CHAPTER 1
INTRODUCTION

Dependency on the software system by a human being is increasing drastically. Due to this dependency, higher reliability in the software products is needed. Hence, software developers are being forced to give more attention to improving software product quality regarding reliability. Defect density is a parameter used by the developer to measure the reliability of the product before its formal release. Efficient and early prediction of defect density enables a software developer to identify possible improvements in the product under development. It also motivates the development team to distribute the resources effectively in the development of different parts of the product.

1.1 OVERVIEW

Software failure rate is the major challenge for software developers. Recently, a report on software failure rate has been released by the Standish Group. This report was prepared on the basis of 50,000 projects around the world in the last five years. These projects are ranging from tiny enhancements to massive systems re-engineering implementations. The success rate is based on the factors such as on time, on budget with a satisfactory result. The outcomes of this study mentioned in Table 1.1.

<table>
<thead>
<tr>
<th></th>
<th>2011</th>
<th>2012</th>
<th>2013</th>
<th>2014</th>
<th>2015</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>29%</td>
<td>27%</td>
<td>31%</td>
<td>28%</td>
<td>29%</td>
</tr>
<tr>
<td>Challenged</td>
<td>49%</td>
<td>56%</td>
<td>50%</td>
<td>55%</td>
<td>52%</td>
</tr>
<tr>
<td>Failed</td>
<td>22%</td>
<td>17%</td>
<td>19%</td>
<td>17%</td>
<td>19%</td>
</tr>
</tbody>
</table>

For a successful development of software products, managing its quality is the prime concern of the developers. The management activities provide protection method that helps to prevent the occurrence of faults in the product. Avoidance or poor
implementation of these activities leads to product failure and create dissatisfaction to the
customer. The implementation and planning of these activities must occur throughout the
software development lifecycle.

Following are some examples of these activities:

- Defining defects.
- Classification of defects on severity basis.
- Inspection of documents, code, and other artifacts of the product.
- Testing code and recording the identified defects.
- Setting some acceptable quality levels as milestones.
- Perform and track the corrective actions, if milestones are not achieved.
- Analysis of defect data.

The effectiveness of software development process indicates by these control activities
using some quality control metrics measure. In most of the studies, Defect Density (DD)
dresses as a prime measure for software quality assessment. Software DD expressed as
number of defects in per Kilo Lines of Code (KLOC). The measurement of this metric
performs by the development team during the Software Development Life Cycle (SDLC).
Following are some defect density metric with their formulas given by Fenton and
Pfleeger (1997).

- Code Defect Density

  It indicates the quality of the source code provided to the testing team and the
  quality of the source code provided to the customer and given as:

  First, during the system testing, is represented as,

  \[
  DD = \frac{\text{Number of Defects in Testing}}{\text{Size in KLOC}}
  \]
Second, during the operation of the system, is represented as

\[ DD = \frac{\text{Number of Defects during operation}}{\text{Size in KLOC}} \]

- Defect removal efficiency

It indicates an effectiveness of the testing activities to identify and remove a maximum number of defects from the delivered software system, and represented as:

\[ DD = \frac{\text{Number of Defects in Testing}}{\text{Total number of defect (testing + operation)}} \]

### 1.2 SOFTWARE DEFECTS

A software defect is an error in the code or algorithm due to that the software product does not fulfill the user expectations and specified software requirements. Due to the error, the program might produce an unexpected outcome or causes malfunctioning [Mahanti and Chulani, (2007)]. Following are the basic ideas that explain the software defect:

- The defect is considered as a bug that is created in the application due to some mistake of a programmer.

- If the software product result deviated from the expected result that is defined in the software specification document, it is considered as a defect.

- If the software product is failing to satisfy or doesn’t meet the expectations of the end user, it can be considered as a defect. This dissatisfaction may cause due to an error in the methods or the code of the product during the development.
1.2.1 Difference between Defect, Failure, Fault, Error, and Problem

In our proposed approaches, presented in this thesis, defect density has been considered as a prime measure that affects the quality of the software product. The majority of researchers are synonymously using the defects with failure, fault, and error. In this section, a clear definition of these terms has been discussed based on the IEEE Standard [IEEE Std 1044, (2010)].

