Chapter 1

Introduction

"I discovered that a fresh start is a process. A fresh start is a journey – a journey that requires a plan." – Vivian Jokotade

Software testing requires a huge amount of testing effort, and we have to ensure the testing irrespective of bugs, i.e., buggy module, bug-free module. Testing effort can be minimized if we know or identify the modules that are likely to be bug-free and those that may contain a bug. This will certainly help us to obtain high-reliability of software system [169] that is known to be the probability of failure-free software operation for a specified period of time in a specified environment. Software testing identifies the software product that matches the expected requirements and ensures the software product is fault-free and reliable. Software defect prediction identifies potential faulty modules before the testing phase to reduce testing effort. This chapter provides the basic outlines for the thesis that enumerate the primary concepts and glossary for the entire document. Some general ideas and definitions of software defect predictions are present in section 1.1, existing methods in section 1.1, and limitations of exiting defect prediction works in section 1.2. Further, we will express the work's motivation in section 1.3, and research goals in section 1.4. Finally, we will elaborate the contributions made in the thesis and literature review in section 1.5 and section 1.6, respectively.

1.1 Software Defect Prediction

Software fault is a physical anomaly that can cause a software to fail to its required function, an encounter fault known as a defect, and the tester called it a bug. We will use fault, defect, or bug as interchangeable terms throughout the thesis. According to ISO/IEC/IEEE 24765 definition, a software defect is *"a manifestation of an error in software"*. Software defect prediction is a process that can conveniently determine the software module or class that is more likely to be fault-prone [266, 162]. The continuous changes, deadline pressure, and the requirement to ensure the flawless behavior of the functionalities force the developers during their activity [57]. However, limited manpower and time bound cause serious threats to test software systems. Thus, the available resources should be efficiently allocated such that the source code that might consist of the more defective module requires more testing resources. For example, the cost of a Java-based project [233] of 100 function points and 10 KLOC. The effort required for the five different phases: 10 percent needed for requirement engineering, analysis, and design requires 20 percent, coding 30 percent, testing 35 percent; finally, installation and training needs 5 percent, as indicated in Table 1.1. The accurate prediction focuses on defect-prone modules, which must be tested more

TABLE 1.1: Method level metrics description.

Software Development Phase	**Effort**
Requirement	2
Analysis and design	4
Coding	6
Testing	7
Installation	1
Total	20

extensively; it may help in effectively prioritizing and allocating the limited testing resources to reduce the overall testing effort. SDP is usually a supervised method in which a group of independent variables (predictors) is employed to predict the dependent variable (defect-prone module). A model is train using various machine learning [155], deep learning [249] or statistical learning techniques [116]. Fig. 1.1 refers to the basic architecture of the SDP technique. Classes/instances/modules are extracted from a software project. Relevant software metrics values are selected and fed into the learning model to train. After training, the trained model predicts the faulty or non-faulty class of different projects (new or target projects). Training of a prediction model may be any one of the two possible types. First, defect prediction

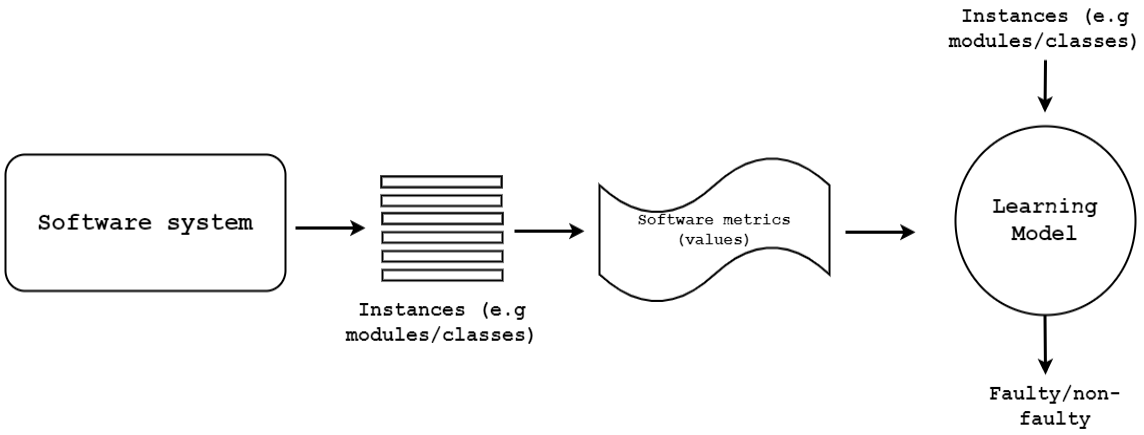

FIGURE 1.1: Basic architecture of software defect prediction technique.

techniques are investigated in the context of a within-project that assumes previous defect data for a given project, such prediction approaches called the within-project SDP model. The second approach is to assemble training data composed of known external software projects called cross-project (similar) strategy.
The three primary factors that affect the performance of SDP models are:

1. An appropriate selection of learning techniques positively influences the performance of SDP models. Ghotra et al. [76] concluded that the performance of SDP methods could be varied up to 30% by selection deferent classification techniques. Panichella et al. [186] demonstrated that the prediction performance of different classifiers is better despite the similar prediction approach.

2. Choosing an appropriate set of software metrics also impacts the performance of predictive models. The feature selection is a procedure to select software features for the predictive mechanism; unsuitable selection leads to information loss and will produce poor performance.

3. The quality of the datasets is an essential aspect of the performance of SDP approaches. Sheppard et al. [220] raise over the quality of software defect datasets, they found most of the PROMISE [215], and NASA [220] are noisy and have duplicate instances; which leads to provides unsatisfactory results.

Software defect prediction can be categories into four different classes, as shown below.

