An Empirical Study on relation between Design-Defects and C&K Metrics Suite

Presented by
Team 10

Abstract—In this study we investigated the practicality of implementing object-oriented (OO) design metrics for the purpose of detecting presence of design defects. To perform our research, we considered three large-sized professionally developed tools (Ant, Mockito and Wildfly) and CK metrics suite is used. Like majority of the papers, we use logistic regression as a detection strategy and observe that, the metrics do not perform well at the hypothesized task.

Keywords : Object-oriented design metrics, code smell, design defect prediction model, CK metrics.

1 Introduction

1.1 Motivation

As the software evolves, the code and the architecture become more complex, which can inevitably induce defects in the system. Finding defects later in the developmental stages can increase the cost of software remodeling. Software developers usually use various metrics for determining the quality of the system. Using these metrics for predicting faults during development as well as every stage can increase the efficiency of the overall development process.

1.2 Goal

Our goal is to investigate correlation between CK metrics and design defects by using multiple kinds of code smells.

1.3 Contribution

In our study we have used publicly available data sets (Open Source Projects) like Ant, Mockito and WildFly. Moreover, we are adding new evidence that can correlate between CK metrics and Design-Defects. In addition to adding more valuable practical findings in this area where shared-knowledge is limited or unexplored.

1.4 Relevance

In Software Maintenance change made on a system after its delivery, takes up to 90% of all total cost of a typical software. Introducing new functionalities, altering defects, and changing the code to improve its quality are major parts. There has been much research concentrating on the study of bad design practices, also called smells, defects, antipatterns or anomalies in the literature. Keeping these issues in mind, in our research we have tried to identify the correlation between CK metrics and design defects to build a prediction model that forecasts design defects.

In this research, we conducted yet another empirical validation of OO class metrics. However, while most other studies have focused on the analysis on a solitary version, or release, of a software system, we focused on empirical validation of the software produced by a highly iterative software development process. Additionally, we have taken three distinct adaptations for the three different programming projects. The important question we wished to answer is, Can the OO metrics suites, we employed, identify design-defects classes in multiple, sequential releases of softwares, Ant, Mockito and WildFly.

1.5 Organisation

The rest of this paper is organized as follows :

- **Section 2:** In this section we have summarized different the research paper. And also, we have mentioned similarities and difference, Limitations and Improvements.
- **Section 3:** Study Definition, here all the information regarding the project selected, version, length of the project and etc., has been described. Moreover, the formal definitions of out hypothesis, have been described here.
- **Section 4:** Analysis Method, in this section we have discussed about the statistical methods we have applied on our data, which can be useful in proving or rejecting the hypothesis.
- **Section 5:** Study Results, in this section we have written if our hypothesis has been held true or false.
- **Section 6:** Discussion, in this we have discussed about the statistical results that we have obtained in Section 4.
- **Section 7:** Threats to Validity, in this section we have discussed about Construct Validities, External Validity and Conclusive Validity.
- **Section 8:** Conclusion
- **Section 9:** References

Source for mining repository has been selected as GitHub adhering, as it is a very well known and an extensively used revision control system.

2 Related Work

2.1 Summary

All the research work studied and presented here share a common goal, to investigate correlation between CK metrics and design defects using code smells for defective classes, an attempt is also made for building a prediction model for design-defect prediction using the conclusions derived in each paper. As the software evolves, the code and the architecture become more complex, which can inevitably induce defects in the system. Finding defects later in the development stages can increase the cost of software remodeling. Software developers usually use various metrics for determining the quality of the system. Using these metrics for predicting design-defects during development as well as every stage can increase the efficiency of the overall development process [12]

Satwinder Singh [15], used Analyst4J tool for detecting metrics as well as code smells, for determining faulty classes, while, in paper 'Software bug prediction using object-oriented metrics' [14], Ckjm tool is used for CK metrics collection. Method for code smell detection is not specified. Victor R. Basili [12], used a manual approach, the study was performed on codes written in C++ language by various students of differing skill-sets and skill levels, data about errors was collected during the testing phase and fixes during the repair phase. GEN++ tool was used for collecting CK metrics. All the papers had different ways to collect the data, but used the same logistic regression model. If the count of design defects or smell detected was greater than 0, it was labeled 1 else 0. This is because logistic regression is the most appropriate model for binary classification and fault detection is a binary classification problem, hence same methodology is shared among all the papers.

Bayesian inference was performed on 2 out of 6 metrics which showed a threshold for both metrics, which is a posterior probability of occurrence of error. Likelihood probability of error increasing is concluded at 40% with density 6 for CBO and 35% with density 9 for NOC [15]. Regression Analysis was performed and using P-values for each metric, following are the metrics and their category for design-defect prediction, DIT, RFC, NOC : very significant, WMC, CBO : significant and LCOM : insignificant [12]. From these studies we can safely conclude that though the final metrics from the results may variate, the CK metrics suite is significant in terms of design-defect prediction.

