ISSN: 0067-2904

# Defect Prediction Tool for Object-Oriented Software

**Rasha Gh. Alsarraj[1]***
*Department of Software, College of Computer Science and Mathematics, University of Mosul, Mosul, Iraq.*

Received: 27/2/2024 Accepted: 12/10/2024 Published: 30/10/2025

**Abstract**

Automated software defect prediction (ASDP) is a fundamental and vital action in the software progress domain. Nevertheless, software as a contemporary system is integrally huge and complex, with several associated metrics that capture diverse aspects of software components. Such a huge associated metrics number renders building a Software Defect (SD) prediction (SDP) model so complex. Therefore, selecting and identifying a metrics subset that enhances the SDP performance of the method is imperative. The chief goal of the current work is to build an ASDP (SDP) tool and assess its performance. The SDP tool calculates significant software (CK) metrics that measure code characteristics, for evaluating software quality and anticipating issues, and four basic transformations to predict (SD) and defect density (DD). The research also compares the initial values of extracted metrics from software to those resulting from the Log, Ln, Power, and Root transformations shown in the case study. In the suggested SDP tool, the kurtosis of the software metrics decreased, and predicting DC and DD became more precise. This helps increase software quality by enhancing the classes that include defects, the suggested SDP tool has a user-friendly interface and is easy to use.

**Keywords:** Basic transformations, Defect density, Metrics of CK, Software Defect, Threshold value.

## أداة تنبؤ بالعيوب للبرامج كائنية التوجه

**رشا غانم السراج**
قسم البرمجيات، كلية علوم الحاسوب والرياضيات، جامعة الموصل، نينوى، العراق

**الخلاصة**

يعد التنبؤ الآلي بخلل البرامجيات (*ASDP*) نشاطًا أساسيًا وحيويًا في مجال تطوير البرامجيات. ومع ذلك، فإن البرامجيات الحديثة ضخمة ومعقدة بشكل متكامل مع العديد من المقاييس المرتبطة التي تلتقط جوانب متنوعة لمكونات البرامجيات. هذا العدد الضخم من المقاييس المرتبطة يجعل بناء نموذج تنبؤ بخلل البرامجيات (*SDP*) معقدًا للغاية. لذلك، يعد اختيار وتحديد مجموعة المقاييس الفرعية التي تعزز طريقة أداء *SDP* أمرًا ضروريًا. الهدف الرئيسي للعمل الحالي هو بناء أداة للتنبؤ بخلل البرامجيات (*SDP*) وتقييم أداءها. تحسب أداة *SDP* المقاييس المهمة للبرنامج وهي مقاييس (*CK*) ، وأربعة تحويلات أساسية للتنبؤ بخلل البرامجيات (*SD*) وكثافة الخلل (*DD*). يقارن البحث أيضًا القيم الأولية للمقاييس المستخرجة من البرمجيات بتلك الناتجة عن

*Email: rasha.alsarraj@uomosul.edu.iq

تحويلات **Log** و **Ln** و **Pow** و **Root** الموضحة في دراسة الحالة. بعد استخدام أداة **SDP** المقترحة، انخفض تفرطح مقاييس البرمجيات، وأصبح توقع الفئات التي تحوي خلل (**DC**) وكثافة الخلل (**DD**) أكثر دقة، وهذا يساعد في تحسين جودة البرامجيات من خلال تحسين الفئات التي تتضمن عيوبًا. بالإضافة إلى سهولة استخدام أداة **SDP** وواجهة سهلة الاستخدام.

## 1. Introduction

SDP is the most active area of research in the engineering of software and shows a significant role in the quality of software assurance. The rising software dependency and complexity have augmented the struggle to deliver high quality, maintainable, and low-cost software, besides the possibility of creating Software Defects (SD). Typically, SD yields unexpected or incorrect findings and behaviours in unintended ways [1].

Defect Prediction (DP) is a very essential and crucial activity. Utilizing predictors of defects can minimize costs and expand the quality of software through defect-prone recognizing instances (modules) before testing, i.e., engineers of software can effectively optimize the restricted resource allocation for maintenance and testing. SDP [2, 3] can be performed through software determining cases such as class, procedure, file, package, or change degree as either non-defective or defective. The DP model is able to be constructed via utilizing several metrics of software and defect historical data gathered from the preceding release of similar projects [4, 5] or even diverse ones. This model will predict whether instances of software are faulty or not. With the model of prediction, software engineers can efficiently allocate the obtainable resources for testing on instances that are defective to improve the quality of software in the initial development phases of the life cycle. Nowadays, SDP is a common investigation matter in the topic of Software Engineering (SE) [6, 7].

SE contains many areas of knowledge, i.e., quality of software (SQ), SE Process (SEP), and SE Management (SEM). Within the framework of SEP, measurements with the software product are significant for determining the software efficiency processes. Among such measurements are total product defects and DD [8]. The term defect is used to denote diverse kinds of anomalies; nevertheless, such a term was not utilized as the sole one referring to an irregularity. Engineering standards and cultures have utilized numerous vocabularies with various denotations, i.e., error, computational error, defect, bug (or fault), and failure. An individual who makes an error leads to a defect. A defect is described as an imperfection or deficit in a work product that requires repair or replacement because it does not match its standards or specifications. DD is an extent for determining the software efficiency procedures. DD is defined as the entire defect summation subdivided by the software size [9]. The chief contributions of the current work are as follows:

1. An automated calculation of the metrics of CK for object-oriented software.
2. Build an automated defect and DD prediction tool for software.
3. Predict the defects of software utilizing four transformation methods according to the metrics of CK.

The current work is prepared as follows: In Sec. 2, a brief review of previous work on SDP is offered. Sec. 3 describes the metrics of CK. The suggested tool in is shown in Section 4 and the experiment and findings of the case study are in Sec. 5, before conclusions and upcoming works in Sec. 6.