(i) *Classification:* It includes the study regarding classification of software structure (functions, files, modules, etc.) into non- defective/defective proneness, and different levels of fault extremity by applying statistical techniques (logistic regression [112], discriminant analysis [152]) and machine learning methods such as an artificial neural network [113], support vector machine [209].

(ii) *Regression:* This type is predicting the number of software faults in the software system by applying various methods (support vector regression [228], genetic algorithm [200]).

(iii) *Mining association rule:* It mainly shows the relationship between software metrics, software modules, and faultiness by utilizing methods such as relational association rule [46] or CBA2 algorithm [148].

(iv) *Ranking:* The objective is to study the rank of software structure concerning the number of faults in the module. It prioritizes the defect-prone modules according to fault density; the denser module requires more testing effort.

However, in this thesis, we work over the classification and regression-based approaches. Now, we will illustrate the various categories of regression-based approaches that we have considered in our thesis work, i.e., cross-version defect number estimation (regression), cross-project fault number prediction (regression), and hybrid regression analysis.

1.1.1 Cross Version Defect Number Prediction

Software Defect Number Prediction (SDNP) [202, 201] is one step ahead of software defect prediction; it not only identifies the defective or non-defective module but also indicates the number of faults in each module. SDNP is an active research area due to the development of large and complex software systems. There are some recent articles [23, 37] are published on SDNP. Researchers and software practitioners have proposed a few approaches and algorithms using various statistical methods and machine learning techniques. Chen et al. [37] compared the performance of the supervised and unsupervised method of defect count prediction. Similarly, Rathore et al. [202] applied liner and non-linear heterogeneous ensemble learning techniques to predict a number of faults in the software system. The number of defects may be possible using cross-version also, know as Cross Version Defect Number Prediction

(CVDNP). It is a process to estimate the number of bugs present in every module of the next version of a software system. It is also known as cross-version defect count prediction. The model is trained by all the existing versions of the same software system and estimates the number of defects in every module of the next version.

- Let the software system S has n versions, where n is a positive integer.
- Let f is the number of features in each version, f is fixed for all versions of the same software system.
- Every version has a different number of modules/classes/instances, and defective modules consist of a finite number of faults.
- The proposed CVDNP technique in this thesis can estimate the bug count vector of the next $((n+1)^{th})$ version of a software system.

In this thesis, we have emphasized cross-version defect number estimation instead of software defect number prediction.

1.1.2 Cross Project Defect Prediction

SDP uses various features of projects such as process metrics, previous-values of defect, source code metrics, etc., to characterize the prediction method and predict a module/class defective or non-defective. Such prediction model is based on training data and some learning algorithms, whereas the performance evaluation is carried out using testing data. When the training and testing data belong to the same project, it is called within project software defect prediction (WPSDP). WPSDP requires extensive historical training data from software projects. However, in practice, it is infrequent to get adequate training data for a new project. Still, there are datasets from other projects such as the PROMISE repository [215], which comprises many diverse software projects released for defect prediction. Cross-project defect prediction (CPDP) employing data from one repository (aka. source project) to train a learning model and predict the defective or non-defective module of different projects (aka. target project); it dispenses a new viewpoint of SDP technique [142, 78, 172]. The main challenge lies in constructing a model that can capture better generalizable properties over target projects and disregards non-generalizable properties that do not hold.

Cross-Project Defect Number Prediction (CPDNP) is one step ahead of CPDP. CPDNP is a way to estimate the number of defects present in each module of the software system. It is mainly integration of CPDP and SDNP [202, 203] mechanisms. SDNP is the method to predict the defect count of each module of a test project. As most of the defect datasets are skewed distributed toward negative class, so it suffers from class imbalance problem [100], which leads to unsatisfactory results. Overfitting is also a prominent challenge in such predictive models. In this chapter, we deal with the CPDNP problem and propose a new predictive model to conquer the integrated challenges of CPDP and SDNP. The chapter mainly emphasis over the CPDNP technique.

1.1.3 Hybrid Regression Analysis

Regression problem formulation is a way to estimate all the software features and the number of bugs in every module of a software system. Such a prediction mechanism can be modeled by employing all the existing versions of the same software. To date, no previous work on software bug prediction has focused on the prediction of the successive version of the same software system. Let there be n versions in a software system, and the objective is to predict the data for the $(n+1)^{th}$ version. Let there be n versions in some software system, say S, where n is a positive integer. S = [S_1, S_2,....,S_n]. Let the dataset corresponding to the i^{th} version be d_i, where each d_i has a m-dimensional feature vector, say f. So d_i = [f_1, f_2,...., f_m, b_i], here b_i is the Bug Count Vector (BCV) of software S_i. The predictive data contains all metrics values and BCV.

1.2 Limitations of Existing Software Defect Prediction Approaches

Every SDP domain does have many challenges; some of them are common across domains. Such as obstacles of high dimensionality, overfitted results, and Class Imbalance (CI) problem. Data class imbalance means the cardinality of defective modules (minority class) has smaller than non-defective modules (majority class). Boehm et al. [21] concluded that in most cases, 20% of software modules could end up having 80% of total software defects. Suppose the cardinality of defective modules

lies in a minority class compared with the non-defective modules. In that case, the SBP may not perform well and can lead to biased results towards the majority class. The enormous imbalanced datasets are popular to minimize the efficiency of machine learning techniques for the prediction of exceptional class [154]. The variation among the distribution of data points leads to the poor performance of some ML techniques, mainly for the minority class. Two well-known techniques address the class imbalance problem [12, 266], first at the algorithmic level and second at the data level (sampling methods).

1.2.1 Limitation on Cross Version Defect Number Prediction

The existing CVDNP methods consist of a few major challenges are shown below.

