2007

Classification of software components based on clustering

Swetha Reddy Konda
West Virginia University

CLASSIFICATION OF SOFTWARE COMPONENTS BASED ON CLUSTERING

Swetha Reddy Konda

Thesis submitted to the
College of Engineering and Mineral Resources,
West Virginia University
in partial fulfillment of the requirements
for the Degree of

Master of Science
in
Computer Science

Committee Members
Katerina Goseva Popstajanova, Ph.D., Chair
Hany Ammar, Ph.D.
James Mooney, Ph.D.

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2007

Keywords: Clustering, Clustering tree, Decision Trees, Homogeneous groups

Abstract

Classification of Software Components based on Clustering

Swetha Reddy Konda

This thesis demonstrates how in different phases of the software life cycle, software components that have similar software metrics can be grouped into homogeneous clusters. We use multi-variate analysis techniques to group similar software components. The results were applied on several real case studies from NASA and open source software. We obtained process and product related metrics during the requirements specification, product related metrics at the architectural level and code metrics from operational stage for several case studies. We implemented clustering analysis using these metrics and validated the results. This analysis makes it possible to rank the clusters and assign similar development and validation tasks for all the components in a cluster, as they have similar metrics and hence tend to behave alike.

Acknowledgement

First and foremost, I would like to thank my advisor, Dr. Katerina-Goseva Popstajanova for her guidance and patience while helping me work on this research. She was always there to give me valuable suggestions, without which my thesis would not have taken proper shape. In addition, I would like to thank my committee members Dr.Hany Ammar and Dr.Jim Mooney for their support and valuable time. Also, I would like to thank Dr.Tim Menzies for providing assistance in implementing clustering using WEKA and obtaining decision trees.

My sincere gratitude is also expressed to Christina Moats and Kenneth Costello for their guidance and assistance at the NASA IV & V facility. I would like to thank Walid Abdelmoez for his work on reliability and maintainability based risk assessment. Arin Zahalka and Margaret Hamill provided more assistance to me in giving me information required to implement clustering on case study Indent.

This work is funded in part by a grant from the NASA Office of Safety and Mission Assurance (OSMA) Software Assurance Research Program (SARP) managed through the NASA Independent Verification and Validation (IV & V) Facility, Fairmont, West Virginia. I thank them for their support. Last but certainly not least, I would like to thank my parents, friends, Swetha, Richa and Mayank and my husband Vinay for their support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1: Introduction

Clustering involves organization of collection of patterns into meaningful clusters based on their similarity. Software modules are grouped according to the value of their software metrics in clustering. We assume that the components that have similar metrics behave alike and hence are grouped together into clusters. It is useful to know the behavior of the software components and classify them as we could assign similar activities to all the components in a cluster and rank the clusters. We implemented clustering in different phases of software life cycle and classified the software components into homogeneous clusters.

Software Integrity Level Assessment Process (SILAP) is the current state of practice at NASA that is done early in software life cycle during the requirements specification. SILAP uses some of the definitions from the COCOMO model to define complexity criteria and uses domain expert's knowledge to assign score to several Product/Process metrics of the software components. We implemented clustering on the software components of 12 real NASA projects, using the process related metrics defined in SILAP.

Also, we implemented clustering using the design metrics obtained in the architecture level derived from the unified modeling language (UML) on a case study, CM1 from the Data Metrics Program [33]. CM1 is a software component of a data processing unit in an instrument, used to exploit data to probe the early universe.

We also clustered the components of on open source software, Indent using the code metrics obtained in the operational stage. Indent has 9 files totaling about 7000 lines of code. It is used to beautify the C code. Running it has no effect on the functionality of the code but makes the results aesthetically pleasing and more readable. Our results demonstrate that classification of software components into meaningful homogeneous clusters can be done in different phases of the software lifecycle.

The rest of the thesis is organized as follows: Related work and our contributions are discussed in chapter 2. In addition, chapter 3 provides the background on clustering algorithms, classifiers that we used and explains the meaning of decision trees. In chapter 4 we discuss ways to classify components early in software life cycle during the

requirements specification and present the results obtained on 12 real NASA projects using the metrics obtained from the current state of practice in NASA called the Software Integrity Level Assessment Process (SILAP). Chapter 5 presents results of classification of software components in the architectural level on a case study, CM1, based on the design metrics obtained from the Unified Modeling Language (UML). Chapter 6 discusses the case study Indent and presents the results of classification of its components, based on the code metrics obtained in the operational stage. Chapter 7 presents our conclusions and lessons learned.