## 2. Related Work

A lot of work is performed in the field of SDP to reduce development costs before the testing stage. Some related studies are discussed below:

In 2015, Jaechang and Sunghun [10] suggested novel methods, CLAMI and CLA, which display the DP potential on unlabeled data sets in an automated way with no necessity for manual effort. The key CLAMI and CLA idea methods are to label an unlabeled data set by utilizing the metric value magnitude. In their study, as empirical on 7 projects as open-source, suggested models resulted in comparable or better findings to characteristic DP models and other baseline methods in the majority of projects concerning f-measure, recall, and/or AUC, and they perceived that such models operate effectively with project data sets merely utilizing a major determined metrics subset that follows the usual defect-proneness dataset inclinations.

In 2015, Peng et al. [11] employed three kinds of forecasters, utilizing the software size of metric sets in three situations. At that time, validation, the acceptable predictor of performance is according to the Top-k metrics in statistical methods terms. Lastly, they tried to reduce the Top-k metric subset by eliminating metrics as redundant, and they verified that the least metric subset was stable with tests of 1-way ANOVA. This work was carried out on 34 versions of 10 projects as open-source offerings within the PROMISE repository. The outcomes mention that the predictors constructed by the least metric subset, or the Top-k metrics were able to offer an acceptable finding in comparison to predictors as benchmarks.

In 2016, Chakkrit et al. [12] investigated the performance of DP models where Caret (a parameter-based automated optimization technique) was performed. Via open-source fields, they found that Caret expands the AUC efficiency of DP models by 40% points; classifiers of Caret-optimal are at least as steady, as 35% of them are extra stable compared to predictors utilizing the settings as default; and Caret upsurges the possibility to produce a high-performing predictor by 83%a and achieve those parameter settings, which can certainly have a great influence on the DP model's performance.

In 2017, Shamsul et al. [13] suggested two novel models as hybrid SDPs for identifying significant metrics (attributes) utilizing a filter and wrapper technique combination. The suggested framework's performance was also confirmed using a statistical multi-variant process of quality control utilizing a multi-variant exponentially moving weighted average. The suggested framework reveals that the heuristic is hybrid and able to guide the process of metric selection in a computationally effective manner via intrinsically integrating characteristics into the wrapper from the filters and utilizing the advantages of the wrapper and filter methods.

In 2018, Soumi et al. [14] proposed a new model according to Non-linear Manifold Techniques of Detection for eliminating unrelated and undesirable features of high dimensional datasets via reduction of dimension with extra accuracy of prediction and improved quality of software. In the related work, a new step to attaining the objective was taken by creating a novel model according to nonlinear MDTs and comparing its efficiency with existing techniques of feature selection to identify the most precise DP method. The diverse performance categorizing methods with existing and new methods were assessed, compared, and tested statistically through the Friedman test and post-hoc analysis. The findings verified that the new model suggested according to nonlinear MDTs is superior in oriented performance in comparison to the level of accuracy of all other methods.

In 2019, Manjula et al. [15] presented a hybrid method by merging genetic algorithms (GA) for attribute improvement alongside deep neural networks (DNN) for categorizing. GA enhanced release was integrated, which comprises a novel chromosome design practice and computation of fitness function. The technique of DNN was improvised too, utilizing an auto-encoder as adaptive that offers better selected signification of software characters. The suggested hybrid improved efficiency approach because of optimization approach deployment is confirmed by case evaluations. A trial study was conducted regarding SDP by considering the PROMISE data set and utilizing the tools of MATLAB.

In 2020, NezhadShokouhi et al. [16] suggested an SDP method that utilizes the distance of Mahalanobis in feature and balancing extraction classes. In balancing classes, the distance of Mahalanobis is utilized to yield a varied artificial sample for a class as a minority (class as a defect), and for extraction attributes. The distance of Mahalanobis scale learning is utilized for projecting sample information to a novel space where samples per similar class are close to one another and diverse classes' samples are distant. To assess the suggested method's efficiency, we executed a chain of trials on 12 public SD data sets from the NASA warehouse through comparison with the baseline techniques regarding GM, F1, and MCC. According to such performance standards, the suggested method was better than alternative techniques as a result of the worthy qualities of the Mahalanobis.

In 2020, Junhua and Feng [17] suggested a method that implemented the law of power function characteristics for SDP. The approach kernel utilized the law of power function for describing distributions of metrics and transformation values set in the curvature maximal point of each law of power function curve as the value of the threshold for metrics labeling. They showed that their suggested method is possible and highly effective at DP with unlabeled data sets.

In 2021, Sultan [18] investigated the association between the size of the system (as a factor being contextual) and thresholds as a metric. He built models of prediction whose appraisal thresholds were solely based on the size of the system and evaluated their feasibility as a method of estimation of thresholds. The findings display that the suggested estimation threshold method is achievable, and it can attain correctness extraordinarily comparable to extra-complex threshold models.

In 2022, Dominik [19] assessed the overall usefulness of metrics in software for DP as well as the impact of individual metrics on the prediction. Software product metrics can be computed relatively quickly during the entire software development process; therefore, their usefulness in DP can potentially lead to improvements in resource allocation as well as reduced development or maintenance costs. The findings across all projects mention that overall metrics of software can be useful for DP. The Decision Tree (DT) and Random Forest (RF) mostly achieved good findings.

In 2023, Wenjun [20] suggested a new SDP method according to the program feature of semantic mining, which involves extracting semantics from the code and then mining the SD characters on the semantics. Lastly, SD was predicted by utilizing the mined defect characters. Such a method can effectively extract the semantic code data and extra mine the defect characters' impact on semantics. Nevertheless, such a method utilizes an abstract syntax tree (AST) to execute an intermediate code signification. The trial findings display that, in comparison with the present SDP methods, the PSFM method gained a greater value of F-measure.