(i) Predicting the BCV in the upcoming version of a software system may alert the developer to examine the probable fault-prone modules, and predicting the number of bugs before the testing phase minimizes the enormous testing effort for different modules. Minimal works have been done in this domain.

(ii) Information regarding all prior versions of the software project will be helpful to build a better BCV estimator for the upcoming version of the same software. Considering the BCV prediction as a regression problem is still a limited research direction.

(iii) Our empirical study reported that classical ML-based bug count prediction models are inefficient over large software systems with high computational costs.

(iv) Deep learning architecture can solve many real-life problems, recently few deep learning-based SBP models surpassed the performance over classical ML-based models. So, it can also be helpful in BCV prediction.

1.2.2 Cross Project Defect Prediction

CPDP is still challenging and leads to poor performance because the model is trained to utilize a set of projects that might not generalize well over the new project [175]. The main challenge is constructing a model that can capture better generalizable

properties over target projects and disregards non-generalizable properties that do not hold for the target project. In ML literature, to bridle the data distribution between various domains [47, 58, 175] transfer learning has been applied. It extracts mutual information from one field and transfers it to another area. Cross-project Defect Number Prediction (CPDNP) is one step ahead of CPDP. CPDNP is a methodology to estimate the number of defects present in each module of the software system. It is mainly the integration of CPDP and software defect number prediction (SDNP) [202, 203] mechanisms. SDNP is the method to predict the defect count of each module of a test project. As most of the defect datasets are skewed distributed toward negative class, so it suffers from class imbalance problem [100], which leads to unsatisfactory results. Overfitting is also a prominent challenge in such predictive models.

(i) CPDP is unable to capture better generalizable characteristics over target projects that produce poor performance.

(ii) Most of the released defect datasets are imbalance causes CI problem, whereas overfitting is also a substantial challenge in most of the predictive methods. Both of these challenges lead to produce disappointing outcomes.

(iii) Rudimentary research has been done over SDNP as a regression problem. Only preparatory work has been conducted that integrates the SDNP and CPDP methods into one approach to the best of our knowledge.

(iv) According to our empirical study, the classical ML methods are not so useful in predicting defect numbers for large projects (a vast number of modules) compare to deep learning models. A deep neural network can be more efficient and robust over such approaches for substantial projects.

1.2.3 Limitations on Hybrid Regression Analysis

Prediction of metric values and the number of bugs in software modules help to develop high-quality software products at low cost, further reducing the testing and development efforts of SDLC. The limitation of existing work are:

(i) Still, no previous work on software bug prediction has focused on predicting the entire software (number of bugs along with software metrics in each module) of the successive version of the same software system.

(ii) The success of deep learning architectures in time series prediction and sequence prediction motivates their application for the next version prediction of software systems.

(iii) Class imbalance and overfitting is also a significant challenge in the hybrid regression domain.

1.3 Motivation

We categorize the motivation section for all different SDP domains; the main objective of any categories of the SDP approach is to identify the faulty modules and reduces the testing effort, and the developer team leader can properly allocate various testing resources. So there are few common motivational points for all SDP domains.

Classification Based SDP

Software practitioners revealed that deep representation is better than traditional feature extraction techniques [92, 128] in various applications. Deep Belief Network (DBN) and Stacked Denoising Autoencoder (SDA) have successfully implemented SBP, whereas SDA outperforms for noisy datasets [254]. Few software practitioners [80, 194] claim the quality of datasets and their noisy instances, which should be considered during the SBP model. Some other researchers revealed that ensemble learning could lead to the risk of over-fitting problem [253] and the biased result (class imbalance issue) [240] but surpass the performance over weak classifiers. It is necessary to extract more relevant software features that cause a noisy removal extraction process for better results in the SBP model.

It has been found that deep representation dominates standard features to reveal the core of the research object [92, 128]. The DBNs have been successfully implemented over SBP by many researchers [285, 262]. SDA has first time applied for SBP by [249], SDA outperforms the deep belief network mainly for noisy data it is concluded by [254]. Wang et al. [266] found that ensemble learning can solve the class imbalance problem in the domain of SBP. The researcher tries to propose more robust EL methods that can beat the performance of existing EL-based SBP models. Sampling techniques are the redistribution of data points in the dataset, which helps to avoid skewed distribution of positive or negative instances. There is no class imbalance problem when the data distribution of positive and negative

instances is approximately equal.

Motivation for Cross Version Defect Number Prediction

Research on software bug count prediction [37, 202] is recently growing faster due to the development of large and complex software systems. There are few recent articles [23, 37] is published on a similar issue. Researchers and software practitioners have proposed numerous models and algorithms using various statistical methods and machine learning techniques. Chen et al. [37] compared the performance of the supervised and unsupervised method of defect count prediction. Similarly, Rathore et al. [202] applied liner and non-linear heterogeneous ensemble learning techniques to predict a number of faults in the software system. They only considered module-level bug count prediction. None of them considered the bug count vector prediction over the complete software system for the upcoming version. The software project's dataset consists of many instances; each instance represents a specific module of a software project. Every column vector of these datasets represents various software metrics such as line of code (LOC), Number of children (NOC), etc. The last column of the dataset is a column vector; it is also known as a bug count vector. The bug count vector gives information about bug value in each software module, and it is represented as y_i. Let x_{ij} represent an item in the dataset, where i indicates the module, and j represents the software metric. y_i is the value of bug for i^{th} instance or module. For example, the project ant-1.3 of PROMISE repository [215] has 125 modules; the last column is a bug information vector, also called as bug count vector, it gives bug information of each of the 125 modules.

Deep learning architectures are capable of solving most of the real-life problems. Handwriting recognition [276], Fingerprint recognition [161] many more. Recently deep learning architecture was applied over feature extraction [249] in SBP that outperforms over baselines methods. Deep learning methods are so powerful for future sequence prediction after training using current and past sequences. It can also generate a predicted sequence. Our empirical study concludes that deep learning can also apply to bug count vector prediction and can be effective over the baseline technique. The two main reasons for the concatenation of different versions of the same software are, first, the software features are the same as all previous versions in our experimental data. Second, it also increases the data size for model training. The concatenation of all versions of the same software system makes the training more robust and effective.

Motivation for Cross-Project Defect Prediction:

In the new age of software systems, the software is becoming bulkier, complex, and sophisticated. Predicting only defective or clean module do not succor efficiently over minimizing testing effort. Predicting the number of defects in each module provides more information and minimizes the required testing effort. If a developer team leader has information about the number of defects in every module or the total number of defects in the project, he will optimally utilize testing resources; a more defective module requires more testing effort. Reducing the testing effort will lead to lowering the software development cost. Rathore et al. [202] suggested a linear/non-linear heterogeneous ensemble learning-based model predicts the number of defects in software systems.

Deep learning learns to generate features and trains the existing classical feature for prediction mechanism. The deep neural network is more robust and applicable for large datasets [211]. As we are using datasets of diverse projects, and amalgamation of those projects leads to massive datasets. Deep neural network and attention mechanisms have successfully applied and produced tremendous outcomes in several domains compared with classical machine learning methods. Chen et al. [34] proposed a model named DeepCPDP; they employed bi-LSTM along with attention layer over cross-project defect prediction. They conducted experiments over ten PROMISE repository projects, and they found DeepCPDP outperforms over eight baseline methods. Fan et al. [61] applied an attention mechanism to generate significant features for accurate defect prediction; they validated their approach by utilizing seven java projects. They found f-score and ROC are 14% and 8% higher than the existing methods. The attention mechanism is also successfully applied in solar-cell manufacturing defect prediction [239]; they utilized solar cell electroluminescence images data to designed subnetwork by connecting the novel channel-wise attention with spatial attention subnetwork. Their experimental results show that the proposed method surpasses the performance of other methods in terms of defect classification and detection results. In practical situations, the state-of-the-art prediction methods give unsatisfactory results due to a lack of training modules over new projects. According to some recent research, the deep learning models have more efficiency in predicting defective modules [249]. It is widely implemented in medical image identification [88], language translation [227], speech recognition [50], and many more.

A single prediction model is constructed from one source project after employing baseline approaches; when it optimally performs on diverse target projects, there

will be no requirement of the compositional model. To validate the compositional model's requirements, we analyzed different baseline learning models (e.g., TCA+) and investigated their performance over the cross-project defect prediction mechanism. TCA+ [175] is a two-phase transfer learning approach and efficient while predicting the target project's defects. Integrating software defect number prediction and cross-project defect prediction into one mechanism leads to more efficient methods for predicting the number of defects in every module/class of a new software project called cross-project defect number prediction. To the best of our knowledge, no such work has been done over CPDNP, and we are the first to deal with it. Deep learning (DL) and transfer learning (TL) methods have already been successfully applied independently over SDNP and CPDP. In this chapter, we are employing both DL and TL for better prediction mechanisms.

Hybrid Regression Analysis:

Deep learning [134] is progressing very rapidly in various domains. Long Short Term Memory (LSTM) [94] has found enormous applications, specially for sequence learning [245], to predict the next sequence in sequence datasets. LSTM is widely used in language modeling [242] for text classification [296], vascular disease detection [83], and many more different applications. The correct working of most machine learning techniques is based on certain assumptions: the test and train sets must be drawn from the same distribution and same feature space.

Let there be n versions in a software system. Our objective is to predict the data for the $(n+1)^{th}$ version. We employ a strategy where we train an LSTM model using the first n-1 versions as the train data to predict the n^{th} version. Then, this trained model is used to predict the $(n+1)^{th}$ version given versions 2 to n. The instance with class labels is created from the software repository. The data is then sent for preprocessing to remove noise, data duplication, etc. After preprocessing, the metadata (bigger data consisting of information of existing versions) is created using all existing versions of software and then fed to the trained model to predict the next version of the same software.

1.4 Research Goal

The research's main objective is to adopt various techniques to develop a software defect prediction approach with improved performance. During the thesis work, we

mainly focus on classification, regression, and hybrid analysis-based SDP estimation approaches. However, as explained, there are several issues with the existing approaches. We, therefore, attempted to and justified answers to the following research questions (RQs) as part of our study.

RQ-1: How the proposed approaches subjugate the class imbalance and overfitting problems in all these domains?

RQ-2: How much the proposed is cost-effective over existing methods in all these defect prediction field?

RQ-3: How much the performance of proposed models are significant?

RQ-4: Training time comparison of suggested approaches and baseline methods.

RQ-5: How much the various proposed techniques are stable?

We have added similar research queries for every chapter in order to justify these RQ in every chapter. We answer these research queries by addressing each chapter's research query for better clarification. A detailed discussion about these points is given in every individual chapter.

1.5 Contributions

This thesis is committed to the development of efficient software defect or defect count predictor in various SDP classes. To perceive answers to the research questions (RQs) as stated in section 1.4, we attempt to investigate the problem of software fault prediction in several aspects. First, when the deep features of software metrics are extracted from deep learning architecture are more informant towards predictive modeling. Second, data amalgamation of existing projects can be employed to create a bigger training set. Third, dealing with challenges of class imbalance, overfitting, and stability of the model. We are going to deal with a few other aspects, such as cost-effectiveness, prediction performances, training time, etc.