2.2 Comparison with the study proposed

Similarities & Differences: The paper work of [12] is not shown in table as it is very similar to the work by Satwender[15].

	Basilli [12]	Satwender [9]	Our work
Suite of Metrics	CK	CK	CK
Metrics tool	Gen++	JDeodarant.	Understand.
Language	C++	Java	Java
Dependent Variable type	Code faults and coding errors.	Implementat-ion Code Smells.	Design and Implementation smells.
Data Collection method	Manual Paper based	Analyst4J tool.	Designite Java.
Model	Logistic Regression.	Bayesian Inference	Logistic Regression.
Nature	Binary.	Binary	Binary

Table 1

The approach for classifying errors/smells into 1 and 0 is shared among all the papers and our work. A significant difference in this paper is that we are considering both the design and implementation code smells for detecting presence of defects. Since we are using the latest automated tools we expect a much better quality in terms on analysis.

Limitations : The study in paper is performed on systems developed by students, which are taken as study subjects. These systems would consist of defects and code smells which would have not been present if professionally developed systems were used in the first place. Additionally, defective classes are being defined during testing manually which might incorporate errors induced by humans[12]. Results were concluded, and analysis was performed on 2 out of 6 metrics in paper [15] which does not tell anything about the involvement of other metrics in prediction.

Improvements: Our study will be done on professionally developed and well-known systems, we select 3 softwares in addition to three release for each product. Hence the analysis we perform can be said to be relevant in real working environment with some extent of certainty. These different softwares simulate different work culture and environments. Also, we are using specialized and updated software's for the entire process which might improve the results of analysis. Inclusion of both implementation and design code smells, adheres to a stricter defect detection ruleset.

3 Study Definition

Objective of this research was to find Empirical evidence of the association between the bad-smells and class error probability. In this study they designed a Bayesian inference for individual metrics which provide posterior probability for defect occurrence[15].

In this study, we collected data about design-defects found in object- oriented classes. Based on these data, we verified how much design-defects are influenced by internal (e.g., size, cohesion) and external (e.g., coupling) design characteristics of OO classes [12].

Relationship between object-oriented metrics and defect prediction was studied in terms of code smells by, computing the accuracy of the proposed model on different datasets using logistic regression [14].

3.1 Projects

Projects selected are all based on Java, as most of the open source software's are available in this language and codes written in java require Object Oriented concepts to be implemented implicitly and it is one of the most extensively used and documented languages known. All the projects selected have more than 300k lines of code which increases the probability of the code being defective or consisting of defects in general.

Project Selected	Tools and plug-ins [6]
• Ant • Mockito • Wildfly	• Understand 5.0.9 by Scitools. • Jupyter notebook. • DesigniteJava.

Table 2

Revisions Selected for studying the most desirable projects :

Mockito	Ant	Wildfly
• 2.25.0	• 1.01.0	• 16.0
• 2.21.1	• 1.05.2	• 12.0
• 2.17.6	• 1.10.5	• 8.0

Table 3

3.2 Research Questions

In our study, we have applied six different CK metrics with the focus on how accurately they are able to predict the design-defects. In the later part, we are testing our results against null hypothesis, using logical regression. To make it easier, we haven't considered defect severity while modeling our prediction model. Here we are trying to answer the below given question.

- Are CK metrics useful in predicting the existence of design defects?
  Null hypothesis : C&K metrics has no correlation with design
  Alternative hypothesis: C&K metrics is correlated with design defects.
- Which CK metrics are more likely to predict design-defects?
  Null hypothesis : No CK metrics is useful in predicting design defects.
  Alternative hypothesis: Some, if not all, CK metrics are useful in predicting design defects.

3.3 Variable Selection

Independent Variables : Here we are taking different CK metrics as Independent variables. More specifically, WMC, DIT, NOC, CBO, RFC and LCOM. Dependent Variables :We are studying and building a defect prediction model, with dependent variable being presence of code smell i.e. (if code smell >0 : classify as 1, else : 0).

3.4 Definitions of Metrics

3.4.1 C&K Metrics

The C&K Metric Suite has been implemented to analyse design-defect in the projects under consideration [7] [4]. Chidamber and Kemerer (CK) et al. [1] gives the formal definition of the metrics as follows :

Weighted Methods per Class (WMC): This measures the sum of complexity of the methods in a class. The complexity of the class may be calculated by the cyclomatic complexity of the methods. The high value of WMC indicates that the class is more complex as compare to the low values.This measures the sum of complexity of the methods in a class. The complexity of the class may be calculated by the cyclomatic complexity of the methods. The high value of WMC indicates that the class is more complex as compare to the low values.

Depth of Inheritance Tree (DIT): DIT metric is used to find the length of the maximum path from the root node to the end node of the tree. DIT represents the complexity and the behavior of a class, and the complexity of design of a class and potential reuse. Subsection text here.