This work presents the SDP tool as a step, in the realm of automated software defect prediction. It improves the precision and trustworthiness of defects and Defect density predictions by incorporating CK metrics and applying four transformations.

## 3. Metrics of CK

SDP metrics are the most significant function for building a model of prediction that tries to improve the quality of software by predicting as many SDs as possible. Most DP metrics can be separated into process metrics and metrics of code [21]. Metrics of code signify the way the source code is written, whereas metrics of process signify the complexity of the development process [21, 22]. Metrics of code are collected straight from the existing code of the source and mostly measure the code of the source properties, i.e., complexity and size. Its assumption of ground is that the code of source of greater complexity can be extra defect-prone [7]. The software as object-oriented is implemented effectively as a friendly user according to the customer's requirements; nonetheless, the object-oriented (OO) model is utilized to increase complexity and size, as well as object-oriented code, which helps the beginner and the experienced programmer [23].

Chidamber and Kemerer suggested the initial OO package of software standards, named Metrics of CK [24]. The metrics suits the authors' pretension that such measures can support utilizers in software understanding complexity, in noticing flaws in software, in forecasting definite project results, and in the software of external criteria, i.e., testing, SD, and effort of maintenance. Metrics of CK aid in investigating complexity, understandability/re-usability, usability/application-specific, and testability/maintainability. Therefore, it is vital to possess a program of metrics in the whole software stage of the Development Life Cycle (SDLC) to observe the input quality, output, and procedure [25]. Metrics of CK are the most widely used OO software metrics, and thus there is no necessity for comparing such metrics with others. The utilized metrics of CK are [23, 26]:

1. WMC: Weighted Method per Class offers the complication summation of each method in the class.
2. DIT: Depth of Inheritance Tree, the software as a maximum from the node of the root tree to the class node is regarded as the depth of the class and is measured through the predecessor class.
3. NOC: Number Of Children, the subclasses that are derived directly from the main class in the class hierarchy is known as children.
4. CBO: Coupling Between Objects, the total associated diverse classes.
5. RFC: Response For a Class refers to all layouts that are executed, obviously in message response, which is obtained via an object class.
6. LCOM: Lack of Cohesion of Methods, computes the methods' variances in a class through quality incidents or factors used by methods.

## 4. Methodology

The primary purpose of the ASDP (SDP) tool is to find out the prediction of the defect of software and important aspects related to the DP, utilizing the basic transformations according to the metrics of CK for the software, as illustrated in Fig. 1. The SDP tool is built using the Java programming language, and to achieve the above purposes, the SDP tool consists of three interfaces:

1. The main interface: contains the following elements:

- Choose SW: to select the software in which the defect will be predicted.
- CK Metrics Parser: calculates and saves the Metrics of CK for software in a file and calls the Metrics of CK interface to show them.
- Calculate the transformation for all values of the metrics of CK and save the result in a new file according to the following equations:

Log Transformation: $\log(\mathrm{CK})$ ………………….  ……1

Ln transformation: $\ln(\mathrm{CK})$ ……2

Pow transformation: $\mathrm{Pow}(\mathrm{CK})$ ……3

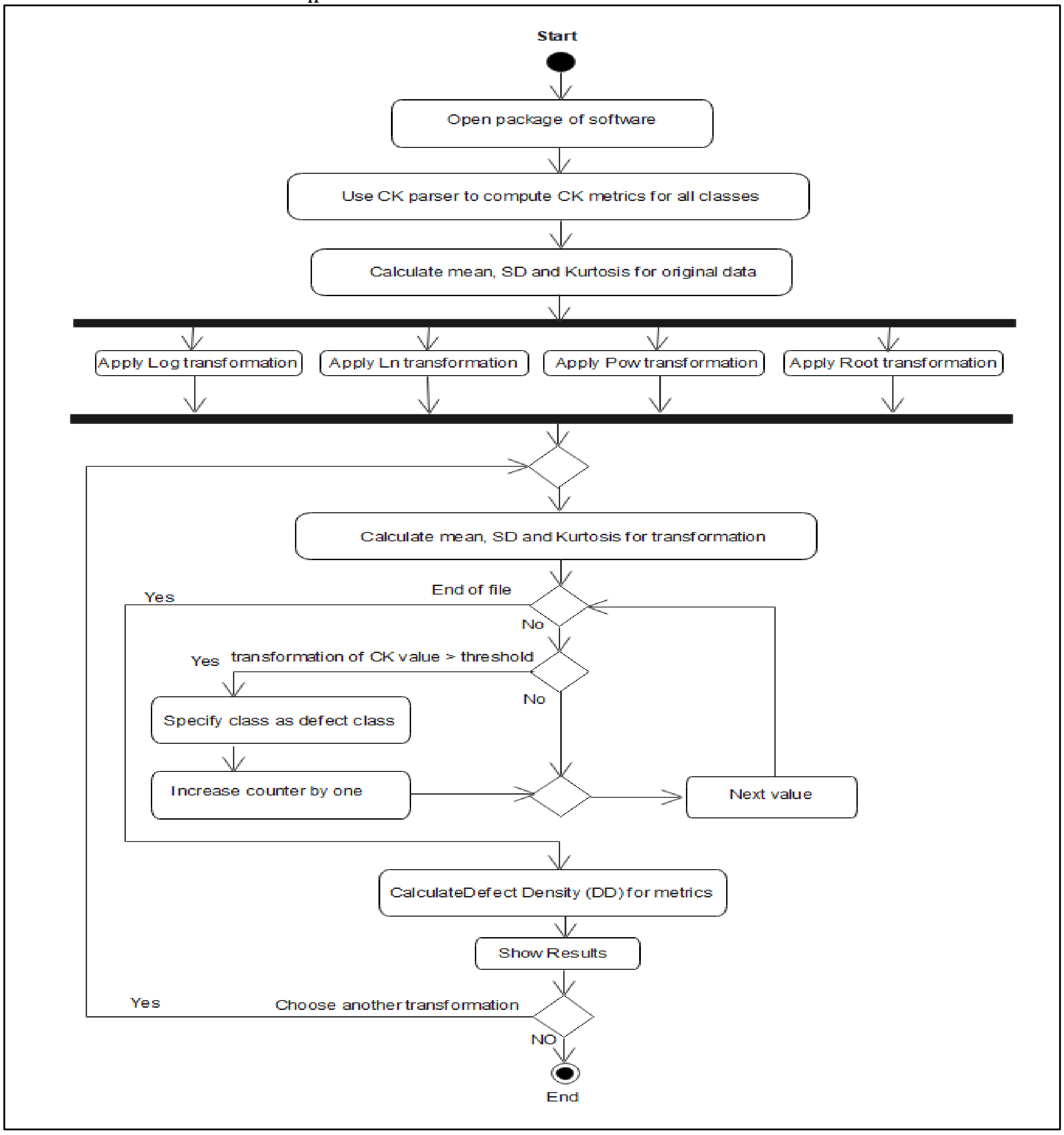
Root transformation: $\sqrt{(\mathrm{CK})}$ ……4