Chapter 2: Related Work

In this chapter we summarize the related work, and also discuss how we classified components into homogeneous clusters with clustering. Although clustering has been used for the classification of components [21], [8], [7], most of the previous work implemented it later in the software life cycle, as the details required are not available until later stages of design phase. Most of the previous works implemented clustering on large sample datasets. Very few implemented clustering on small size dataset [15].

In [21], unsupervised learning clustering techniques such as k-means and Neural gas clustering algorithm were used to analyze the software quality in the absence of fault proneness labels. Clustering algorithms can group software modules according to their values of software metrics. Software fault measurement metrics were used for clustering. The software engineering assumption is that fault prone software modules will have similar software metrics and so will likely form clusters. Similarly, not fault-prone modules will likely group together. When the cluster analysis is complete, a software engineering expert labels it fault prone or not fault prone. Data sets from two NASA projects JM1 and KC2 were used as empirical case studies. JM1 has 8850 and KC2 has 520 software modules. The software measurements and fault data were obtained at the program function, subroutine or method levels, so a software module is a program function, a subroutine or a method. Clustering was implemented on these software modules to analyze the software quality.

Most of the Clustering techniques used in the previous work worked well for large data sets. In our work our case studies had a small size dataset, so we did research on a method that works well on small size dataset. One of the previous works that used clustering to classify small size dataset was [15]. In [15], clustering using Wards method was implemented for identifying clusters in small dataset of journals based on five citation flows. This paper suggests that hierarchical clustering techniques, Wards minimum variance method or simple average method works well for small size dataset. Journals that were clustered together are deemed to be cohesive..

Another work used clustering to cluster the software execution profiles and predict failures [7]. The case studies used in this paper included the Java word count

program, the Java directory listing program, the Java regular expression parser and regular expression finder, the java pretty printer and the GNU Collection Compiler (GCC) version 2.95.2.They found that clustering isolates the failures and observed that a considerable number of failures were isolated in small clusters of executions. In [8], Podgurski et al used GCC case study which has 330,000 lines of code, and another case study called Lilypond which has 48,000 lines of code and implemented clustering algorithms. The cluster analysis revealed that execution profiles of failures typically have unusual profiles. All clustering of executions in this study was done using agglomerative hierarchical clustering algorithm, later in the software life cycle to identify failures in execution profiles and classify them.

Clustering results presented in this thesis illustrate that it can be used for identifying homogeneous clusters in the software components based on the software metrics available, in different phases of the software life cycle.

- We implemented clustering during the requirements specification based on the process/product metric values assigned by domain experts. We used these metrics from the current state of practice at NASA IV & V called Software Integrity Level Assessment Process (SILAP). SILAP considers several factors that affect consequence of failure and error potential of the software components. The list of software components in a project is graded against a set of criteria for these factors and uses weights assigned by domain experts to generate a weighted score for consequence and error potential. We clustered the components of 12 projects using SILAP scores.
- We also implemented clustering early in the software life cycle using the design metrics obtained at the architectural level. We used the reliability and maintainability based risk metrics obtained from previous works [9], [1], [2], [3] to implement clustering.

Brief description of the methodology used in the previous work [9] to obtain the reliability based risk metrics is presented here for the sake of completeness. In [9] Architecture level risk assessment was done in the early phases of software life cycle to obtain reliability based risk metrics such as dynamic complexity, severity and fan out. Unified Modeling Language (UML) [6] and commercial modeling environment Rational

Rose Real Time (RoseRT) [34] were used to get information and data necessary for the estimation of reliability based risk. For each component and connector in the software architecture a heuristic risk factor was obtained. The Markov model was used to obtain the scenario risk factors. The risk factors of use cases were obtained by averaging the scenarios risk factors. Then, the overall system risk factor is obtained by weighting the independent use cases risk factors with the probability of their execution. Furthermore, critical components that would require careful analysis, design and more testing effort were identified.