Number of children (NOC): According to Chidamber and Kemerer, the Number of Children (NOC) metric may be defined for the immediate sub class coordinated by the class in the form of class hierarchy. These points are come out as NOC is used to measure that "How many subclasses are going to inherit the methods of the parent class". The greater the number of children, the greater the potential for reuse. The greater the number of children, the greater the likelihood of improper abstraction of the parent class.

Coupling between Objects (CBO) : CBO is used to count the number of the class to which the specific class is coupled. The rich coupling decreases the modularity of the class making it less attractive for reusing the class and more high coupled class is more sensitive to change in other part of the design through which the maintenance is so much difficult in the coupling of classes.

Response for class (RFC): The response set of a class (RFC) is defined as set of methods that can be executed in response and messages received a message by the object of that class. Larger value also complicated the testing and debugging of the object through which, it requires the tester to have more knowledge of the functionality.

Lack of Cohesion in Methods (LCOM) : This metric is used to count the number of disjoints methods pairs minus the number of similar method pairs used. Since cohesiveness within a class increases encapsulation it is desirable and due to lack of cohesion may imply that the class is split in to more than two or more sub classes. Low cohesion in methods increase the complexity, when it increases the error proneness during the development is so increasing.

3.4.2 Bad Smells

Design smells	Implementation smells
Abstraction Design smells	Long Method
Encapsulation Design smells	Complex Method
Modularisation Design smells	Long Parameter List
Hierarchy Design smells	Long Identifier
	Long Statement
	Complex Conditional
	Virtual Method Call from Constructor
	Empty Catch Block
	Magic Number
	Duplicate Code
	Missing Default

Table 4

Bad smell means a piece of code in program that signifies the design problem in software code structure. Bad smell is a design defect, not a run time error. These bad smells require refactoring to improve the code quality [8]. We have used DesigniteJava tool to detect bad smells in the projects. The following are the list of bad smells

3.4.3 Design Smells [17]

Abstraction Design Smells: The definition of ab-straction design principle is given as follows : "The principle of abstraction advocates the simplification of entities through reduction and generalization : reduction is by elimination of unnecessary details and generalization is by identification and specification of common and important characteristics." The following are the smells detected that violates the principle of abstraction : Imperative Abstraction, Multifaceted Abstraction, Unutilized Abstraction, Duplicate Abstraction.
Encapsulation Design Smells: The definition of encapsulation design principle is given as follows :

"The principle of encapsulation advocates separation of concerns and information hiding through techniques such as hiding implementation details of abstractions and hiding variations". The following are the smells detected that violates the principle of encapsulation :
Deficient Encapsulation, Unexploited Encapsulation
Modularization Design Smells : The definition of modularization design principle is given as follows : "The principle of modularization advocates the creation of cohesive and loosely coupled abstractions through techniques such as localization and decomposition." The following are the smells that violates the principle of modularization: Broken Modularization, Insufficient Modularization, Hub-like Modularization, Cyclically-dependent Modularization.
Hierarchy Design Smells : The definition of modularization design principle is given as follows : "The principle of hierarchy advocates the creation of a hierarchical organization of abstractions using techniques such as classification, generalization, substitutability, and ordering.". The following are the smells that violates the principle of hierarchy : Wide Hierarchy, Deep Hierarchy, Multipath Hierarchy, Cyclic Hierarchy, Rebellious

Hierarchy, Unfactored Hierarchy, Missing Hierarchy.

3.4.4 Implementation Smells

Identical or very similar code exists in more than one location.
Duplicate Code: Identical or very similar code exists in more than one location.
Long Method: A method, function, or procedure that has grown too large.
Long Parameter List: More than three or four para-meters for a method.
Magic Number: A magic number is a numeric value that's encountered in the source but has no obvious meaning. This "anti-pattern" makes it harder to understand the program and refactor the code.
Missing Default Case : A default case is missing in a case or selector statement.
Long Statement: The code contains long statements (that typically do not fit in a screen).
Complex Conditional: The longer a piece of code is, the harder it's to understand. Things become even more hard to understand when the code is filled with conditions.
Virtual Method Call from Constructor: Constructors of derived types are called after constructors of base types. On the other hand, calls to virtual methods are always executed on the most derived type. This means that if you call a virtual member from the constructor in a base type, each override of this virtual member in a derived type will be executed before the constructor of the derived type is called. If the override in the derived type uses its members, this can lead to confusion and errors.
Empty Catch Block: An empty catch block code smell undermines the purpose of exceptions. When an exception occurs, nothing happens, and the program fails for an unknown reason. The application can be in an unknown state which will affect the subsequent processing.
Long Identifier: Using too long identifiers may decrease the readability a lot.

3.5 Data Collection

3.5.1 Internal Metrics Implementation [5] [18]

CBO (Count Class Coupled): CBO represents the number of classes coupled to a given class. According to this metric "Coupling Between Object Classes" (CBO) for a class is a count of the number of other classes to which it is coupled. Theoretical basis of CBO relates to the notion that an object is coupled to another object if one of them acts on the other, i.e. methods of one-use methods or instance variables of another (Chidamber et. al. 1994). As Coupling between Object classes increases, reusability decreases, and it becomes harder to modify and test the software system. Chidamber and Kemerer state that their definition of coupling also applies to coupling due to inheritance, but do not make it clear if all ancestors are involuntarily coupled or if the measured class has to explicitly access a field or method in an ancestor class for it to count. In general, the definition of the CBO is ambiguous, and makes its application tough (Mayer et. al. 1999). **NOC (Count Class Derived):** Number of children (NOC) of a class is the number of immediate sub-classes subordinated to a class in the class hierarchy. Theoretical basis of NOC metric relates to the notion of scope of properties. It is a measure of how many sub-classes are going to inherit the methods of the parent class (Chidamber et. al. 1994).
WMC (Sum Cyclomatic): It is the sum of the cyclomatic complexity of each method for all methods of a class. If a Class C, has n methods and c1, c2 …cn be the complexity of the methods, then

WMC(C)= c1 + c2 +… + cn.

The specific complexity metric that is chosen should be normalized so that nominal complexity for a method takes on a value of 1.0. If all method complexities are considered to be unity, then WMC = n, the number of methods. WMC break an elementary rule of measurement theory that a measure should be concerned with a single attribute (Chidamber et. al. 1994).

DIT (Max Inheritance Tree): The DIT for class X is the maximum path length from X to the root of the inheritance tree. According to this metric Depth of Inheritance (DIT) of a class is "the maximum length from the node to the root of the tree. The definition of DIT is ambiguous when multiple inheritance and multiple roots are present.

LCOM (Percent Lack of Cohesion): The LCOM metric value is calculated by removing the number of method pairs that share other class field from number of method pairs that does not share any field of other class. Consider a Class C1 with n methods M1, M2 ..., Mn. Let Ij = set of instance variables used by method Mi. There are n such sets {I1},{I2}... {In}. Let P = {(Ii,Ij) |Ii ∩ Ij = ∅ } and Q = (Ii ,Ij) |Ii ∩ Ij ≠ ∅ . If all n sets I1,I2... In. are ∅ then let P = ∅ [4]. Lack of Cohesion in Methods (LCOM) of a class can be defined as :

LCOM = |P|- |Q|, if |P|>|Q|
LCOM = 0 otherwise

The high value of LCOM indicates that the methods in the class are not really related to each other and vice versa. According to above definition of LCOM the high value of LCOM implies low similarity and low cohesion, but a value of LCOM = 0 doesn't implies the reverse (Mayer et. al. 1999).

Cyclomatic complexity metric used for measuring the program complexity through decision-making structure such as if-else, do-while, foreach, goto, continue, switch case etc. expressions in the source code. Cyclomatic complexity only counts the independent paths by a method or methods in a program. Complexity value of program is calculated using following formula :

V (G) =E -N +P,
where
V (G) - Cyclomatic complexity
E - No. of edges of decision graph
N - No. of nodes of decision graph
P - No. of connected paths

If there is no control flow statement like IF statement, then complexity of program will be 1. It means there is single path for execution. If there is single IF statement then it provides two paths : one for TRUE condition and other for FLASE condition, so complexity will be 2 for it with single condition.

RFC (Count Decl Method All): According to this metric Response for a Class (RFC) can be defined as |RS |, where RS is the response set for the class. The response set for the class can be expressed as :

RS = {M} ∪ all i { R i }

where, {Ri} = set of methods called by method i and {M} =set of all methods in the class.

The response set of a class is a set of methods that can potentially be executed in response to a message received by an object of that class. (Henderson et. al. 1991) But here the point to be noted is that because of practical considerations, Chidamber and Kemerer recommended only one level of nesting during the collection of data for calculating RFC. This gives incomplete and ambiguous approach as in real programming practice there exists "Deeply nested call-backs" that are not considered here. If CK intend the metric to be a measure of the methods in a class plus the methods called then the definition should be redefined to reflect this (Mayer et. al. 1999).

3.5.2 Challenges Faced

Scitool's Understand software is used for metrics collection, it provides the data of metrics not only on classes, but also on files and directory, resulting to an incorrect analysis, so extra cleaning had to be done on the data. Software systems selected for study are well known, widely used and large systems, retrieving data on personal laptops having lower RAM memory led to multiple crashes and was time consuming to some extent. Arrangements had to be done for a laptop with RAM size of at least 12 GB for metrics collection. Initially we were using only CK metrics and with single criteria for each type of code-smell category, as our knowledge was limited. After some discussions and extra study, we concluded that there should be multiple criteria for each type

of code-smell detection. We finally settled in for the use of an external tool, but we propose to study further and develop our own set of criteria for code-smell classification as a separate work. Data cleaning and Preparation was a major challenge as the output of Designite java separates the Packages, Methods and Types of classes then detects code smells. This step composes 80% of the work time. The output from Designite java was to be manipulated in a way that it could be merged with the output from Scitool's Understand.