2. CK Metrics Parser interface: as mentioned in the main interface. View metrics are extracted from the software.
3. DP interface: this interface includes the following tables, each of which was calculated according to its equation for each metric:

- Mean (M): $\frac{\sum_{i=1}^{n} X_i}{n}$ ……5
- Standard Deviation (SD): $\frac{\sqrt{(\sum_{i=1}^{n} X_i - \text{mean})^2}}{(n-1)}$ ……6
- Kurtosis : = $\frac{\sum_{i=1}^{n}(X_i - \text{mean})^4}{(n*SD^4)}$ ……7
- Threshold(Th): $M + SD + C$ ……8

where C is a constant value

- Defect Class (DC) : $\sum_{i=1}^{n} dc_i$ ……9
- Defect Density (DD): $\frac{DC}{n}$ ……10



Start
Open package of software
Use CK parser to compute CK metrics for all classes
Calculate mean, SD and Kurtosis for original data
Apply Log transformation
Apply Ln transformation
Apply Pow transformation
Apply Root transformation
Calculate mean, SD and Kurtosis for transformation
Yes
End of file
No
Yes
transformation of CK value > threshold
No
Specify class as defect class
Increase counter by one
Next value
CalculateDefect Density (DD) for metrics
Show Results
Yes
Choose another transformation
NO
End

**Figure 1:** Activity diagram of the proposed tool (SDP)

## 5. Case Study

To demonstrate how the SDP tool works and is utilized, open-source software (Eclipse) is used. First, the software(SW) is selected from the available options in the interface labelled 'Choose SW', then the metrics are extracted from the CK Metrics parser. After that, the percentage of SD, their density, and other information related to the prediction process for the specific transformation are predicted, as shown in Figs. 2, 3, and 4.

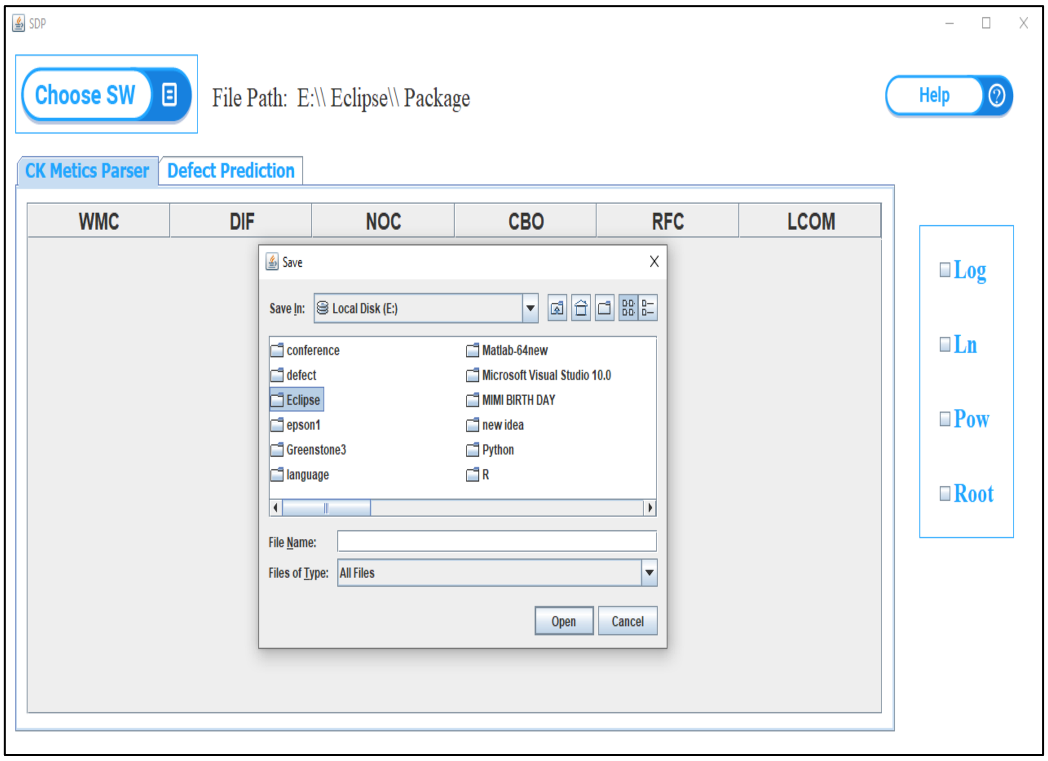
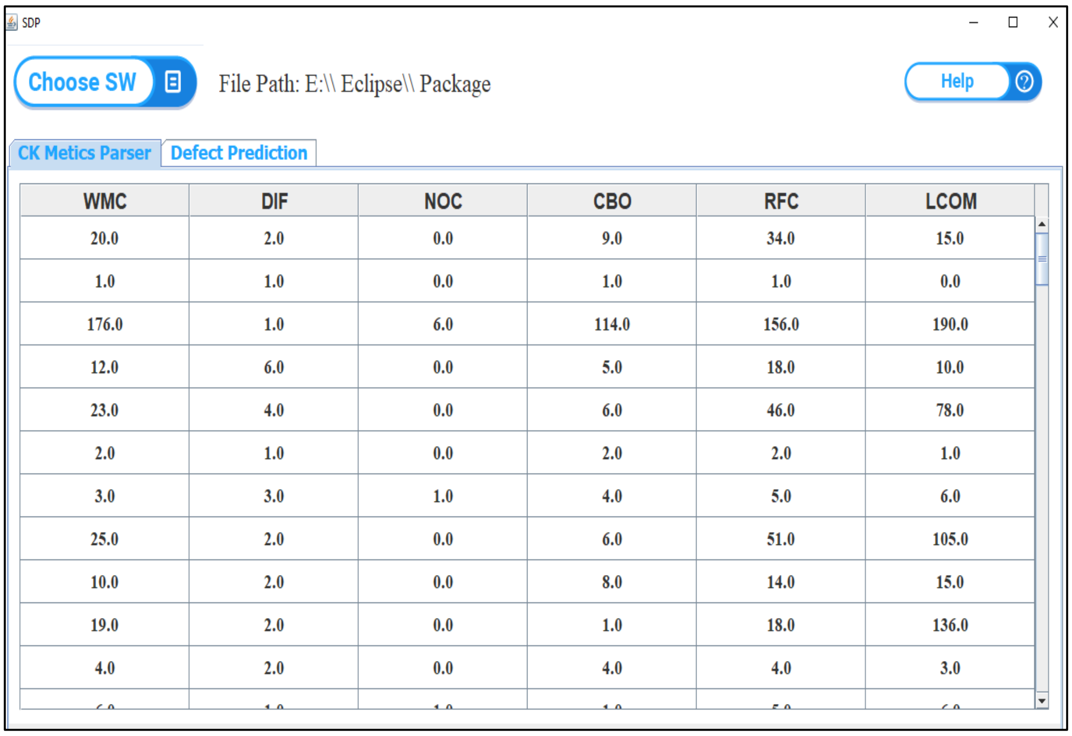
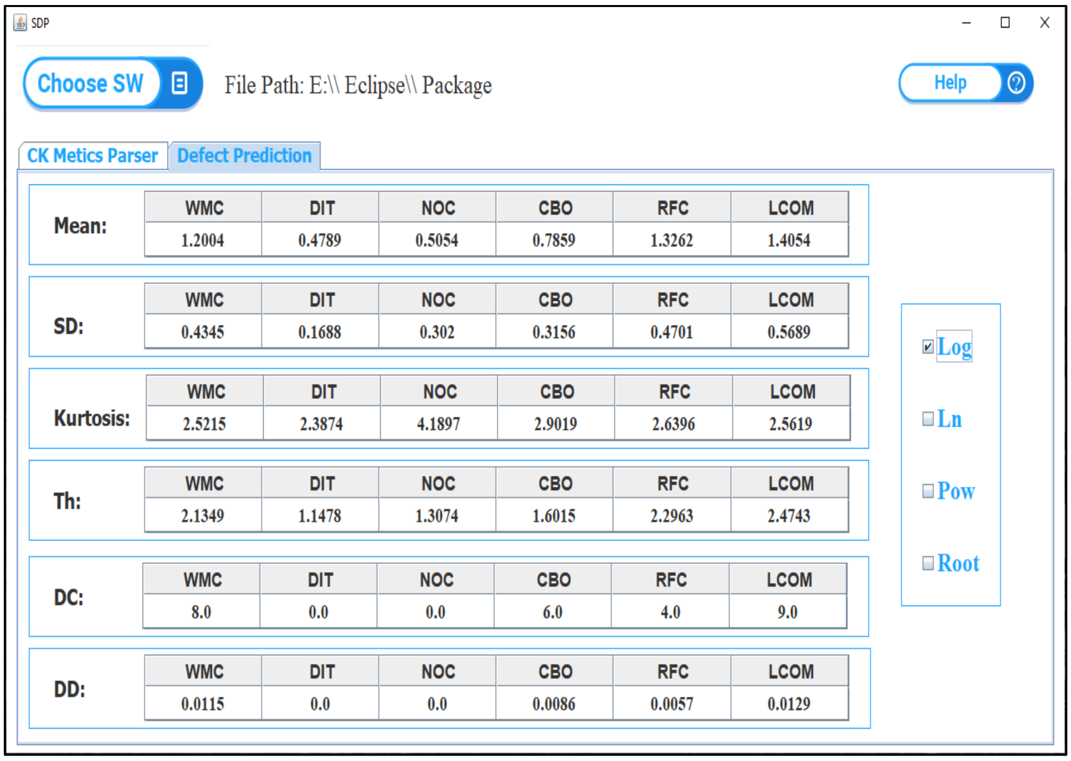


**Figure 2:** Choose software interface



| WMC | DIF | NOC | CBO | RFC | LCOM |
|---|---|---|---|---|---|
| 20.0 | 2.0 | 0.0 | 9.0 | 34.0 | 15.0 |
| 1.0 | 1.0 | 0.0 | 1.0 | 1.0 | 0.0 |
| 176.0 | 1.0 | 6.0 | 114.0 | 156.0 | 190.0 |
| 12.0 | 6.0 | 0.0 | 5.0 | 18.0 | 10.0 |
| 23.0 | 4.0 | 0.0 | 6.0 | 46.0 | 78.0 |
| 2.0 | 1.0 | 0.0 | 2.0 | 2.0 | 1.0 |
| 3.0 | 3.0 | 1.0 | 4.0 | 5.0 | 6.0 |
| 25.0 | 2.0 | 0.0 | 6.0 | 51.0 | 105.0 |
| 10.0 | 2.0 | 0.0 | 8.0 | 14.0 | 15.0 |
| 19.0 | 2.0 | 0.0 | 1.0 | 18.0 | 136.0 |
| 4.0 | 2.0 | 0.0 | 4.0 | 4.0 | 3.0 |

