Abstract—Software product measures have been widely used to predict software defects. Though these measures help develop good classification models, studies propose that relationship between software measures and defects still needs to be investigated. This paper investigates the relationship between software measures and the defect prone modules by studying associations between the two. The paper identifies the critical ranges of the software measures that are strongly associated with defects across five datasets of PROMISE repository. The paper also identifies the ranges of the measures that do not necessarily contribute towards defects. These results are supported by information gain based ranking of software measures.

Index Terms—Metrics/Measurement, Product metrics, Software Quality, Association mining

I. INTRODUCTION

Software defect prediction techniques either classify a software module as defect-prone (or not defect-prone) or predict the number of defects in a software. Both types of models take software measurements as input. The models help software project managers in controlling the software project, resource planning and test planning [23], [28].

Literature suggest early identification of defect-prone modules in order to improve software process control and achieve reduced defect correction cost and effort, and high software reliability [19], [13], [27], [30]. Most of the software quality prediction models use software product measures for defect prediction [20], [21], [6], [36]. Primary reason for this is availability of public domain data. Another reason is that measurements collected from software product code normally perform better than the design measures [14]. However, use of certain product code measures has been shown ineffective for prediction of defects in object-oriented paradigm [29].

Despite the large amount of work done so far the models based on product measures are seldom used in industry because of their inability to capture exact causes of defects and the exact relationship between software measures and software defects at the best remains unknown.

This study uses Association Rule Mining (ARM) to study the relationship between software measures and software defects using public domain datasets [3]. Association Rule Mining (ARM) is an important data mining technique and is employed for discovering interesting relationships between variables in large databases [31]. ARM is used to find interesting correlations, frequent patterns, associations or casual structures among items of large datasets [33]. The study identifies the attributes and their ranges that co-occur with defects. These ranges can be used for better planning of resources and testing. The ranges can further help study the relationship between the ranges and software defects. This paper also shortlists certain attributes and ranges that do not necessarily cause defects. Frequency of attributes with critical ranges is calculated among all datasets under study to find importance of each attribute. Attributes with critical ranges are more important than the attributes with indifferent ranges. The importance of these attributes is verified through Information Gain (IG) based attribute ranking. The identified attributes that have critical ranges are ranked high whereas the attributes that do not contribute towards defects are ranked low by the IG based approach.

Rest of the paper is organized as follows: section II discusses the related work, section III presents our research methodology and our approach to find focused itemsets. Section IV present the results of the experiments and these results are discussed in section V. Section VI concludes the paper and presents the future directions.

II. RELATED WORK

Empirical studies have been performed to investigate relationships between software measures and defects. Different studies have investigated the causes of defects, selected the software measures that are important to find defects and identified the software measures that do not help in classification of defect prone modules. Further, use of association mining for defect prediction has also been reported. Rest of the section discusses these studies.

Bayesian Belief Networks (BBN) have been used to discover causes of defects. Neil et. al have highlighted the importance of factors like contextual information to improve prediction of software defects [24]. They have used a combination of software process and product measures to build BBN. Their expert driven BBN was capable of making better prediction on the basis of contextual information. Fenton et. al [11] have criticized existing models for being unable to take a holistic view while predicting defects. For example, certain models predict defects using size and complexity metrics whereas others use testing metrics only. Consequently ignoring the potential predictors like process measures. Moreover, these
models do not take into account the relationship between software measures and defects. Fenton et al. have presented a BBN to overcome the aforementioned limitations. Their model has shown encouraging results when used to predict defects in multiple real software projects [10]. The BBN has been improved to work independent of software development lifecycle. [12]. BBN approach has also been employed to predict software quality in [1], [9], [25], [13].

Causal relationship between software measures and defects is important in understanding and improving software processes [4], [5], [7]. Thapaliyal et al [34] have shown that in object oriented paradigm, ‘Class Size’ is a metric that has strong positive relationship with defects but ‘Coupling between Objects’ and ‘Weighted Methods per Class’ are insignificant to be used to predict defects. This study employed weighted least square model for empirical analysis.

Where many studies find the important software attributes that help defect prediction, software attributes have also been studied to identify the attributes that are not good predictors of defects. Rana et al. have compared certain defect prediction models with and without Software Science Measures (SSM) [29]. Their results show that models developed without SSM had better performance.

Association Rule Mining (ARM) has been useful to predict software defect correction effort and determine association among software defects [32], [26], [18]. An association rule based classifier, CBA2, have been empirically evaluated to predict software defects [2]. Accuracy and comprehensibility of CBA2 has been comparable to C4.5 and RIPPER, two recognized defect classification models. Kamei et al. have presented a hybrid approach to classify the software module as fault-prone or not-fault-prone [17]. The hybrid of ARM and logistic regression has performed better in terms of lift, however, its performance has been inferior in terms of support and confidence when compared with the individual models based on logistic regression, linear discriminant and classification tree. Association rules have been employed to identify the action patterns that may cause defects [8]. Each rule represents actions as antecedent and number of defects as consequent. The antecedents can be of numeric or categorical type, whereas the consequent is discretized as low, medium and high. Actions co-occurring with defects are used to avoid future defects. The proposed approach has shown promising results when used for a business project.

### III. Methodology

In order to apply association mining, the data needs to be discretized. Defect prediction data is dominated by non-defect prone modules and frequent itemsets associated with defects is not straightforward. Wang et al. have proposed the partitioning of data to solve similar problems [35]. Inspired by their approach we have partitioned the data into two; one partition \( D_l \) includes defect prone instances whereas the other partition \( D_f \) consists of not-defect prone instances. Frequent itemsets are generated in each partition and support of each itemset is calculated to gauge its usefulness and strength. Afterwards focused itemsets are selected from partition \( D_l \) and IG based validation is performed. Figure 1 shows the methodology and rest of the section discusses each of the steps in detail after discussing the datasets.

#### A. Datasets

This study uses public datasets from PROMISE repository [3]. These datasets have facilitated defect prediction studies. Each instance in a dataset represents a software module and the attributes are software measures calculated for that module. Each dataset has 22 attributes, including one class attribute which indicates if the software module is defect prone (D) or not-defect prone (ND). Most of the attributes are quantitative in nature. Table I lists the datasets, number of instances in each dataset and percentage of negative instances in each dataset. The percentages show that the datasets are dominated by ND modules. Software measures used in the datasets and their definition are given in Table II.

#### B. Discretization

Frequent itemsets are generated from categorical attributes only, therefore, the data needs to be discretized. Our approach discretizes the data by dividing each quantitative attribute into equi-frequency bins [16]. In other words, during discretization, each attribute is divided into intervals.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Language</th>
<th>Instances</th>
<th>%age of ND Modules</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cm1</td>
<td>C</td>
<td>498</td>
<td>90.16%</td>
<td>A spacecraft instrument of NASA</td>
</tr>
<tr>
<td>jm1</td>
<td>C</td>
<td>10885</td>
<td>80.65%</td>
<td>Real-time prediction for ground system</td>
</tr>
<tr>
<td>pcl</td>
<td>C</td>
<td>1109</td>
<td>93.05%</td>
<td>Flight software for earth orbiting satellite</td>
</tr>
<tr>
<td>kcl</td>
<td>C++</td>
<td>2109</td>
<td>84.54%</td>
<td>Storage management for ground data</td>
</tr>
<tr>
<td>kc2</td>
<td>C++</td>
<td>522</td>
<td>79.5%</td>
<td>Processing of science data</td>
</tr>
</tbody>
</table>

### Table I

Datasets used in this study: Characteristics of project taken from [15], [22]
TABLE II
SOFTWARE MEASURES REPORTED IN SELECTED DATASETS

<table>
<thead>
<tr>
<th>Software Measure</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>vi(G)</td>
<td>Cyclomatic Complexity</td>
</tr>
<tr>
<td>ev(G)</td>
<td>Essential Complexity</td>
</tr>
<tr>
<td>iv(G)</td>
<td>Design Complexity</td>
</tr>
<tr>
<td>LOC</td>
<td>McCabe’s Lines of Code</td>
</tr>
<tr>
<td>V</td>
<td>Program Volume</td>
</tr>
<tr>
<td>L</td>
<td>Program Level</td>
</tr>
<tr>
<td>D</td>
<td>Program Difficulty</td>
</tr>
<tr>
<td>I</td>
<td>Intelligence Content</td>
</tr>
<tr>
<td>E</td>
<td>Effort</td>
</tr>
<tr>
<td>B</td>
<td>Error Estimate</td>
</tr>
<tr>
<td>T</td>
<td>Programming Time</td>
</tr>
<tr>
<td>N</td>
<td>Program Length in terms of Total_Op and Total_Opnd</td>
</tr>
<tr>
<td>LOCode</td>
<td>Halstead’s Lines of Code</td>
</tr>
<tr>
<td>LOCComment</td>
<td>Lines of Comment</td>
</tr>
<tr>
<td>LOBlank</td>
<td>Blank Lines</td>
</tr>
<tr>
<td>LOCodeAndComment</td>
<td>Lines of Code and Comment</td>
</tr>
<tr>
<td>Uniq_Op</td>
<td>Unique Operators</td>
</tr>
<tr>
<td>Uniq_Opnd</td>
<td>Unique Operands</td>
</tr>
<tr>
<td>Total_Op</td>
<td>Total Operators</td>
</tr>
<tr>
<td>Total_Opnd</td>
<td>Total Operands</td>
</tr>
<tr>
<td>BranchCount</td>
<td>Total Branches in Program</td>
</tr>
</tbody>
</table>

C. Partitioning

As mentioned earlier, the partitioning step horizontally divides a dataset into two parts, $D_i$ and $D_f$. Partition $D_i$ comprises of records having class attribute value $\text{defects} = \text{true}$. Partition $D_f$ comprises of records having class attribute value $\text{defects} = \text{false}$. During this step, each dataset $D$ is divided into two subsets such that $|D| = |D_i| + |D_f|$, $D = D_i \cup D_f$, and $D_i \cap D_f = \phi$

D. Generating Frequent Itemsets

We have applied Apriori algorithm [16] on each partition to find frequent itemsets. An itemset is frequent if it is occurring many times in a partition and satisfies a minimum Support threshold. As mentioned earlier, an item in our case is an interval, so this step finds out those intervals of all attributes that co-occur frequently either with $D$ modules in partition $D_i$ or with $ND$ modules in partition $D_f$. It is pertinent to mention here that these frequent itemsets can have different lengths. An Itemset with one item is called 1-Itemset and an Itemset with $k$ items is known as $k$-Itemset. A 1-Itemset is essentially one interval from the range of values an attribute can have.

Our approach focuses on special itemsets, which do not include the class attribute. Normally, Apriori algorithm does not distinguish between class and non-class attributes and uses the discretized values of a class attribute to generate frequent itemsets. A class attribute is included in a frequent itemset if it is associated with another item in the data. Our approach requires the Apriori algorithm to generate only those itemsets which are individually or collectively associated with class attribute (i.e. Defects).

E. Calculating Support of Itemsets

Support count $\text{Count}_i$ of an itemset $\text{iset}_i$ is the frequency of its occurrence in a partition. If $m$ is total number of independent attributes and $n$ is the number of intervals for each attribute, Support of $\text{iset}_i$ in a partition $D_j$ is calculated as follows:

$$\text{Support}_i = \frac{\text{Count}_i}{|D_j|} \times 100$$

where $i \leq m \times n$ and $j \in \{t, f\}$. By convention, value of $\text{Support}_i$ varies from 0% to 100% instead of 0 to 1.0 [16].

F. Selecting Focused and Indifferent Itemsets

In order to find critical ranges for each attribute, we need to identify itemsets in partition $D_i$ that satisfy a minimum support threshold $\alpha_i$. At the same time a similar threshold $\alpha_f$ of minimum support is applied on itemsets in partition $D_f$. The itemsets (or intervals) with $\text{Support} \geq \alpha_i$ and $\text{Support} \geq \alpha_f$ are itemsets of interest and are called $\text{Itemset}_i$ and $\text{Itemset}_f$ respectively. Focused itemsets appear in $\text{Itemset}_i$ only and do not appear in $\text{Itemset}_f$ whereas indifferent itemsets appear in both:

$$\text{Indifferent} = \text{Itemset}_i \cap \text{Itemset}_f$$

$$\text{Focused} = \text{Itemset}_i - \text{Indifferent}$$

Attributes with indifferent bins indicate that they do not facilitate classification and can be dropped before developing a defect classification model. The attributes with focused itemsets are good indicators of defects and these ranges of values further need to be studied in order to get a better understanding of causes of defects.

G. Evaluation Framework

Identification of focused and indifferent itemsets need to be validated. For each dataset, we compare the attributes with focused itemsets with a list of all attributes ranked with respect to their Information Gain (IG) [16]. The attributes with focused itemsets should be ranked higher whereas the attributes with indifferent itemsets should be ranked lower by the IG based ranking approach.

IV. RESULTS

Results reported in this section have been obtained using Weka [37]. We have performed the same set of actions on each dataset and have recorded the performance. As a first step, we have discretized the datasets and divided each attribute into ten equi-frequency bins [16]. Each dataset is then partitioned into $D_i$ and $D_f$ as discussed in section III-C. Within each partition we have applied Apriori algorithm with minimum Support values shown in Table III to generate frequent itemsets. Table III also shows number of itemsets with different length. Results discussed here are with 1-Itemsets only. Afterwards, frequency of occurrence, $\text{Support}_i$, of each of the 1-Itemsets is calculated. Table IV shows top five 1-Itemsets and their $\text{Support}_i$ in each partition. The itemsets in boldface are focused itemsets whereas the itemsets in italic are indifferent itemsets. The focused itemsets present the intervals (ranges)
The focused itemsets do not only identify the critical ranges and are very important for software managers and researchers. If, during a software project, values of the mentioned software measures fall in critical ranges, this should raise an alarm and the project schedules, resource plans, and testing plans should be adjusted accordingly. Further a defect prediction model can use the critical ranges for improved classification defect-prone modules.

The focused itemsets do not only identify the critical ranges for each software measure, they also indicate the software measures that can improve the detection of defect-prone modules. Some attributes have at least one focused itemset in all dataset, hence showing strength of that attribute in prediction of defects. Total vote counts for each attribute are reported in Table VI. An entry with 1 indicates that the attribute has at least one itemset for the respective dataset. An entry with 0 represents that there was no focused itemset from that dataset.

In order to identify the indifferent and focused 1-Itemsets we have used the thresholds given in Table V. This can be noticed that in all cases $\alpha_1 \leq \alpha_f$ in Table V. This is because of the fact that there is less data regarding defect-prone modules hence the itemsets associated with defects have low support. So in order to pick a reasonable number of focused itemsets, we had to keep $\alpha_1$ very low.
this attribute for the respective dataset. A dash (-) represents that the attribute did not appear as frequent itemset at all. Across five datasets, the majority vote reveals that loc, n, v, d, i, e, b, t, IOC, uniq_Op, uniq_Opnd, total_Op and total_Opnd consistently contribute in occurrence of defects. Whereas ICodeAndComment, ev(g) and IOComment do not necessarily cause defects.

In addition to identifying the focused itemsets, the experiments have also identified some ranges that are neutral to occurrence or presence of defects. ICodeAndComment = \((-\inf - 0.5]\), ev(g) = \((-\inf - 1.2]\), and IOComment = \((-\inf - 0.5]\) are few examples. ICodeAndComment = \((-\inf - 0.5]\) appears in all datasets without exception. Support4 for this itemset is very high for both partitions with minimum Support4 = 46.75% for partition D1 and Support4 = 76.45% for partition D1 of dataset pe1. ICodeAndComment is a count of all lines of code including comments. The interval represented by this itemset shows very low values of this measure. This indicates that when a software is small in size, measured in terms of lines of line code and comments, this measure does not contribute in occurrence or absence of defects. Low values of ev(g) (i.e. ev(g) = \((-\inf - 1.2]\)) also appears as an indifferent itemset in four datasets.

The identification of attributes with focused and indifferent itemsets is supported by an information gain based attribute ranking. We have used Weka to perform an information gain based attribute ranking to validate our results. We have calculated attribute ranks for each dataset and have taken an average for each attribute. The attributes are then sorted w.r.t. their average ranks and the attribute with highest average rank value is assigned rank 1. The Table VI also shows IG-based ranks for all the attributes. If we analyze the attributes with TotalVotes \(\geq 4\), all the attributes except uniq_Op are ranked in top 11 by IG-based rankings as well. Similarly the bottom five attributes are common in TotalVotes and IG-based ranking. Unlike IG-based attribute ranking technique, our approach identifies critical ranges for the significant attributes.

### VI. Conclusion and Future Work

Software measures have been investigated over the years for software defect prediction. We study the relationship between software product measures and software defects. We have selected public datasets and discretized the data to study associations of software measures and defects. From the discretized data we have generated frequent itemsets and identified the 1-Itemsets strongly associated with defects. We call these 1-Itemsets the focused itemsets. The 1-Itemsets that are strongly associated with both the presence and absence of defects are called indifferent itemsets. The indifferent itemsets should not bother software managers whereas the focused itemsets serve as critical ranges. If software measurements for a project fall in these intervals the managers should take steps to avoid defects in the respective modules. These critical ranges are further helpful in development of defect prediction models. Analysis of the focused itemsets across five datsets shows that very high ranges of loc, n, v, d, i, e, b, t, ICode, uniq_Op, uniq_Opnd, total_Op and total_Opnd consistently contribute in causing defects. Whereas ICodeAndComment, ev(g), iv(g) and IOComment do not necessarily cause defects. These findings are supported by information gain based attribute ranking. In addition to identification of the attributes highly associated with defects, this study also identifies the critical ranges of these attributes.

These results cannot be generalized for being produced using five datasets only. We intend to validate these results using more public datasets. We also plan to study the association between software defects and longer itemsets.
ACKNOWLEDGMENT

The authors would like to thank Lahore University of Management Sciences (LUMS) and Higher Education Commission (HEC) of Pakistan for supporting the research.

REFERENCES