3.5.3 Correctness of Metrics

C&K metrics are widely used and have been validated multiple times, so we mention the hypothesis for the construct of these metrics instead, with regards to our proposed study.
WMC: A class with significantly more member functions than its peers are more complex and, by consequence, tends to be more design-defect-prone [12].
DIT: A class located deeper in a class inheritance lattice is supposed to be more design-defect-prone because the class inherits a large number of definitions from its ancestors. In addition, deep hierarchies often imply problems of conceptual integrity, i.e., it becomes unclear which class to specialize from in order to include a subclass in the inheritance hierarchy [12].
NOC: Classes with large number of children (i.e., subclasses) are difficult to modify and usually require more testing because the class potentially affects all of its children. Furthermore, a class with numerous children may have to provide services in a larger number of contexts and must be more flexible. We expect this to introduce more complexity into the class design and, therefore, we expect classes with large number of children to be more design-defect-prone [12].
CBO: Highly coupled classes are more design-defect-prone than weakly coupled classes because they depend more heavily on methods and objects defined in other classes [12].
RFC: Classes with larger response sets implement more complex functionalities and are, therefore, more design-defect-prone [12].
LCOM: Classes with low cohesion among its methods suggests an inappropriate design (i.e., the encapsulation of unrelated program objects objects and member functions that should not be together) which is likely to be more design-defect-prone [12].

3.6 External Metrics Collection

3.6.1 Tools used

- Scitools Understand : For collecting C&K metrics.
- DesigniteJava : For Code-Smells
- Jupyter Notebook : For data analysis.

3.6.2 Tools Explored

- BugZilla
- PMD
- JDeodrant
- InFusion

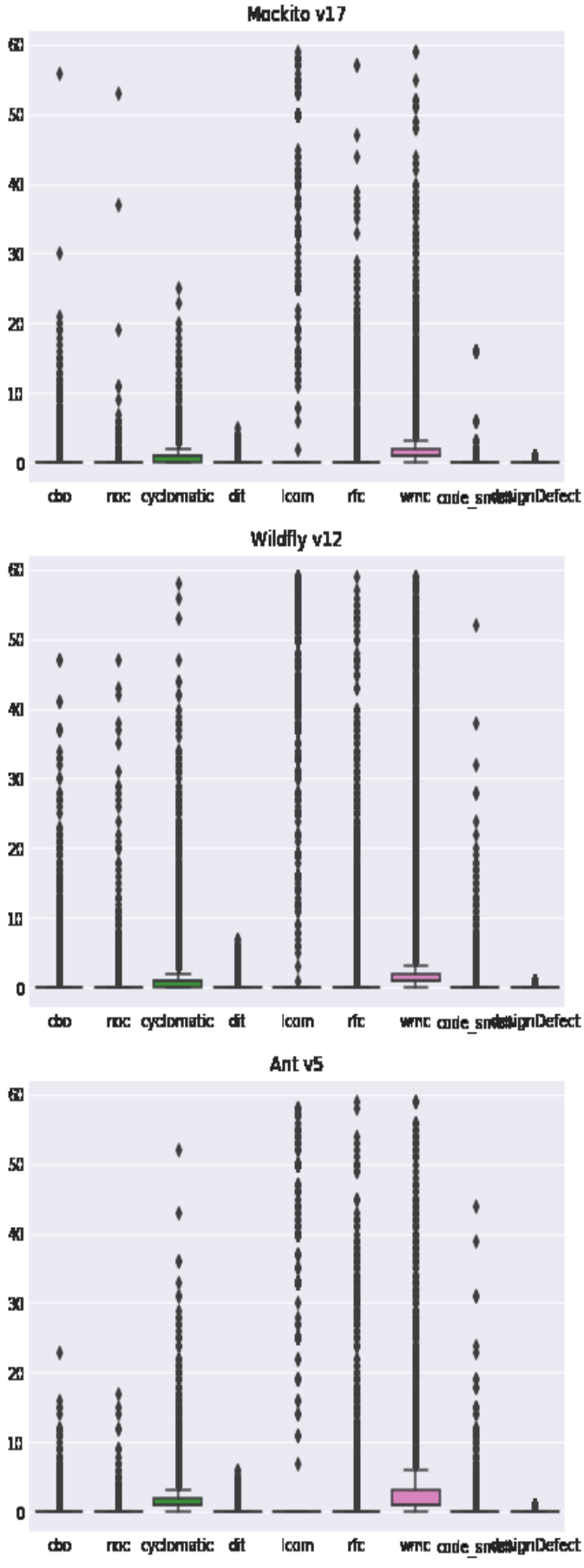
3.7 Data Collection

Before performing statistical analysis, the most important task is to clean the data. Since the data is produced by a software there should not be any missing values, hence the data is clean in terms of quality. But we have used different softwares for metrics data and code smell data collection. Check for the format outputted, as these differ a custom code was written to combine both the data-sets in a standard form. Then the major consideration is the imbalance in dataset.

Count of Design defects	
0 (Absence)	2,70,434
1 (Presence)	14000

Table 5

We have performed all the analysis on the raw data-set then performed modelling after dealing with imbalance in the data using Smote treatment, which first performs oversampling on minority class and then under-sampling on majority class. This is needed because if we use the imbalanced data, then the accuracy of the model would always be greater than 90%, which is a sign of an over-fitting. The resulting model would miss-classify many 1's as 0's.



We observe that there are a lot of outliers present in the data, we concluded whether the outliers are relevant or not in further sections of the paper.

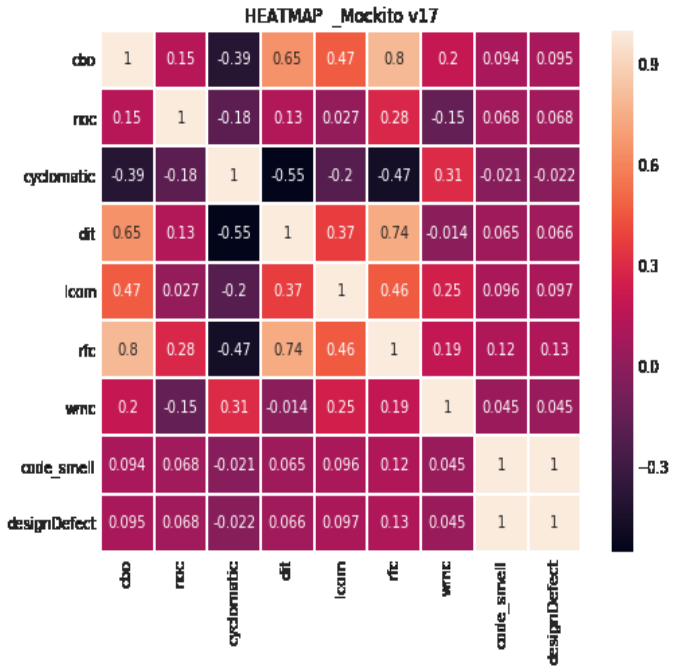
4 Analysis Method

4.1 Statistical Tests

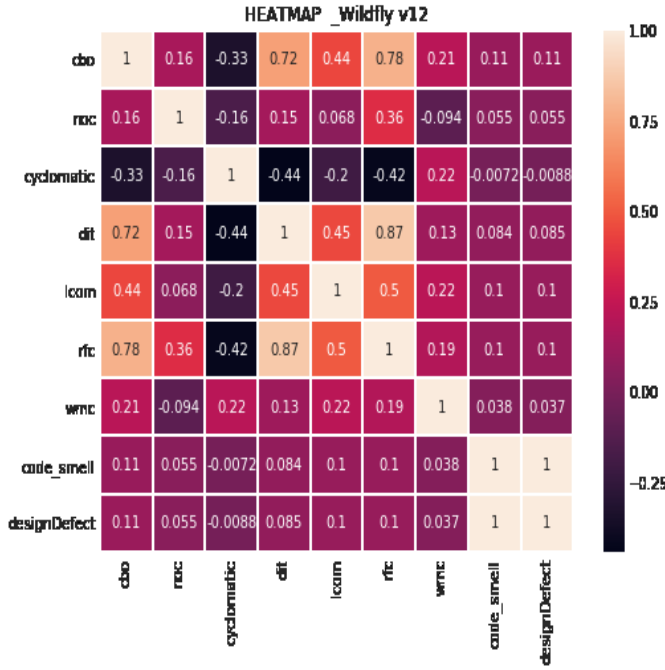
We are studying and building a design defect prediction model, with dependent variable being presence of defect i.e. (if code smell >0 : classify as 1, else : 0). Hence, we need a statistical test for binary variables. We are using logistic regression as we have only two outputs, i.e. binary.

4.2 Variables with High Correlation

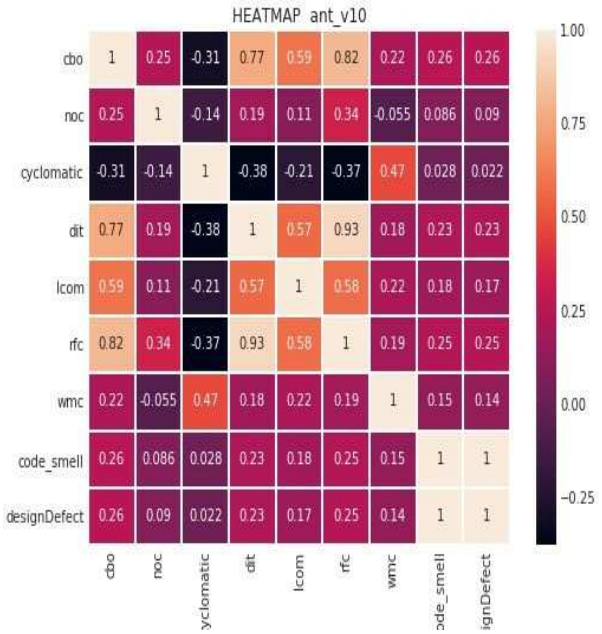
From the open source projects taken under consideration, the following are the high correlations that we observe. Variables executing correlation >50



Mockito
CBO vs RFC (80%) CBO vs DIT (65%)
RFC vs DIT (74%)



WildFly
CBO vs RFC (78%) CBO vs DIT (72%)
RFC vs DIT (87%)



Ant
CBO vs RFC (80%) CBO vs DIT (76%)
RFC vs DIT (94%)

4.3 Regression with all independent variables in the model :

We have created a full set of regression plots for all possible combinations of the metrics and presence of defect using normal and log transformed data available at Plots folder in our Github repo. The regression plots tell us that there is no linear correlation between the independent and dependent variables in base data nor log transformed version. Hence, we use glm for building our predictive model.

```
Coefficients:
             Estimate Std. Error z value Pr(>|z|)
(Intercept) -0.5483330  0.0186920 -29.335  < 2e-16 ***
noc          0.0750264  0.0220516   3.402 0.000668 ***
cyclomatic   0.2215726  0.0087300  25.381  < 2e-16 ***
dit         -0.0005539  0.0366738  -0.015 0.987950
lcom         0.0183649  0.0019974   9.194  < 2e-16 ***
wmc         -0.0861789  0.0059489 -14.487  < 2e-16 ***
cbo          0.1637456  0.0131096  12.491  < 2e-16 ***
rfc          0.0628512  0.0071986   8.731  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

We combine all 3 datasets into one for producing a predictive mode. We compare the P-value to a significance level to make conclusions about our hypotheses. The above significance codes display levels of significance. We use the widely accepted threshold value of alpha for this purpose, set at 0.5 on positive and negative sides of the normal distribution. If the P-values lies between 0 and alpha, it is accepted as significant. Closer to 0, higher the significance. We accept the variables with *** as they execute a P-value between (0.000, 0.001). Hence, we can discard 'dit' as its significance is 0.9, which shows that it does not contribute much to our model.

4.4 Hypothesis

Null hypothesis C&K metrics has no correlation with presence of design defects. **Alternative hypothesis** C&K metrics is correlated with presence of design defects. For the purpose of deciding on a hypothesis we developed 3 variants of the model.

	Without outliers	RawData (Imbalanced)	Smote Treatment (Balanced)
Accuracy	94.41%	70.04%	59.19%
Sensitivity	99.7%	93.26%	71.44%
Specificity	2%	23.95%	42.86%
P-value	0.8134	3.823e-13	0.0014

Table 6

The table clearly tells us that outliers are meaningful and should be included in building the model. Based on all the observations, we can safely reject the null hypothesis.

5 Discussion

The Model after Smote balance treatment has a 59% prediction accuracy, with specificity of 42% and sensitivity of 71%, shows that the model is not over-fitted. After eliminating insignificant variables, we see that the model 2 (Section 4.4) is significant compared to model 1 (Section 4.3). Also the raw model, without any treatment has a good P-value signifying the model is useable, even though the specificity and sensitivity tells us the model is not the best. In both cases we can can say there is a correlation between Ck metrics and design defects.

Hence we can safely reject the null hypothesis.

6 Threats to Validity

6.1 External validity

It is "the extent to which the results of a study can be generalized to other situations and to other subjects"

- The projects which we have studied, have characteristics of being large, developed in java and open source. Software Projects which have similar characterises should generate similar results when applied our design-defect prediction model.
- In our study, we have taken considerable amount of design defects detection technique. If we would have chosen different techniques the result may differ from our result. However, we have covered the most significant techniques to detect them
- For our research, we have taken the tools which have been developed in Java language, because of the familiarity with the java programming. The project selection here is based on programming language, which could affect these metrics by changing the coding language. These metrics can be different according to different programming language and programming styles.
- We have practiced the tools which have been developed by the professionals. However, in the study from which we referred, has considered small projects and not professionally developed. In similar scenarios this can raise or lessen the value of the metrics. Our study has left analysis on small projects and it would be interesting to study in future.
- Moreover, sofwares, which we have taken for our analysis, are well-tested and designed significantly well. In this way our research can limit the applicability to the softwares which are in early stages of their development or not tested regressively or not designed to the best of the capabilities.
- The timespan between the versions, which we have considered for softwares, is of considerable amount. Between closer versions, there won't be much difference regarding code structure and code complexity. So, comparing these nearer versions can significantly affect to the values of these metrics.

6.2 Construct Validity

It is "the degree to which a test measures what it claims, or purports, to be measuring."

- The approach for classifying defect/smell into 1 and 0 is shared among all the papers and our work. Here we are not considering the severity of defect/smell. For future work, it would be interesting to link our analysis with weighted defects/smells.
- We found the bad-smells using designite java tool for smell and metrics detection. Previous work suggests that this kind of

tool, does not necessarily generates unbiased results for detecting the bad-smells. We might not be able to detect all the bad-smells automatically. Going ahead, we can use different bad-smells detection tools and can compare the collected data.

6.3 Conclusive Validity

It is "the degree to which conclusions we reach about relationships in our data are reasonable."

- As a conclusion, we have negated the 'null-hypothesis', which signifies that there is considerable correlation between the response and the predictor variables.

7 Conclusion

In our study, we tried to find the correlation between design defects and CK metrics using code smells. We concluded that there is correlation between CK metrics and presence of design defect. Moreover, our study covers, how the internal and external factors affect the proneness of design-defect. We also concluded that other than DIT metric all other metrics have lower P-Value and are useful at detecting design defects. Finally, our results show that we can use our prediction model, built in Section 3 & 4 combinedly, to predict the design defects in large projects.

8 Future work

,

- We can apply our prediction model on smaller and less complex projects, which are in earlier phase of the development.
- We can use our prediction model for more versions. So that we can also analyze the design defects in accordance to the software evolution
- In our research we have only taken into the considered the presence of the design defect. In future, we can also consider different properties of the defect as well. We can modify our prediction model and also take into the account the defect type. So, we can also predict which metric is more useful to predict which type of defects.
- Moreover, it would be interesting to add the design defect severity while developing the 'Design Defect Prediction Model' and we can check the abilities of our prediction model.

References

[1] F. Chris C. Shyam Kemerer. "A Metrics Suite for Object- Oriented Design". In: *M.I.T. Sloan School of Management* (1993), pp. 53–315.

[2] P. Nesi E Fioravanti. "A Study on Fault-Proneness Detection of Object-Oriented Systems". In: *Proceedings Fifth European Conference on Software Maintenance and Reengineering* (2002), pp. 2231–5268.

[3] Radu Marinescu. "Measurement and Quality in Object-Oriented Design". In: *PhD thesis, Politechnica University of Timisoara* (2002).

[4] Ramanath Subramanyam and M.S. Krishnan. "Empirical Analysis of CK Metrics for Object-Oriented Design Complexity : Implications for Software Defects". In: *IEEE Transactions on Software Engineering* 29.4 (2003).

[5] Pacific Asia Conference on Information Systems. "A Critical Suggestive Evaluation of CK Metric". In: (2005).

[6] Jonas Lundberg Rüdiger Lincke and Welf Löwe. "Comparing Software Metrics Tools". In: *Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis* (2008).

[7] Mohammad Zulkernine Istehad Chowdhury. "Using complexity, coupling, and cohesion metrics as early indicators of vulnerabilities". In: *Journal of Systems Architecture* (2011), pp. 294–313.

[8] Sukhdeep Kaur1 Dr. Raman Maini. "Analysis of Various Software Metrics Used to Detect Bad Smells". In: (2016).

[9] Dr. Rodrigo Morales Alvarado. "Software Measurement PPTs ". In: ().

[10] Sanjay Kumar Dubey Amit Sharma. "Comparison of Software Quality Metrics for Object- Oriented System". In: *IJCSMS International Journal of Computer Sci-*

ence Management Studies, Special Issue of Vol. 12, June 2012 ISSN (), pp. 2231–5268.

[11] Victor R. Basili. "A Validation of Object-Oriented Design Metrics as Quality Indicators". In: ().

[12] Victor R. Basili. "A Validation of Object-Oriented Design Metrics as Quality Indicators". In: ().

[13] "DesigniteJava". In: (). DOI: https://github.com/tushartushar/DesigniteJava.

[14]] Software bug prediction using object-oriented metrics. "DHARMENDRA LAL GUPTA 1,2, * and KAVITA SAXENA 2". In: ().

[15] Heena Kapila Satwinder Singh. "Analysis of CK Metrics to predict Software Fault-Proneness using Bayesian Inference". In: *International Journal of Computer Applications* (), pp. 0975–8887.

[16] Evaluation of code smells and detection tools. "" In: (). DOI: https://doi.org/10.1186/s40411-017-0041-1.

[17] "Refactoring for Software Design Smells : Managing Technical Debt. "Ganesh Samarthyam, Girish Suryanarayana,Tushar Sharma". In: ().

[18] Analysis of Various Software Metrics Used to Detect Bad Smells. "Sukhdeep Kaur, Dr. Raman Maini". In: *The International Journal of Engineering* ().