**Figure 3:** Extracted CK metrics for selected software



File Path: E:\\ Eclipse\\ Package

| | WMC | DIT | NOC | CBO | RFC | LCOM |
|---|---|---|---|---|---|---|
| Mean: | 1.2004 | 0.4789 | 0.5054 | 0.7859 | 1.3262 | 1.4054 |
| SD: | 0.4345 | 0.1688 | 0.302 | 0.3156 | 0.4701 | 0.5689 |
| Kurtosis: | 2.5215 | 2.3874 | 4.1897 | 2.9019 | 2.6396 | 2.5619 |
| Th: | 2.1349 | 1.1478 | 1.3074 | 1.6015 | 2.2963 | 2.4743 |
| DC: | 8.0 | 0.0 | 0.0 | 6.0 | 4.0 | 9.0 |
| DD: | 0.0115 | 0.0 | 0.0 | 0.0086 | 0.0057 | 0.0129 |

☑ Log
☐ Ln
☐ Pow
☐ Root

**Figure 4:** Software Defect Prediction for selected transformation

The statistical findings of transformations are described in Tables 1, 2, 3, 4 of the case study.

**Table 1**: Original extracted metrics, defect class, and defect density of the Eclipse software.

| Metric | Mean | SD | Kurtosis | Threshold | Defect Class | Defect Density |
|---|---|---|---|---|---|---|
| *WMC* | 23.6028 | 30.8253 | 79.8371 | 54.9282 | 66 | 0.094964 |
| *DIT* | 2.5942 | 1.5766 | 3.8975 | 4.6708 | 87 | 0.125179 |
| *NOC* | 0.4892 | 1.8873 | 88.3648 | 2.8765 | 23 | 0.033093 |
| *CBO* | 7.4187 | 8.3468 | 72.9481 | 16.2656 | 54 | 0.077697 |
| *RFC* | 34.3582 | 50.7647 | 210.6208 | 85.6230 | 45 | 0.064748 |
| *LCOM* | 56.6834 | 287.4699 | 604.7420 | 344.6534 | 4 | 0.005755 |

**Table 2:** Extracted metrics, defect class, and defect density of the Eclipse software after log transformations.

| Metric | Mean | SD | Kurtosis | Threshold | Defect Class | Defect Density |
|---|---|---|---|---|---|---|
| *WMC* | 1.2004 | 0.4345 | 2.5215 | 2.1349 | 8 | 0.0115 |
| *DIT* | 0.4789 | 0.1688 | 2.3874 | 1.1478 | 0 | 0 |
| *NOC* | 0.5054 | 0.3020 | 4.1897 | 1.3074 | 0 | 0 |
| *CBO* | 0.7859 | 0.3156 | 2.9019 | 1.6015 | 6 | 0.0086 |
| *RFC* | 1.3262 | 0.4701 | 2.6396 | 2.2963 | 4 | 0.0057 |
| *LCOM* | 1.4054 | 0.5689 | 2.5619 | 2.4743 | 9 | 0.0129 |

**Table 3:** Extracted metrics, defect class, and defect density of the Eclipse software after ln transformation.

| Metric | Mean | SD | Kurtosis | Threshold | Defect Class | Defect Density |
|---|---|---|---|---|---|---|
| *WMC* | 2.7609 | 0.9995 | 2.5215 | 4.2604 | 35 | 0.050359 |
| *DIT* | 1.1016 | 0.3882 | 2.3874 | 1.9899 | 4 | 0.005755 |
| *NOC* | 1.1626 | 0.6946 | 4.1897 | 2.3572 | 5 | 0.007194 |
| *CBO* | 1.8075 | 0.7259 | 2.9019 | 3.0335 | 35 | 0.050359 |
| *RFC* | 3.0503 | 1.0813 | 2.6396 | 4.6316 | 32 | 0.046043 |
| *LCOM* | 3.2324 | 1.3085 | 2.5619 | 5.0409 | 44 | 0.063309 |

**Table 4:** Extracted metrics, defect class, and defect density of the Eclipse software after power transformation.

| Metric | Mean | SD | Kurtosis | Threshold | Defect Class | Defect Density |
|---|---|---|---|---|---|---|
| *WMC* | 1536 | 9623 | 546.020 | 11163.46 | 11 | 0.015827 |
| *DIT* | 9.2158 | 11.4279 | 8.7925 | 24.6437 | 87 | 0.12517 |
| *NOC* | 19.1449 | 76.6494 | 45.8731 | 99.7943 | 5 | 0.007194 |
| *CBO* | 127.64 | 697.09 | 299.689 | 828.739 | 13 | 0.018705 |
| *RFC* | 3762.96 | 40172.95 | 670.64 | 43939.91 | 4 | 0.005755 |
| *LCOM* | 91234.15 | 2128724 | 652.5583 | 2219962 | 1 | 0.001438 |

**Table 5:** Extracted metrics, defect class, and defect density of the Eclipse software after root transformation.

| Metric | Mean | SD | Kurtosis | Threshold | Defect Class | Defect Density |
|---|---|---|---|---|---|---|
| *WMC* | 4.2903 | 2.3760 | 8.0233 | 10.6663 | 11 | 0.015827 |
| *DIT* | 1.5418 | 0.4658 | 2.6096 | 6.0076 | 0 | 0 |
| *NOC* | 1.3937 | 0.7219 | 12.1730 | 6.1157 | 0 | 0 |
| *CBO* | 2.5191 | 1.1167 | 10.4747 | 7.6359 | 2 | 0.002877 |
| *RFC* | 5.0969 | 2.9033 | 13.1539 | 12.0002 | 12 | 0.017266 |
| *LCOM* | 5.8281 | 5.1253 | 93.5605 | 14.9535 | 20 | 0.028776 |

The computed value of the threshold is mentioned in all tables. It was suggested that a constant number that can range between 0 and 1 be added to the threshold formula. The values for log and ln transformations were 0.5, and the values for root and power transformations varied by adding 4.

A larger value of the threshold indicates greater accuracy of results for defective classes. The chosen value, was appropriate for the chosen application, but it might have been increased or lowered. The actual value of different kinds of software is determined by the precision of their anticipated outcomes. During the trial, it was discovered how these values produced the best outcomes and were so dependent on them. The entire number of classes grows when the value of the constant is close to zero, whereas it decreases when it is closer to one. By raising the constant's value, it may be more explicit about exactly which classes contain the most defects. The number of defect classes and defect density are computed based on these values. The results of the original CK metrics extracted from Eclipse software showed that the Defect Density ranged from (0.005755 - 0.125179), after log (0 - 0.12949), after ln (0.005755 -

0.063309), after power (0.001438 - 0.12517), and after root transformation (0 - 0.028776). These findings directly influenced the number of defect classes found, which varied between (4-87) for the original metric, and (0-9) when applying log transformation, (4-44) for ln, (1-87) for power, and (0-20) for root transformation, as shown in figure 5.

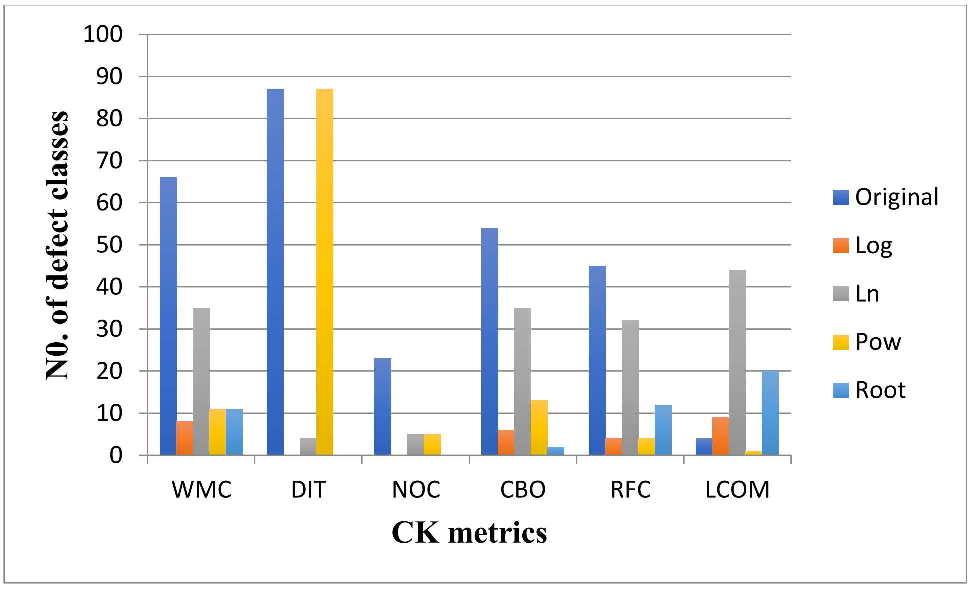


**Figure 5:** Comparisons of transformations based on the number of predicted defect classes

In general, the optimal transformations, ordered based on the results obtained, are: Log, then Root followed by Power, and finally ln. The best transformations for each metric are:

1. WMC: log, then power and root equally, and ln finally.
2. DIT: the log and root equally, then ln, and after that power, respectively.
3. NOC: the log and root equally, and after that, power and ln equally.
4. CBO: the root, then log, and after that power and ln, respectively.
5. RFC: the log and power equally, then root, and after that ln, respectively.
6. LCOM: the power, then log, and after that root and ln, respectively.

## 6. Conclusion

It is challenging to achieve the intended goal while maintaining high-quality software. Conventional testing methods are not adequate for quality assurance because of the few possibilities for customer testing when software evolves. ASDP approaches at diverse development stages are now pervasive in software monitoring quality. However, this needs measuring and tracing many metrics, which is costly and time-consuming. Therefore, a key issue is figuring out what important metrics to use to identify SD. The suggested tool (SDP) calculates the metrics of CK and four basic transformations to predict SD and DD to achieve more precise and reliable findings. It also renders refactoring and maintenance easy by concentrating on defect classes that require additional focus. Moreover, it enhances the source code's functionality and quality. Consequently, in upcoming studies, authors should expand on this work to other metrics and transformations, as well as the use of artificial intelligence algorithms.

## 7. Acknowledgments

The author appreciates the College of Computer Sciences and Mathematics at the University of Mosul for providing facilities that assisted in improving the current work quality.

## References

**[1]** I. Mistrik, R. Soley, N. Ali, J. Grundy, and B. Tekinerdogan, "*Software quality assurance: in large scale and complex software-intensive systems*", *USA: Morgan Kaufmann*, 2015, pp. 420.

**[2]** M. Nevendra, and P. Singh, "Cross-Project Defect Prediction with Metrics Selection and Balancing Approach", *Applied Computer Systems,* no. 2, pp. 137-148, 2022. doi: 10.2478/acss-2022-0015.

**[3]** Q . Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu, "A general software defect proneness prediction framework", *IEEE transactions on software engineering,* vol. 37, no. 3, pp. 356-370, 2011. doi: 10.1109/TSE.2010.90.