Brief description of the methodology used in the previous work [1] to obtain the maintainability based risk metrics is presented here for the sake of completeness. In [1] architecture level maintenance risk assessment methodology has been presented for assessing the maintainability based risk into the context of corrective maintenance early in the software life cycle. Corrective software maintenance deals with fixing defects that escape detection before release and that which manifest as field failures [3]. The initial change probabilities for corrective maintenance were obtained by normalizing the frequency of occurrence of each component by the total number of error reports. The maintainability based risk metrics such as change propagation probabilities and size of change were estimated by analyzing the architecture of the system under investigation using structural diagram or class diagram. From these artifacts the components and the connectors of the component based system architecture were identified. The maintenance impact of change in the component was estimated using the size of change metric [1]. This way maintainability based risk metrics of the components could be obtained early in the software life cycle.

In our work we used the maintainability based risk metrics and reliability based risk metrics for implementing clustering on the case study CM1 [33].

- We also implemented clustering on the components of open source software, Indent, based on the code metrics that are available during the operational stage. We clustered the nine components of Indent, using the component entropy and expected visit counts as their software metrics. The expected visit counts represent the expected number of executions of a component. The conditional entropy was used to define the component entropy. We found that components of

Indent that were the most frequently executed and that had maximum number of failed test cases that required a fix were clustered together.

Chapter 3: Background on Methods Used for Clustering

In this chapter we present different clustering techniques that can be used for classification and description of the J48 classifiers that is used to obtain decision trees. Clustering groups a given collection of unlabelled patterns into meaningful clusters.
Pattern clustering activity involves the following steps [12].

- ***Pattern Representation****:* It is a reference to the number of classes, the number of available Patterns, and the number, type, and scale of the features available to the clustering algorithm. The most effective subset of features to be used in clustering are selected from the original features. This process of identifying effective subset of features is called feature selection.
- ***Pattern Proximity***: Pattern proximity is estimated using distance function which is defined on pairs of patterns. For example, the most commonly used similarity measure is the Euclidean distance, in which points have location in space and the distance between points (x1,y1) and (x2,y2) is $dist(x, y) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. Some alternatives are Manhattan distance $dist(x, y) = |x_1 - x_2| + |y_1 - y_2|$, Mahanalobis distance between any two samples x(i) and x(j) is $(x - y)^T \sum^{-1} (x - y)$. Mahanalobis distance takes into account correlation between features and normalizes each feature to zero mean and unit variance [31].
- ***Clustering or grouping****:* It can be done in many ways. Hierarchical or partitional clustering techniques can be used. Hierarchical clustering algorithms produce a series of nested partitions depending on the criterion for merging (agglomerative) or for splitting (divisive) the clusters based on their similarity. Whereas, the partitional algorithms attempt to cluster the set directly, in a manner that depends on a set of parameters. They identify the partition that optimizes a clustering criterion. A partitional clustering algorithm obtains a single partition of data instead of a clustering structure such as a dendrogram produced by hierarchical technique. The k-means is the most commonly used and the simplest algorithm employing a squared error criterion. It starts with a random initial partition and

keeps reassigning the patterns to clusters based on the similarity between the pattern and the cluster centers until a convergence criterion is met.

- ***Abstraction of Data:*** It refers to compact description of each cluster. The representation should be such that it is easy to understand. The output is represented by graphical display, Clustering tree and Banner Plot.
- ***Assessment of output****:* It is done by cluster validity analysis which uses a specific criterion of optimality.

3.1 Transformations on data

Several transformations can be applied on the dataset before applying the dissimilarity measures and implementing clustering [7]. Different normalization techniques and fusion rules could give better results when clustered [13]. Experiments conducted indicated that normalization schemes such as min-max followed by a simple sum of scores fusion yielded better clustering results [13].

Some of the transformations are

Binary metric: In this transformation, non zero values of the features are replaced by one. This is done in order to emphasize the coverage of the program elements rather than the differences in the frequency of the coverage [7].

Proportional metric: In this transformation each attribute is normalized. The range of values for each attribute is computed, and then each value is mapped to its relative position within the range.

Min-Max Normalization: This normalization scheme is best for cases where the bounds (maximum and minimum value) of the data are known. Given a set of values $\{V_K\}$, k=1,2,….n , the normalized score is given by [13]

$$V'_K = \frac{V_K - \min}{\max - \min} \qquad (3.1)$$

The transformed scores can be combined using fusion techniques such as simple sum, maximum value and minimum value [13].