- **Defect**

  It is an indication of the failure of meeting requirements or specification due to a deficiency in the work product. This defective product either needs to be replaced or repaired.

- **Failure**

  It indicates the inability of the software product to execute the specified function. In another word, this can be considered as an event in which product component does not perform required functionality.

- **Error**

  It is an outcome of human mistakes that leads to an inappropriate result from the working product.

- **Fault**

  It is a symptom of an error in software.

- **Problem**

  It is termed as a difficulty experienced by a person, resulting from an unacceptability come across with a system in use.

In Table 1.1, the relationships among these software anomalies are described with some example.
<table>
<thead>
<tr>
<th>Pair of anomalies</th>
<th>Their relationship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem-Failure</td>
<td>One or more failure may create the problem</td>
</tr>
<tr>
<td></td>
<td>One or more problem may be the cause of a failure</td>
</tr>
<tr>
<td>Failure-Fault</td>
<td>A failure indicates the presence of one or more fault</td>
</tr>
<tr>
<td></td>
<td>A fault may become the cause of one or more failure</td>
</tr>
<tr>
<td>Fault-Defect</td>
<td>A defect is a super type of a subtype fault</td>
</tr>
<tr>
<td></td>
<td>Each fault is considered as a defect but not vise versa.</td>
</tr>
<tr>
<td></td>
<td>A defect is considered as a fault when it identified during the operation of the product and may lead to a failure.</td>
</tr>
</tbody>
</table>

1.2.2 **Software defect life cycle**

Defect life cycle, also known as defect journey, explain the different stages of the defect from its occurrence to its final verdict assigned to it. This defect life cycle may start or close at any stages/phases of the SDLC. The workflow of software defect life cycle is presented in Figure 1.1. Following are the different stages of a defect during its life cycle:

- New
  A potential defect that is logged or raised the first time is assigned as a new defect.

- Assigned
  Once the defect is raised and identified by the testing team, the head tester approved it as a bug and assigned to the developer team.

- Open
  This stage is also known as active where the developer is handling the defect and starts an analysis to fix it. At this stage, there are two possible decisions on defect either rejected/deferred or validate for next processing on it.
Fixed
After making some necessary modification in the code by the software developer, he or she assigns the state “fixed” to the defect.

Pending Retest
At this stage, the modification in the code for fixing the defect, the modified code needs to be retested by the tester. The modified code is given to the tester and the developer assigned the “pending retest” state to the defect.

Retest
The tester at this stage retests the modified code provided by the developer and assigns the “retest” status to the defect.

Verified
The tester retests the defect after fixed by the developer and if no new bugs are detected in the code, the tester assigns the “verified” status to the defect.
➤ Reopen

During retest, if a tester identifies that the defect still persists in the code then he may change the status to “Reopened.” Now the defect again goes through the defect life cycle.

➤ Closed

After retesting the code if the defect no longer exists then the tester assigns the status as “closed”.

➤ Duplicate

If the defect is repeated or the similar logic in the defect, the developer assigns the status to it as “duplicate.”

➤ Rejected

If the developer identifies that the defect is not authentic or genuine, the developer assigns the status as “rejected.”

➤ Deferred

If a developer thinks that the identified defect is not of prime importance or it may be handled at later stage of the development, he/ she assigns the status as “deferred.”

➤ Not a bug

If this defect is not affecting the functionality of the system in any way, the developer assigns the status as “not a bug”.

1.3 SOFTWARE MEASUREMENT

Even though, if a project is not in difficulty or ghastly condition, measurement of software is mandatory. It is impossible to rate our project health without measuring it.
Measurement is a mandatory aspect of software engineering as it gives the status of the product quality or readiness.

1.3.1 Direct and Indirect Measurement

The major difference in direct and indirect measurement of attributes is that direct measure does not require other attributes for measurement, on the other hand, an indirect measurement of attributes involve others attributes. For example, defect density (an indirect measure) is measured by involving two metrics i.e. software size and number of known defects. Whereas the size of the code, a direct measure, does not require others attribute to measure it. Some examples of direct and indirect measurement in software engineering are presented in Table 1.2

<table>
<thead>
<tr>
<th>Direct measure</th>
<th>Indirect measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lines of code</td>
<td>Programmer productivity</td>
</tr>
<tr>
<td>Number of known defects</td>
<td>Defect density</td>
</tr>
<tr>
<td>Duration of testing</td>
<td>Defect detection efficiency</td>
</tr>
<tr>
<td>Time of a programmer spends on project</td>
<td>Test effectiveness ratio</td>
</tr>
</tbody>
</table>

1.3.2 Measurement for prediction

At the point when we think to measure something, we typically imply that we wish to survey some elements that as of now exist. This kind of evaluation/estimation uses to identify what exist. This estimation also indicates about what has happen before. In a case, when the entity that we are seeking to measure is not available yet, measurement for prediction has to be performed. A mathematical model and prediction procedures are used together to obtain an unknown parameter(s) in a prediction system [Littelwood, (1989)]. These models also provide the methods for interpreting the obtained result. In this thesis,
five approaches have been proposed those are suggested to use for assessing the implications of independent variable(s) on the dependent variable in different scenarios.

1.3.3 Classification of Software measures

As depict in numerous studies, the primary commitment of any product estimation movement is to recognize the elements and ascribes we wish to quantify.

In the software measurement there are three such classes of measures:

- **Process**

  The measures in this class are used to assess the efficiency of the process used for development.

- **Product**

  These measures are the prime measures used by the quality management team. The measures in this class reflect whether the product is completed in all aspects to satisfy the user.

- **Project**

  The measures in this class are used by the project manager to track the progress of development.

Within each class of these measures, there is a need to distinguish between internal and external attributes. Internal attribute measure represents the internal quality of the product in terms of that attribute. Whereas, external attribute measure of an entity represents how that entity is related to the environment. In other words, the external attributes of the entity represent the behavior of that entity.
1.3.4 Software Quality Measurement

The quality assessment is mostly based on direct or indirect measures that we have already discussed in the previous section, few examples of some of these measures are:

- Defect-based quality measures
- Usability measures
- Maintainability measures

In this thesis, the major focus on the defect based quality measures where the defect density has been considered as a prime factor for quality assessment.

1.4 PREDICTION FOR QUALITY MANAGEMENT

Finding and settling defects is by and large the most expensive activity in software development. Higher efforts are applied in software verification and validation (V & V) activity to meet the acceptable level of quality in the software product. In testing, the vital part of V & V, near about half of the development cost is invested to achieve the quality in the software product [Bhattacherjee et al., (2007); Jones, (2001)]. For a safety-critical software system, the testing cost is more increases.

The software defect prediction approach enables a software developer to identify possible error-prone modules/classes in the software product. The testing team pays more attention to the coding part during the testing [Gopalani et al., (2012)]. This prediction approach provides an efficient way to increase the efficiency and effectiveness of software testing.

The measurement of software defect provides the following benefits to the software development team [Grady, (1996); Huang, (2003)]:

- The developer can evaluate the release readiness of the software product before its delivery to the customer,
- An effective software process improvement leads to higher quality development,
Effective planning and controlling the testing resources during the software development.

1.4.1 Software Defect Prediction

The prime concern of this thesis is to analyze an impact of different software metrics on software defect density using prediction approaches. As discussed in the previous section of this chapter, defect density is the ratio between known defects and size of the product; software defect data is an important source of information for formulating our research approaches. Software defect data provides support for taking the decision in software process improvement. In contrarily, ignoring this information leads to severe consequences for an organization’s business [Grady, (1996)].

Classification of software anomalies [IEEE Std 1044, (2010)] has already been discussed in section 1.2.1 of this chapter. Since the defect in software can prompt failure of the whole software system, which could be dangerous for health or life of system if it is a safety-critical system. Hence, most of the software companies aim to ensure no defects in the software product before its delivery to the customer.

1.4.2 Software Defect Prediction Techniques

Software defect prediction techniques are grouped into two categories i.e. the techniques used to predict the number of defects in the product module/class and the techniques used to classify the module/class either faulty or non-faulty. In proposed approaches presented in this thesis, uses defect density as an important parameter for assessment of software quality, therefore, we concern about the former technique of defect prediction. Following Figure 1.2 elaborates the different techniques for software defect prediction.
1.4.2.1 Techniques to predict number of defects

The techniques used for prediction of number of defects known as black box techniques and white box techniques. In the black box technique, previous values of a parameter are used to predict its future values. For example, number of discovered defects during development/testing is used to predict the number of latent defects in the product. These models don’t consider others attributes of the software for prediction. In white box technique, in addition to this parameter (discovered defects), other product attributes also consider for prediction of number of latent defects in the product. In the following paragraph, first, we discuss the two black box techniques followed by white box techniques.
Software Reliability Growth Models (SRGMs)

SRGM uses some mathematical formulas to estimate the reliability of a software system on the basis of the latent defects. These models use the defect data provided from the development and testing phase.

Initially, a suitable model is selected with the consideration of testing and development approach. Thereafter, another sub-model from the selected model will be used for making defects prediction. First, the SRGM models apply on the current defect data of the product for the fitting of a mathematical growth model by observing partial defect inflow [Bhattacharya and Mahanti, (2000)]. Further, the model is used for prediction of latent defects in the product. In a study of Wood A. (1996), he investigated the significant correlation between the pre-release defect count and post-release defect count by analyzing SRGM models.

Capture-Recapture Analysis

In this method, detected defect patterns are analyzed by independent defect detection activities, i.e. inspectors or inspection, with respect to testing.

Thereafter, the number of remaining defects is estimated according to the following equation [Briand et al. (2000)]:

\[ N(\text{estimated in work product}) = \frac{n(\text{inspector}_1) \times n(\text{inspector}_2)}{m(\text{number of defects found by both inspectors})} \ldots (1.1) \]

\[ \text{Remaining defects (est.)} = N(\text{estimated}) - N(\text{uniquedis covered}) \ldots (1.2) \]

Briand et al. (2000) analyzed the Capture-Recapture model to investigate the effectiveness of the number of inspectors. They identify that when the number of
inspectors is too small, no model is sufficiently accurate, and underestimation may be substantial.

- **Expert Opinions**

This approach is based on the experience of the expert, if he or she is available for analysis. The major limitation of this approach is its subjective nature. Hence, the results obtained from different opinions are not consistence. This approach is only used for prediction of number of defects at the product level. It should be avoided for prediction of defects at the small sub-systems/module level. Vincent et al. (2007) compared the performance of expert opinions with other SRGM models for prediction of software defects.

- **Causal Models**

These models are used to establish a causal relationship between software process and product attributes with number of defects expected to be found in the system. Fenton et al. (2008) investigate the weakness of traditional software metrics for forecasting of defects and omission of some causal factors. They proposed a model based on Bayesian Networks (BNs) to address the problem of these traditional forecasting approaches. A causal model based on the BNs for software defect prediction is shown in Figure 1.3.

- **Analogy-Based Predictions**

Analogy based estimation techniques rely upon collection and comparison of a range of metrics between the past and the current project for identifying the most analogous project [Mokus, (2003)]. For software defect predictions typically size, type of application, complexity of functionality and other parameters are used to identify similar projects to make the estimations. The analysis can be done at the project, sub-system or component level.
Figure 1.3: BNs for Software Defect Prediction [Fenton, (2008)]

- **Multivariate Regression**

These types of the model use some software metrics set as the predictor variable and establish the model based on statistical regression for prediction of software defects. In a work of Khoshgoftaar et al. (1993), multiple linear regressions have been used to predict the software changes using a set of software complexity metrics as independent variables. In another work of Khoshgoftaar et al. (1992) forecasted program faults using a set of code complexity metrics and number of changes as predictors.

- **Constructive Quality Model (COQUALMO)**

This model is based on software defect introduction and removal models [Boehm, (1981)] that are similar to the Tank and Pipe model [Jones, (1975)]. This model is represented as a composite model of Defect Introduction (DI) and Defect
Removal (DR). Figure 1.4 a) and b) describe both the sub-model used in COQUALMO.

**Figure 1.4: COQUALMO Model [Chulani and Boehm, (1999)]; a) DI sub model, b) DR sub model**

- **Correlation Based Models**

Similar to the previous models, correlation based models also use defect data available during the development and testing phase. The number of known defects discovered during development and testing is used as a source of information for prediction expected number of defects that may occur in the next
phase or iteration. Nanditha et al. (2015) propose a defect prediction model to control the quality of software products using statistical process control. The key contributors for building the prediction models were derived using Correlation and ANOVA based feature selection methods.

1.4.2.2 Techniques for defects classification

Apart from the approaches for prediction of number of latent defects, software defect classification techniques strive to recognize the faulty and non-faulty software artifact in the product. These techniques use different software product attributes to make this classification of software artifacts. Classification of defects is usually performed at the lower level of granularity such as file and class level. Following are the techniques used for defect classification

- **Logistic Regression**

  This approach is used to discriminate the software modules as faulty or non-faulty. Like another regression approach, this approach also uses some metrics information for classification of software modules. Khoshgoftaar and Allen (1999) introduce an approach that addresses the prior probabilities and costs of misclassification to a logistic regression-based classification in a software quality model. Zimmermann et al. (2007) also used regression approach to classifying the Eclipse projects file/package as faulty or non-faulty.

- **Machine Learning Models**

  These models use data mining techniques and some algorithm based statistical methods for defect classification. The models use some prediction variables as input information and provide the classification results of software modules/artifacts as faulty or non-faulty. Much of studies are using machine learning algorithms for software modules classification. Gondra (2008) perform the classification using the NASA project data set and obtained results as 72.6% correct classification.
1.4.2.3 Limitation of Prediction/Classification Techniques

In this thesis, we are investigating the relationship between some software product metrics with the software defect density. Since the defect density calculated using the number of known defects, the classification techniques are not worth for this investigation. On the other hand, there are many prediction techniques those are used for forecasting some defects in the software modules. There are some limitations of using some of these techniques. These limitations are mentioned in Table 1.3.

Table 1.4: Limitation of defects prediction techniques

<table>
<thead>
<tr>
<th>Technique</th>
<th>Limitations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software Reliability Growth Models</td>
<td>• Expected total number of inherent faults calculated by SRGM is highly sensitive to time to failure data.</td>
</tr>
<tr>
<td></td>
<td>• It seems that sufficient amount of failure data needs in SRGM, but it is impossible always to provide required amount of failure data during development and testing phase. [Kim et al., (2006)]</td>
</tr>
<tr>
<td>Capture-Recapture Analysis</td>
<td>• It performed best when stringent assumptions are made.</td>
</tr>
<tr>
<td></td>
<td>• Relaxing assumption leads to complicated estimation.</td>
</tr>
<tr>
<td></td>
<td>• Robust when simply used to predict whether a criterion for re-inspection has been exceeded. [Briand and Freimut, (2000)]</td>
</tr>
<tr>
<td>Expert Opinions</td>
<td>• This method is subjective in nature.</td>
</tr>
<tr>
<td></td>
<td>• Inability to scale down properly at lower levels of granularity.</td>
</tr>
<tr>
<td></td>
<td>• Required higher consistency in the opinions.</td>
</tr>
<tr>
<td>Causal Models</td>
<td>• The required causal relationship between software process and product attributes. [Fenton and Neil, (1999)]</td>
</tr>
<tr>
<td>Analogy-Based Predictions</td>
<td>• Past similar project metrics information is required.</td>
</tr>
<tr>
<td></td>
<td>• Higher consistency needs to be maintained. [Mokus,(2003)]</td>
</tr>
<tr>
<td>COQUALMO</td>
<td>• Expert determined sub-models are used for making a composite model.</td>
</tr>
<tr>
<td></td>
<td>• Classification of defects according to their origin SDLC phase is required. [Chulani and Boehm, (1999)]</td>
</tr>
</tbody>
</table>
In this thesis, multivariate regression, and correlation-based models have been used for the formulation of defect density prediction model. In chapter 5 and chapter 6, correlation based approach uses for making the prediction of software defect density with the help of static code metrics and hybrid metrics suite. In chapter 7, multivariate regression approach has been used to estimate the prediction performance of defect density by different repository metrics in open source software.

1.5 OBJECTIVES OF THE RESEARCH WORK

This thesis is devoted to providing various approaches for making a prediction of software defect density. The ultimate goal is to identify the prediction performance of different software product metrics of software defect density both for in-house software and open source software. The main objectives of the thesis are

1. To propose an effective module size distribution model for minimization of software defect density of an in-house product.

2. To formulate an effective model for reduction of defect density by relationship of size with defect density.

3. To empirically investigate the impact of static code metrics on the software defect density.

4. To analyze the effect of Hybrid Metrics Suite On Defect Density

5. To propose a model for prediction of defect density in open source software using some easily available repository metrics.
1.6 ORGANIZATION OF THE THESIS

The work reported in thesis is organized into eight chapters, as given below:

CHAPTER-1: INTRODUCTION

This chapter provides the introduction to the different approaches for prediction of software defect density. A brief discussion of a software product, process, and project metrics is also covered. This chapter describes the relationship of defect density with some important product metrics.

CHAPTER-2: LITERATURE SURVEY

This chapter presents an in-depth literature review on software defect density, different software metrics suite, and classification technique for prediction of module/class fault-proneness. The chapter also reviews the dependency of software faults on a different product, process, and product metrics.

CHAPTER-3: INVESTIGATION OF MODULE SIZE DISTRIBUTION IMPACT ON DEFECT DENSITY

In this chapter, a model has been proposed that established the relationship between software defect density and module size distribution. To empirical investigate the model; three data sets from different source have been collected. The analysis results conclude that the distribution of module size in geometric progression gives a reduction in defect density [Verma and Kumar, (2014)].

CHAPTER-4: FORMULATION OF AN EFFECTIVE MODEL FOR DEFECT DENSITY REDUCTION

In this chapter, a model has been proposed which shows that the defect density decreases with increasing module size. The model has been analyzed with three data sets where the distribution of the modules is done according to the size. A relationship has been established between defect density and module size, which clearly indicates that the
defect density decreases at a higher rate with respect to module size [Verma and Kumar, (2016)].

CHAPTER-5: EMPIRICAL VALIDATION OF DEFECT DENSITY PREDICTION USING STATIC CODE METRICS

In this chapter, the relationship has been established between the defect density and the source code metrics that used to rate the prediction performance of defect density using these static code metrics [Verma and Kumar, (2015)].

CHAPTER-6: EFFECT OF HYBRID METRICS SUITE ON DEFECT DENSITY: AN EMPIRICAL ANALYSIS

In this chapter, a hybrid metrics suite using all the traditional metrics suites has been formulated. The effect of the metric suites has been analyzed on the defect density using a regression model.

CHAPTER-7: PREDICTION OF DEFECT DENSITY IN OPEN SOURCE SOFTWARE USING REPOSITORY METRIC SUITE

In this chapter, a relationship of defect density with different repository metrics of open source software has been established with the significance level. Five repository metrics namely Size of the project, Number of defects, Number of developers, Number of downloads, and the Number of commits have been identified for predicting the defect density of open source project. Simple and Multiple Linear regression have been performed on 62 open source software available at sourceforge.net. The result reveals a statistically significant level of acceptance for prediction of defect density by some repository metrics individually and jointly.

CHAPTER-8: CONCLUSIONS AND SCOPE OF FUTURE WORK

This last section of thesis concludes the study to address the identified problems and discuss the scope for enhancement in future.