**[4]** H. Kim, "Using pre-release test failures to build early post-release defect prediction models. in *Proceedings of the IEEE 25th International Symposium on Software Reliability Engineering*, Naples, 2014. doi:10.1109/ISSRE.2014.21

**[5]** F. Li, P. Yang, J.w.Keung, W. Hu,H. Luo, and X. Yu, "Revisiting revisiting supervised methods for effort-aware cross-project defect prediction", *IET Software,* vol. 17, no. 4, pp. 472-495, 2023. doi:10.1049/sfw2.12133.

**[6]** K.Nehéz, and N. Khleel, "A new approach to software defect prediction based on convolutional neural network and bidirectional long short-term memory", *Production Systems and Information Engineering,* vol. 10, no. 3, pp. 1-15, 2022. doi:10.32968/psaie.2022.3.1.49.

**[7]** Z. Li, X. Y. Jing, and X. Zhu, "Progress on approaches to software defect prediction", *IET Software,* vol. 12, no. 3, p. 161–175, 2018. doi:10.1049/iet-sen.2017.0148

**[8]** p. Bourque and R.Fairley , " Guide to the Software Engineering Body of Knowledge", *SWEBOK V3.0. IEEE Computer Society,* 2014.

**[9]** C. López-Martín,Y. Villuendas-Rey ,M. Azzeh , A.B. Nassif, and S. Banitaan , "Transformed k-nearest neighborhood output distance minimization for predicting the defect density of software projects", *Journal of Systems and Software,* vol. 167, p. 110592, 2020. doi:10.1016/J.JSS.2020.110592

**[10]** J. Nam and S. Kim, "CLAMI: Defect Prediction on Unlabeled Datasets (T)", in *Proceedings of the 30th IEEE/ International Conference on Automated Software Engineering (ASE)*, Lincoln, 2015. doi: 10.1109/ASE.2015.56.

**[11]** P. He, B. Li, X.Liu, J. Chen, and Y.Ma, "An empirical study on software defect prediction with a simplified metric set", *Information and Software Technology,* vol. 59, pp. 170-190, 2015. doi:10.1016/j.infsof.2014.11.006.

**[12]** C. Tantithamthavorn , S. McIntosh, A E. Hassan, and K. Matsumoto, "Automated Parameter optimization of classification techniques for defect Prediction models", in *Proceedings of the 38th International Conference on Software Engineering (ICSE ' 16)Association for Computing Machinery*, New York, 2016. doi:10.1145/2884857.

**[13]** S. Huda ,S. Alyahya ,M. Ali, S. Ahmed, J. Abawajy, H. Al-Dossar and J. Yearwood, "A Framework for Software Defect Prediction and Metric Selection", *IEEE Access,* vol. 6, pp. 2844-2858, 2017. doi: 10.1109/ACCESS.2017.2785445

**[14]** S. Ghosh ,A. Rana, and V. Kansal, "A Nonlinear Manifold Detection based Model for Software Defect Prediction", *Procedia Computer Science,* vol. 132, pp. 581-594, 2018. doi: 10.1016/j.procs.2018.05.012.

**[15]** C. Manjula,and L. Florence, " Deep neural network based hybrid approach for software defect prediction using software metrics", *Cluster Computing,* vol. 22, no. 4, pp. 9847-9863, 2019. doi:10.1007/s10586-018-1696-z

**[16]** M. NezhadShokouhi , M. Majidi,and A. Rasoolzadegan, "Software defect prediction using over-sampling and feature extraction based on Mahalanobis distance", *The Journal of Supercomputing,* vol. 76, pp. 602-635, 2020. doi:10.1007/s11227-019-03051-w

**[17]** J. Ren, and F. Liu, "A novel approach for software defect prediction based on the power law function", *Applied Sciences,* vol. 10, no. 5, p. 1892, 2020. doi:10.3390/app10051892.

**[18]** S. Alhusain, “Predicting Relative Thresholds for Object Oriented Metrics”, in *Proceedings of the IEEE/ACM International Conference on Technical Debt (TechDebt)*, Madrid, 2021. doi:10.1109/TechDebt52882.2021.00015.

**[19]** D. REBRO, “ Software Metrics for Software Defects Prediction”, *Master’s Thesis,Masarykiana university, Czech.,* 2022.

**[20]** W. Yao, M. Shafiq,X. Lin, and X. Yu, “A Software Defect Prediction Method Based on Program Semantic Feature Mining”, *Journals Electronics,* vol. 12, no. 7, p. 1546, 2023. doi:10.3390/electronics12071546.

**[21]** F. Rahman,and P. Devanbu, “How, and why, process metrics are better”, in *Proceedings of the 35th International Conference on Software Engineering (ICSE)*, San Francisco, 2013.

**[22]** L. Madeyski,and M. Jureczko, “Which process metrics can significantly improve defect prediction models? An empirical study”, *Software Quality Journal,* vol. 23, pp. 393-422, 2015. doi: 10.1007/s11219-014-9241-7.

**[23]** U. Pooja,and N. PK, “Prediction of Software Defects Using Object-Oriented Metrics”, *International Journal of Civil Engineering and Technology (IJCIET)*, vol. 9, no. 1, p. 889–899, 2018. Available: https://iaeme.com/Home/issue/IJCIET?Volume=9&Issue=1

**[24]** C. Chou, “Metrics in Evaluating Software Defects”, *International Journal of Computer Application( IJCA),* vol. 63, pp. 23-29, 2013. doi: 10.5120/10447-5147

**[25]** M. Qureshi,W. Qureshi, “Evaluation of the Design Metric to Reduce the Number of Defects in Software Development”, *International Journal of Information Technology and Computer Science,* vol. 4, no. 4, pp. 9-17, 2012. doi: 4. 10.5815/ijitcs.2012.04.02.

[26] A. M. Altaie, A. Y. Hamo and R. Gh. Alsarraj, “Software Fault Estimation Tool Based on Object-Oriented Metrics”, *Iraqi Journal of Science,* no. Special Issue2, pp. 63-69, 2021, doi:10.24996/ijs.2021.SI.2.7