We can use statistical tools like R [32] and S plus to implement hierarchical clustering [18], [28]. Also, Waikato Environment for Knowledge Analysis (WEKA) [24]

[25], a machine learning scheme that enables preprocessing, classifying, clustering, attribute selection and data visualizing can be employed when we want to apply a learning method (classifiers) to the dataset and analyze its output to extract information about that data. WEKA allows us to run the EM clustering and get the j48 classifier.

3.2 Hierarchical Clustering Technique

It was found that hierarchical technique is more appropriate for small sample datasets than the partitional algorithms [35], [16]. Hierarchical cluster analysis has agglomerative methods and divisive methods that find clusters of observations within the dataset.

The divisive method starts with all observations in one cluster and then splits (partition) them into smaller clusters. The agglomerative methods begins by considering each observation as a separate cluster and proceeds to combine until all observations belong to one cluster.

The most commonly used hierarchical clustering methods are [35]

- ***Single Link Method:*** Here, the distance between two clusters is the minimum of the distances between all pairs of clusters drawn from the two clusters.
- ***Complete Link Method:*** Here, the distance between two clusters is the maximum of all pair wise distances between patterns in the two clusters.
- ***Average linkage method*****:** Here, the distance between two clusters is computed as the average distance between objects from the first cluster and objects from the second cluster. The averaging is performed over all pairs (*x*,*y*) of objects, where *x* is an object from the first cluster, *y* is an object from the second cluster.
- ***Wards Method****:* At each step of the cluster process in this method, the two clusters are merged that result in the smallest increase in the with-in cluster sum of squares that is the sum of squared distances between each point and the resultant cluster centroids. It is distinct from the other methods because it uses an analysis of variance approach to evaluate the distances between clusters. It minimizes the sum of squares of any two clusters that can be formed in each step.

All the above mentioned methods display the clustering results graphically by means of a clustering tree or by a banner plot. Clustering tree is a tree in which objects are

represented by the leaves. The vertical coordinate of the place where the two branches join equals the dissimilarity between the corresponding clusters. The Figure 1 shows an example of a clustering tree. If we look for two clusters in Figure 1, then components 1, 6, 4, 11 form one cluster and components 2, 9, 12, 10, 3, 7, 5, 8 form another cluster.

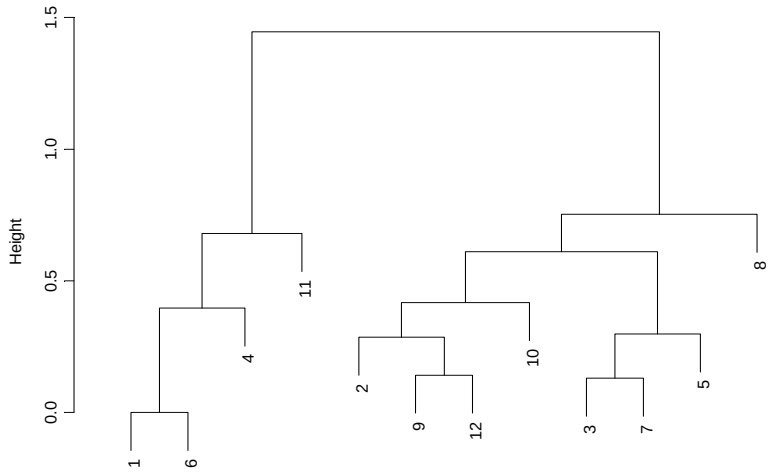

Figure 1. Example of a Clustering tree

The Banner plot representation [19] has a banner that shows the successive mergers from the left to right. It looks like a waving flag. It can be imagined as ragged flag parts at the left and flagstaff at the right. The objects are listed from the top to bottom. The mergers which commence at the between cluster dissimilarity are represented by horizontal bars of correct length. The banner represents the same information as the clustering tree. A banner consists of stars and stripes. The stars refer to linking of the objects and stripes refer to those objects. A banner is always read from left to the right. Each line with stars starts between the clusters being merged. There are fixed scales above and below the banner, ranging from 0.00 (dissimilarity = 0) and 1 (highest dissimilarity is found).It gives a better overall insight into cluster structure and data quality. Figure 2 shows the Banner Plot.

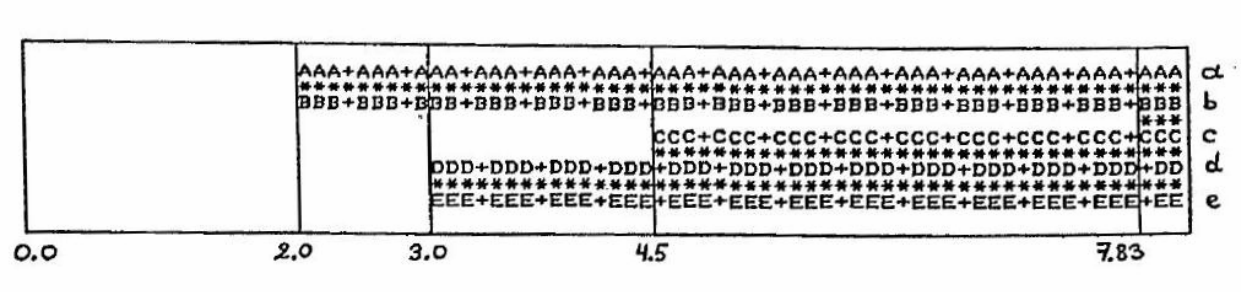

Figure 2. Example of a Banner plot [19]

The banner leads in a natural way to the coefficients describing the strength of the clustering structure found in the dataset (Agglomerative coefficient and Divisive Coefficient). The average width of the banner plot gives an idea of the quality of clustering that is the amount of structure that has been found by the algorithm. If the data has a clear cluster structure, the between cluster dissimilarities (and hence the highest level) will become much larger than the within cluster dissimilarities, so the black lines in the banner become longer. For each object j, the line containing its label is seen and its length is measured on a zero-one scale above or below the banner. The Agglomerative coefficient [18], [19] is thus the average width of (or percentage filled or fraction of blackness in the plot) of the banner plot. It is a dimensionless quantity between zero and one, which does not change when all the original dissimilarities are multiplied by a constant factor, which means that dissimilarities are assumed on a ratio scale. It tells us the strength of the clustering structure that has been obtained. But Agglomerative coefficient (AC) tends to become larger when n increases, so it should not be used to compare datasets of very different sizes.

The Agglomerative Coefficient (AC) is defined for a dataset as

$$AC = \frac{1}{n}\sum l(i) \qquad \text{where } i = 1 \text{ to } n \qquad (3.2)$$

Where for each object *i*, *l*(*i*) is the length of the line containing its label.

When Agglomerative coefficient is small, close to zero, it implies that the clustering algorithm has not found a natural structure, that is no clusters have been found and the data consists of one big cluster. If the value of Agglomerative coefficient is close to one,

it implies that a very clear clustering structure has been found. We use the agglomerative coefficient value to select the clustering method that clusters the data set the best. Thus, when the banner is narrow we find that the agglomerative coefficient is low, indicating that most of the objects remain unlinked for a relatively long time and hence the dataset does not contain very natural clusters which would have been formed sooner.

3.3 Expectation Maximization Clustering

In addition to the hierarchical clustering techniques we also used the Expectation Maximization clustering on the software components for the classification. Expectation Maximization (EM) clustering is a mixture based algorithm [29] that models the distribution of instances probabilistically, so that an instance belongs to a group with a certain probability. EM calculates the densities instead of probabilities. The algorithm is similar to the K-means procedure in that a set of parameters are re-computed until a desired convergence value is achieved. The finite mixtures model assumes all attributes to be independent random variables EM can handle both numeric and nominal attributes. A mixture is a set of N probability distributions where each distribution represents a cluster. An individual instance is assigned a probability that it would have a certain set of attribute values given it was a member of a particular cluster. Suppose $N = 2$, the probability distributes are assumed to be normal and data instances consist of a single real-valued attribute. The algorithm determines the value of five parameters, specifically:

1. The mean and standard deviation for cluster 1
2. The mean and standard deviation for cluster 2
3. The sampling probability P for cluster 1 (the probability for cluster 2 is 1-P)

The general procedure is as follows

1. Initial values for the five parameters mentioned above are guessed.
2. In the case of a single independent variable with mean μ and standard deviation σ, the formula to compute the probability density function is:

$$f(x) = \frac{1}{\left(\sqrt{2\pi\sigma}\right) e^{\frac{-(x-\mu)^2}{2\sigma^2}}} \tag{3.3}$$

In the two-cluster case, we will have the two probability distribution formulas each having differing mean and standard deviation values. The probability density function is used to compute the cluster probability for each instance.

3. The probability scores are used to re-estimate the five parameters.
4. Return to Step 2

The algorithm terminates when a formula that measures cluster quality no longer shows significant increase. This is called as EM algorithm, for expectation maximization. The first step as mentioned above, calculation of the cluster probabilities (Expected class values) is Expectation. The second, that is calculation of distribution parameters, is Maximization of the likelihood of the distributions of the given data [24]. One measure of cluster quality is the likelihood that the data came from the dataset determined by the clustering. The likelihood computation is obtained by the multiplication of the sum of the probabilities for each of the instances.

3.4 Decision Trees obtained using j48 classifier

Decision trees represent a supervised approach to classification. The non terminal nodes represent tests on one or more attributes and terminal nodes reflect the decision outcomes. The WEKA classifier package has its own version of C4.5 knows as J48 classifier [30]. J48 classifier forms rules from pruned partial decision trees built using C4.5's heuristics, which is non-commercial tree building algorithm. The main goal of this scheme is minimization of the number of tree levels and tree nodes and hence maximizes data generalization. It uses a measure taken from the information theory to help with the attribute selection process. Hence, for any choice point in the tree, it selects the attribute that splits the data so as to show the largest amount of information gain. The J48 classifier builds a C4.5 decision tree.

The general approach for a decision tree algorithm is as follows

1. The attribute that best differentiate the output is chosen.
2. A separate tree branch is created for each chosen value.

3. The instances are divided into subgroups so as to reflect the attribute values of the chosen node
4. We terminate the attribute selection process for each subgroup if
   (i) All members for a subgroup have the same value for the output attribute, terminate the attribute selection process for the current path and label the branch on the current path with the specified value.
   (ii)The sub-group has a single node or no further distinguishing attributes can be determined. Branch is labeled with output value seen by the majority of the remaining instances.
5. The above process in repeated for each sub group created in (3) that has not been labeled as terminal.

The above algorithm is applied to a training data. If the test data set is available, the created decision tree is tested. If the test data is not available, the j48 does a cross validation using the training data. If x is the number of folds for cross-validation, then $\frac{(x-1)}{x}$ of the training data is used to construct the model and $\frac{1}{x}$ of the training data is used to test the model. This process is then repeated x times so that all the training data is used exactly once in the test data. The *x* different error estimates are then averaged to yield an overall error estimate [30]. While extensive tests on numerous datasets have shown that ten-fold cross-validation is one of the best numbers for getting the most accurate error estimate, other values can be used. Figure 3 shows an example of a decision tree.

US3 <= 2
| AS2 <= 3: class1 (16.0)
| AS2 > 3: class0 (11.0)
US3 > 2: class2 (6.0)

Figure 3. Example of a decision tree

Each line represents a node in the decision tree. The next line that starts with “ | “ represents the child node of the first line. In general a node with one or more “ | “ characters before the rule is a child node of the node that the right-most line of ' | ' characters terminates at, if it is followed up the page. The next part of the line declares the rule. If the expression is true for a given instance it is classified if the rule is followed by a semi colon and a class designation (that designation becomes the classification of the rule) or, if it isn't followed by a semicolon, we continue to the next node in the tree (the first child node of the node we just evaluated the instance on). If the expression is instead false, we continue to the ``sister'' node of the node we just evaluated; that is, the node that has the same number of '|' characters before it and the same parent node. The classifications are sometimes followed by two numbers in the brackets. The first number tells how many instances in the training set are correctly classified by this node. The second number, if it exists (if not, it is taken to be 0.0), represents the number of instances incorrectly classified by the node.

Chapter 4: Classification of Software Components during the Requirements Specification

We implemented clustering on the software components of 12 real NASA projects using the metrics obtained early in the software life cycle during the requirements specification from the Software Integrity Level Assessment Process (SILAP), the current state of practice in NASA IV & V.

4.1 Software Integrity Level Assessment Process (SILAP)

Software Integrity Level Assessment Process (SILAP) [33], the current state of practice at NASA IV & V is implemented very early in the Software Life Cycle, even before the requirements specification based on several Process/Product metrics and Domain Experts Knowledge. SILAP considers three factors that affect consequence and thirteen factors that affect the error potential of the software components. Some of the Complexity definitions in COCOMO are used in SILAP to define the evaluation criteria "Complexity" early in software life cycle. The list of software components in a project is graded against a set of criteria for different factors related to Consequence and Error Potential. This results in a score for Consequence and Error Potential. The scores are assigned values in a range of 1 to 5. Score 1 is considered a really good score (low Consequence and low Error Potential). Score 5 is considered a bad score (high Consequence and high Error Potential). Using the weights assigned by domain experts to these factors a weighted average of these scores is calculated to generate a score for consequence and Error Potential. These scores are then individually used to select tasks. If the software is a human rated flight, final score for the consequence is obtained taking the human safety into account. An algorithm was used to determine the set of tasks based on the Consequence and the Error Potential scores. SILAP assigns weights to different factors in order to generate a weighted score for consequence and Error Potential. We use clustering and study the behavior of the components based on the SILAP scores.

SILAP considers the factor categories and weights shown in Table 1 and Figure 4 for obtaining a weighted score for Consequence.

Table 1. Factors affecting Consequence (CO1)

Factor Category	Weights
Human Safety(HS2)	0
Asset Safety(AS2)	0.35
Performance(PF2)	0.65

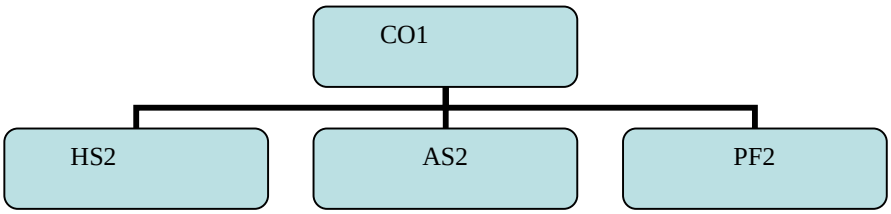

Figure 4. Pictorial representation of the factors affecting Consequence

Figure 4 shows a pictorial representation of the factor categories that affect Consequence. Score for consequence (CO1) is obtained by the weighted average of the Human safety (HS2), Asset Safety (AS2) and Performance (PF2) scores.

SILAP considers the factor categories and weights shown in Table 2 and Figure 5 for calculating weighted score for Error Potential.

Table 2. Factors affecting Error Potential (EP1)

Factor Category	Weights
Development(DV2)	**0.579**
Experience(EX3)	0.828
Development Organization(DO3)	0.172
Process(PR2)	**0.249**
Use of Standards(US3)	0.0955
Use of CM(UC3)	0.0962
CMM Level(CL3)	0.0764
Use Of Formal Reviews(FR3)	0.1119
Use of Defect Tracking System(DT3)	0.0873
Use of Risk Management System(RM3)	0.0647
Re Use Approach(RA3)	0.226
Artifact Maturity(AM3)	0.242
Software Characteristic(SC2)	**0.172**
Complexity(CX3)	0.547
Degree Of Innovation(DI3)	0.351
Size Of System(SS3)	0.102

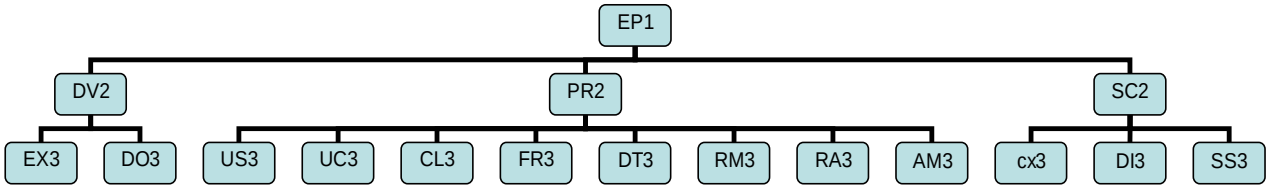

Figure 5. