Trac 1.2.xx versions

Figure 1,3,5,7,9 show the average rate of the code of the respective versions of Trac tested on the pylint. The elevating slope shows that the patch uplift results in improving the standards of the code i.e., it depicts with increase in the version of the project, the developers try to increase the quality of the code by following the standards of that particular programming language. As a result the bug rate of the software product is decreased. Figure 2, 4, 6,8 and 10 validate the statement "with the

increase in version of the software the bugs are reduced, hence the maintenance cost is also reduced. The stabilization phase only focuses on the reliability of the product and tries to make the product bug free. Thus better coding can result in better quality of the software and reduce the maintenance costs.

V. CONCLUSION AND FUTURE WORK

It is a well-known fact that the software maintenance is costly and effort intensive. Therefore, a software system should be maintainable. A lot of research has been done in the object-oriented systems written in different languages while less in software systems written in python. This paper analyzes the change in the bugginess of the python software system i.e., Trac. The results show that the patch uplifts not only increase the coding standards and the number of modules but fixes the bugs as well achieve the better quality of the software. A huge number of projects are developed using the python language paving the way to the researchers to analyze the effectiveness of these software. Not only the error-prone modules are troublesome, but there may be other factors as well that degrade the performance of the software. This work can be extended by evaluating the error prone modules in the software, analyzing which priority bugs involve more change in the lines of code and finding the correlation between the change in the number of lines of code due to fixing a bug and the number of bugs reported.

VI. REFERENCES

[1] Abbes, M., Khomh, F., Gueheneuc, Y. G., & Antoniol, G. (2011). An empirical study of the impact of two antipatterns, Blob and Spaghetti Code, on program comprehension. *15th European Conference on Software Maintenance and Reengineering (CSMR)* (pp. 181-190). Oldenburg: IEEE.

[2] Chen, Zhifei & Chen, Lin & Ma, Wanwangying & Xu, Baowen. (2016). Detecting Code Smells in Python Programs. 18-23. 10.1109/SATE.2016.10.

[3] Chen, Zhifei & Chen, Lin & Ma, Wanwangying & Xu, Baowen. (23-09-2017)Understanding metric-based detectable smellsin Python software: a comparative study.*In Information and Software Technology*.

[4] Dag I.K. Sjøberg,Bente Anda,Audris & Mockus.(2012)Questioning Software Maintenance Metrics:A Comparative Case Study.In *"Empirical Software Engineering and Measurement (ESEM), 2012 ACM-IEEE International Symposium on*.

[5] IEEE Std. 1219 (1993).: Standard for Software Maintenance. *IEEE Computer Society Pres.*,

[6] Marco,Le & Foutse (26 september 2017)“Is it safe to uplit this patch”arXiv:1709.00852v1[cs.SE] 26 Sep2017

[7] Moritz Beller, Alberto Bacchelli, Andy Zaidman & Elmar Juergens (2014).Modern code reviews in open-source projects: which problems do they fix?.*Proceedings of the 11th Working Conference on Mining Software Repositories*

[8] Nicole,Vadim (2016).Does python smell like java. In:*The art of Science and Engineering of programming.Vol I*,no.2,2017 article II;29 pages

[9] Orrù, Matteo & Tempero, Ewan & Marchesi, Michele & Tonelli, Roberto & Destefanis, Giuseppe. (2015). A Curated Benchmark Collection of Python Systems for Empirical Studies on Software Engineering.10.1145/2810146.2810148.

[10] Varuna,Ganeshan ,Singhal "Developing Software Bug Prediction Models Using Various Software Metrics as the Bug Indicator"s.(IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 6, No. 2, 2015.

[11] W. Ma, L. Chen, X. Zhang, Y. Zhou & B. Xu, "How Do Developers Fix Cross-Project Correlated Bugs? A Case Study on the GitHub Scientific Python Ecosystem," *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*, Buenos Aires, 2017,pp.381-392.doi: 10.1109/ICSE.2017.42

[12] Castelluccio, Marco & An, Le & Khomh, Foutse. (2017). Is it Safe to Uplift this Patch?: An Empirical Study on Mozilla Firefox. 411-421. 10.1109/ICSME.2017.82.

[13] Pylint (analyzes Python source code looking for bugs and signs of poor quality)". Logilab.org. 2006-09-26. Retrieved 2018-03-01.

[14] Pylint User Manual – Pylint 2.0.0 documentation". Docs.pylint.org. Retrieved 2018-03-01.

[15] W. Li and S. Henry, "Maintenance metrics for the object oriented paradigm," in First International Software Metrics Symposium, 1993.

[16] M. Catwright and M. Shepperd, "An empirical investigation of an object oriented software system," IEEE Transactions on software engineering, vol. 26, no. 8, pp. 786-796, 2000.

[17] K. Dhambri, H. Sahraoui and P. Poulin, "Visual detection of design anomalies," in 12th European Conference on Software Maintenance and Reengineering (CSMR 2008), 2008.

[18] M. Mäntylä, "Bad smells in software-a taxonomy and an empirical study," Helsinki University of Technology, 2003.

[19] V. R. Basili, L. C. Briand and W. L. Melo, "A validation of object oriented design metrics as quality indicators," IEEE Transactions on software engineering, vol. 22, no. 10, pp. 751-761, 1996.