Thus, the main contributions of this thesis are related to the design, implementation, and evaluation of the performance of software fault prediction methods, where we have addressed several existing issues using novel methodologies. We have proposed novel approaches for each classification-based SDP, cross-version defect number prediction, and cross-project fault prediction; We have also proposed a novel approach in hybrid regression analysis for next version software prediction.

1.5.1 BPDET: Classification Based Software Defect Prediction Method

There are many hindrances in software fault prediction systems for detecting faulty modules, such as missing values or samples, data redundancy, irrelevance features, and correlation. Till now, very few works have been done which address the class imbalance problem in SBP. The main objective of this chapter is to reveal the favorable result by feature selection and machine learning methods to detect defective and non-defective software modules. We propose a rudimentary classification-based framework, Bug Prediction using Deep representation and Ensemble learning (BPDET) techniques for SBP; it combinedly applies ensemble learning (EL) and deep representation (DR). The software metrics which are used for SBP are mostly conventional. Staked denoising auto-encoder (SDA) is used for the deep representation of software metrics, which is a robust feature learning method. Propose model is mainly divided into two stages: the deep learning stage and two layers of the EL stage (TEL). The extraction of the feature from SDA is the very first step of the model then applied TEL in the second stage. TEL is also dealing with the class imbalance problem. The experiments were performed on NASA (12) datasets, and we revealed the efficiency of DR, SDA, and TEL. The performance is analyzed in terms of Mathew co-relation coefficient (MCC), the area under the curve (AUC), precision-recall curve (PRC), F-measure, and Time. Out of 12 dataset MCC values over 11 datasets, ROC values over 6 datasets, PRC values overall 12 datasets, and F-measure over 8 datasets surpass the existing state-of-the-art bug prediction methods. We have tested BPDET using Wilcoxon rank-sum test, which rejects the null hypothesis at 0.025. We have also tested the stability of the model over 5, 8, 10, 12, and 15 fold cross-validation and got similar results. Finally, we conclude that BPDET is stable and outperformed on most of the datasets compared with EL and other state-of-the-art techniques. Key features of BPDET over existing methods.

1. BPDET effectively allocated the testing resources to the more fault-prone module, which reduces the testing effort of the software development life cycle.
2. BPDET is able to handle extensive software metrics, i.e., more complex software systems, the software which has more number of software features have a high probability of software fault-prone modules, and cumbersome to handle such module, in such cases, the BPDET will be very efficient due to deep learning architectural layer, i.e., Staked denoising auto-encoder (SAD) layer.

3. When the dataset has a skewed distribution over positive and negative class, the imbalance issue arises, BPDET provides unbiased results in such situations. As we have compared the Mathews correlation coefficient (MCC) and PRC with other existing SBP models, we have applied the SMOTE [33] sampling methods with those classical SBP methods so that models can be compared appropriately. We have found that BPDET is more efficient for such software systems compared with other SBP methods.

4. BPDET also addresses the over-fitting problem, as we have applied two regularization methods, i.e., Early stopping [271] and Dropout [236] method in the SDA phase. We have also applied min-max normalization [274] into datasets. Even two-layer of EL also handle the over-fitting problem.

5. The overall performance of BPDET (ROC & F-measure) is much better than the state-of-the-art and other traditional SBP techniques. We have compared the performance with 10 different SBP models. We found that BPDET outperforms those SBP models in ten out of twelve public datasets, which makes it more useful.

6. BPDET is highly stable, as we have applied 10-fold cross-validation [127] method in our model, we tested the stability of the model at different levels (5-fold, 8-fold, 12-fold, and 15-fold) we found the values of performance metrics are approx equal.

7. We have applied non-parametric test, i.e., Wilcoxon rank sum test [193] over the performance of BPDET concerning other ten SBP model, H_0 = Median (Difference) = 0, H_a : Median (Difference) > 0, at significance level 0.025, we concluded that the null hypothesis is rejected.

1.5.2 BCV-Predictor: Cross-Version Defect Count Vector Predictor

The evolution of hardware, platform, and user requirements leads to develop the next version of a software system. In this chapter, we formulate a problem and its novel solution, i.e., predicting the bug count vector of a successive version of a software system. After predicting the bug count vector in the next version of the software, the developer team leader can adequately allocate the developers in respective fault-dense modules. In a more faulty dense module, more developers are required. We

have conducted our experiment over seven PROMISE repository datasets of different versions. We build metadata using a concatenation of different versions of the same software system for conducting experiments. We proposed a novel architecture using deep learning called BCV-Predictor. BCV-Predictor predicts the bug count vector of the next version software system; it is trained using metadata. To the best of our knowledge, no such work has been done in these aspects. We also address overfitting and class imbalance problems using the random oversampling method and dropout regularization techniques. We conclude that BCV-Predictor is conducive to predicting the bug count vector of the next version of a software project. We found five out of seven meta datasets reach more than 80% accuracy. In all seven meta datasets, Mean Squared Error (MSE) lies from 0.71 to 4.715, Mean Absolute Error (MAE) lies from 0.22 to 1.679, MSE and MAE over validation set lie between 0.84 to 4.865, and 0.22 to 1.709 respectively. We also compared the performance of BCV-Predictor with eleven baseline techniques and found the proposed approach outperformed most of the meta-datasets. The key contribution of this chapter is shown below.

(a) Regression problem identification and the formulation of objective function to predict the BCV of the upcoming version of software systems.

(b) We developed a novel architecture named BCV-Predictor that effectively predicts the BCV for the upcoming version of the software system, and we compared the performance with 11 baseline methods. It also addresses CIB and the overfitting problem. To the best of our knowledge, this is the first deep learning architecture-based model in software bug count prediction.

(c) We performed extensive experiments on a total of 31 different versions of 7 software projects. Metadata collection process using these software projects, a total of seven metadata projects are created.

1.5.3 DNNAttention: Cross-Project Defect Number Prediction Approach

Due to a lack of an adequate dataset, defects can be predicted by employing data from different projects to train the classifier called cross-project defect prediction (CPDP). Cross-project defect number prediction (CPDNP) is one step ahead of CPDP, in which we can also estimate the number of defects in each module of a

software system; we contemplate it as a regression problem. This chapter dealt with the CPDNP mechanism and suggested a CPDNP architecture by employing a deep neural network and attention layer called DNNAttention. We synthesis substantial data named cross-heap by utilizing an amalgamation of 44 projects from the PROMISE data repository. We fed the cross-heap into DNNAttention to train and evaluate the performance over 44 datasets by applying transfer learning. We have also addressed class imbalance and overfitting problems by employing multi-label random over-sampling and dropout regularization. We compared the performance of DNNAttention using mean squared error (MSE), mean absolute error (MAE), and accuracy over eight baseline methods. We found out of 44 projects, 19 and 20 have minimum MSE and MAE, respectively, and in 19 projects, accuracy yields by the proposed model surpasses exiting techniques. Moreover, we found the improvement of DNNAttention over other baseline methods in terms of MAE, MSE, and accuracy by inspecting 20% line of code is substantial. In most situations, the improvements are significant, and it has a large effect size across all 44 projects. The contributions of the chapter are shown below.

(i) We precisely formalize a problem statement and objective function over cross-project defect number prediction. It is the fusion of two different problems, cross-project defect prediction and software defect number prediction.

(ii) We proposed a prediction mechanism for a cross-project defect number called DNNAttention. It estimates the defect number vector, which consists defect value of every module in the target project using transfer learning. The defect number vector consists of defect information of each module in the project.

(iii) We uniquely synthesize a bigger dataset called cross-heap by the amalgamation of 44 projects from the PROMISE data repository for the source project.

(iv) DNNAttention utilizes deep neural network and attention layer; to the best of our knowledge, no such predictive mechanism has been employed in such a perspective. We compared the performance of the proposed model with eight baseline methods. We found that out of 44, 19, and 20 projects have the minimum MSE and MAE, respectively. Moreover, 19 projects have maximum accuracy.

1.5.4 Hybrid Regression Analysis: Entire Software Prediction

The overall cost of software development is detrimental to determining the utility of any software. However, the uncertain nature of this cost increases the risk associated with the software. Predicting the next version of the software system helps reduce this cost, allowing a better allocation of developers and testers for software development. This chapter develops a novel approach for predicting the number of bugs in the different modules of the next version of a software system. The proposed model is also predicting various software features associated with the next version. Our method predicts the bug count and the metric values for each module in the software using a computational framework consisting of two phases, the data augmentation phase and the next version prediction phase. We perform our experimentation on 8 public datasets from the PROMISE repository using all existing versions. To achieve unbiased results, we conducted each experiment 50 times and took a mean value of it. The proposed methodology has an accuracy of over 60% on 5 of the eight datasets, while MSE and MAE lie between 45.34% to 185.5% and 30.59% to 131.21%, respectively. We have also compared the proposed model's performance with eight baseline learning methods and found the proposed model significantly outperforms these techniques. The key contributions of this chapter are summarized as follows:

(I) We propose a new methodology for predicting the entire successive version of the software system (metrics and bug count for each module).

(II) We develop a novel architecture using a sequence to sequence model for predicting the bug count and normalized values of the software metrics in the corresponding software modules.

(III) We uniquely model various software systems in the time-series domain, utilizing the inherent relationship present across software modules in the adjacent versions.

1.6 Literature Review of Various SDP Methods

We provide here the necessary background for software fault prediction methodologies according to the taxonomy, as discussed in section 1.1. The thesis mainly

focused on significant and newer work in the SDP domain; we also outline some of the classical work.

1.6.1 Literature Review of Classification based Model

Software bug prediction (SBP) is a technique to figure out bugs in the software module by considering software metrics as parameters. SBP is a bridge between software bugs and software metrics. There are many software metrics proposed for SBP. McCabe [160], Halsted [87] and Chidamber & Kemerer's (CK) [39] are most widely used for SBP for an object-oriented software system. Some other software metrics which are widely used in SBP are code smell metrics [86], line of comments/-codes [10], context package cohesion metrics [293], web metrics [19], change metrics [123], mutation-based metrics [22], network metrics [149], modularization metrics [294] and cascading style metrics [20].

Many researchers investigated the influence of feature selection methods on defect prediction. Song et al. [235] suggested that feature selection is an essential phase of any defect prediction framework. Arar et al. [13] suggested a feature-dependent Naive Bayes approach and applied it over SDP. As many features are interrelated, the removal of any such features reduces the performance of the model. They employed Feature Dependent Naive Bayes (FDNB) classification method over the NASA dataset and calculated the dependence value between the pair of features. They found that the results obtained from the proposed model outperform the classical Naive Bayes technique. Shivaji et al. [222, 223] applied a filter-based and wrapper-based feature selection method over the NASA dataset to develop a fault prediction model. Xu et al. [282] conducted substantial experiments using 32 different feature selection methods on three public datasets. They concluded that the performance of K-PCA is the worst, among other methods. Ghotra et al. [77] extended his experiment using various twenty-one different classification models and thirty feature selection techniques over eighteen NASA and PROMISE datasets. They suggested that the co-relation-based filter-subset selection method has the best performance over every dataset.

The class imbalance problem hinders the satisfactory performance of the SDP model. Thus many oversampling [119], and undersampling methods [188] is introduced to resolve this problem. Kamie et al. [111] conducted experiments over four sampling methods and analyzed their performance. They conducted experiments on

two industrial datasets and found unbiased results. Bennin et al. [18] applied five statistical methods over six sampling techniques on ten public datasets and found extensively satisfying results. Tong et al. [249] applied a dropout regularization technique to avoid overfitting problems. Khoshgoftaar et al. [117] suggested a tree-based approach to avoid overfitting problems. Ratsch et al. [207] claimed that using the ensemble learning method can avoid overfitting problem.
Ensemble learning is mainly combined with multiple weak learners to enhance predictive characteristics and build strong learner [210]. Three well know types of ensemble learning are as follow: Bootstrap aggregating (Bagging) [24], AdaBoost [67], Breiman [24] and random forest [25]. EL techniques are also successfully applied in SBP through various models. Wang et al. [266] proposed an EL model using the class imbalance problem learning technique. Siers et al. [225] proposed an approach for SBP using cost-sensitive estimation and voting algorithm. Zheng et al. [295] also proposed using cost-sensitive estimation with boosting neural network. Ensemble learning was applied recently on imbalance data by [122], they employed ensemble learning over imbalanced data by [122] which produces unbiased results.
Deep learning is classified into three architectures: staked denoising autoencoders(SDA), deep belief network (DBN), and convolutional neural network (CNN). CNN is mainly applied in the domain of image processing [114, 128]. SDA with ensemble learning was applied in the SBP model by [249]; they used SDA for deep feature extraction from classical software metrics and two layers of ensemble learning. DBN productively applied to the domain of speech recognition [92]. Few software practitioners applied DBN to the SBP domain [285, 262]. Recurrent neural networks (RNN) were applied by Wang et al. [260] in the SBP model. A hybrid model using a deep neural network was built by Manjula et al. [156]. The iterative feature selection method was applied with RNN in SBP proposed by Turabieh et al. [250].

1.6.2 Literature Review over Cross-Version Defect Prediction

Grave et al. [79] proposed a linear regression-based model that predicts the number of bugs in software modules. They conducted experiments over a telecommunication system with software change metrics. They concluded that module age, change made over the module, and age of the change metrics all together produce better bug prediction accuracy. Ostrand et al. [182] suggested a model using negative binomial

regression (NBR) for the number of bugs prediction and bug density over a particular module. They have used two industrial software systems and LOC metrics to evaluate the model. They have concluded that NBR is significantly useful to predict the number of bugs in prediction models.

Similar work is also reported by Bell et al. [16], which enhances the accuracy of the existing model by 20%. Janes et al. [99] and Liguo et al. [289] also utilized the NBR in a similar aspect. Liguo et al. [289] applied NBR over object-oriented metrics of large telecommunication systems. Janes et al. [99] compared the performance of NBR-based model to the logistic regression-based technique. Fagundes et al. [59] presented the study over the number of bugs prediction model; the study performed over eight NASA datasets, using a zero-inflated prediction technique. They have used the error rate as a performance metric and found significant performances achievement. Afzal et al. [4] suggested a bug count model using genetic programming; the proposed model was trained over three industrial datasets, using weak fault count as an independent variable. Recently, Ye et al. [287] also utilized genetic programming over bug count, utilizing the information of function decomposition of source code.

Over the past few years, ensemble learning-based model has played a vital role in the number of bug count domains. Bagging, boosting, voting, and stacking was widely used in this era. Many researchers have suggested various models [164, 267] to estimate the bug count in a software system. Misirli et al. [164] presented an EL method over SBP applying Naive Bayes, artificial neural networks, and voting techniques. They concluded that the results of EL-based methods are more effective than classical SBP approaches. Zhen et al. [295] performed an experiment over the NASA dataset using boosting algorithm to produce a better bug prediction method. The most recent work on bug count performed by Rathor et al. [202], they have applied linear and non-linear combination of heterogeneous EL method over the PROMISE dataset. Chen et al. [37] illustrated the study over supervised and unsupervised learning methods; they performed experiments using these two categories of ML techniques and compared the results. Recently deep learning has been applied over software bug prediction; Wang et al. [260] proposed software reliability prediction using a recurrent neural network. Majula et al. [156] proposed a deep learning-based model to predict fault in a software system. Dam et al. [48] also proposed LSTM based model for fault prediction; they have also suggested more data instances will intensify the performance.

1.6.3 Literature Review over Cross-Project Fault Prediction and Attention layer

Cross-project defect prediction conquer the challenges of within-project software defect prediction (WPSDP). Ma et al. [150] suggested an approach based on transfer Naive Bayes (TNB); it addresses data distribution problem between the source and the target project. Peters et al. [192] introduces the peter filter and compares it with the Burak filter and applied on cross projects and found more optimal results. Nam et al. [175] extend the TCA; they proposed TCA+ in which the most optimal preprocessing method was selected by the heuristic method. They found TCA+ surpasses the performance over the state-of-the-art method. Chen et al. [34] proposed a deep neural network-based CPDP model; they have also utilized attention mechanism in their architecture, they found attention mechanism subjugates the problem of defect pattern generalization over target project. [186] employed six learning techniques to construct a model called CODEP; they utilized LR, RBF network, Multilayer perceptron, etc. Nagappan et al. [172] suggest an approach called HYDRA; they have applied genetic algorithm, ensemble learning as two different phases and found HYDRA exceeds the performance over TCA+. [142] introduced a two-phase transfer learning model. The first phase proposes a source project, and, in the second phase, it leverages the TCA+ to build two prediction models. Gong et al. [78] proposed a novel approach for the CI problem in CPDP; they have employed stratification embedded in nearest neighbor (STr-NN) in their architecture and conducted experiments over PROMISE repository data. They found the performance of models surpasses the baseline methods, and it also subjugates the CI problem.
Fan et al. [61] employed an attention mechanism along with RNN to construct the SDP model and found their proposed model outperforms classical ML models. Wang et al. [268] applied combined LSTM and attention in aspect level sentiment classification. They give more weight to the fine-grained task by using the attention layer. Zhou et al. [297] applied bi-LSTM with attention layer for relation classification. Hierarchical LSTM was applied with attention layer for image captioning [234]. They found the results are more accurate than existing LSTM based methods. Similar work was performed by Gao et al. [72]; they have employed the repeatable architecture over the video captioning problem. The combined approach were applied in many other real-world application such as 3d gesture recognition

[146], skeleton-based action detection [144], speech recognition [98], ironic sentence detection from Twitter data [157], etc,.

1.6.4 Literature Review Over Hybrid Regression and Deep Learning

The previous works have revolved around software bug prediction, bug count prediction, cross-version defect prediction using classical deep learning techniques, but no approach is proposed to predict the successive version of a software project or data associated with the next version of a software project. Deep learning architectures have been seldomly used for software bug prediction, despite their enormous success in various applications [54][106]. In the software domain, they have mainly focused on the classification task of predicting whether the software module is faulty or non-faulty and very limited towards the regression task of predicting the number of bugs. Deep learning applied in various area such as multiple language translational [55], disease detection detection [60, 166], speech recognition[177, 163]. Most of the big companies applied DL in their recommendation system, such as YouTube recommendation [45], movie recommendation [246]. Wang et al. [264] employed deep convolutional neural fields to predict protein structure. Similarly, Heffernan et al. [90] applied a deep neural network to predict protein secondary structure. Recent few works [278, 36] have been done over caption generation of a given image. Severyn et al. [219] conducted a sentiment analysis using Twitter data by employing a deep convolution neural network, and they found overwhelming results.

1.7 Structure of the Thesis

The thesis consists of eight chapters, Fig. 1.2 illustrates the overview and interconnections between every chapter. Chapter 1 starts with a brief introduction of software defect prediction and its different types. This chapter also discussed motivation, various research goals, a list of contributions, and a literature review. In the chapter 2 we will illustrate various preliminary information related to the work, different projects used in our work, various dataset descriptions, employed performance metrics, and baseline methods. In chapter 3 we will explain our first proposed work, which is related to classification-based SDP. After that, in chapter 4 we illustrate the regression-based work, which is mainly about cross-version defect number

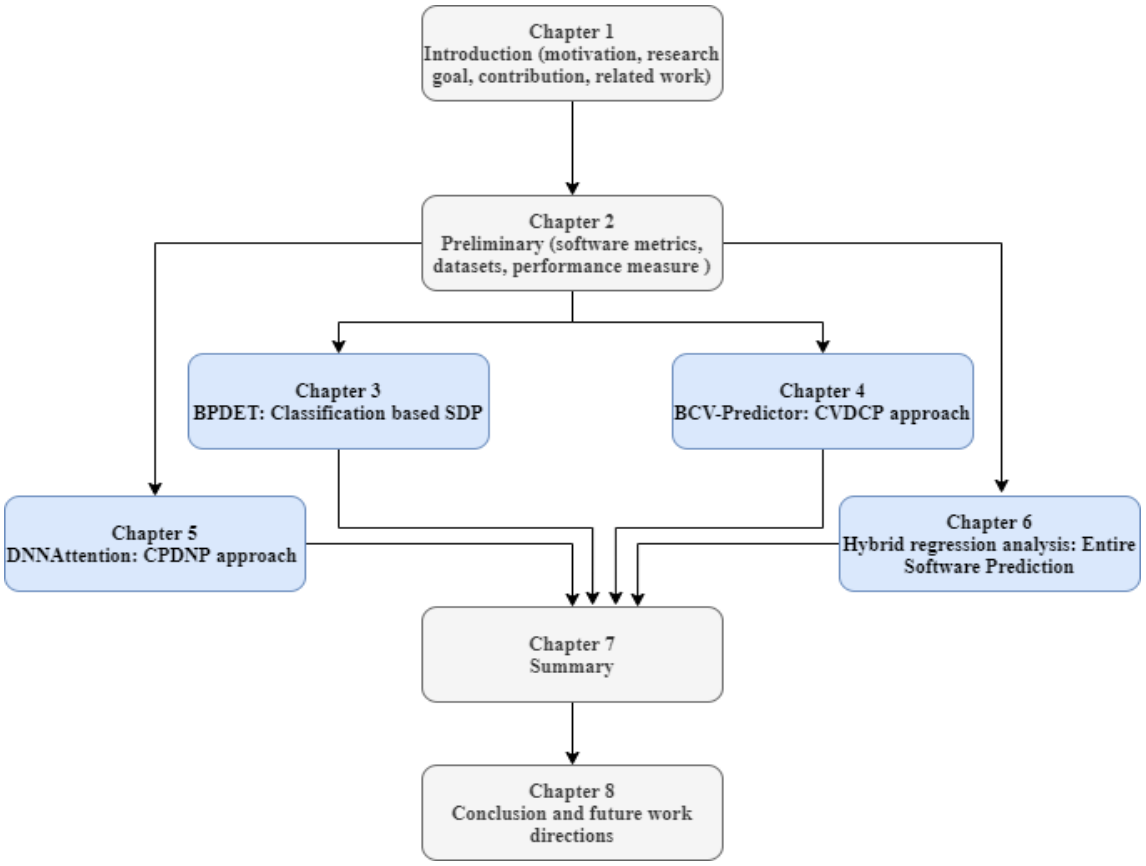

FIGURE 1.2: Thesis Structure.

prediction. Later on, we discuss cross-project defect prediction, followed by hybrid regression analysis-based approach in chapter 5, and chapter 6, respectively. We will discuss the summary of the thesis in chapter 7, and the conclusion and future direction in chapter 8.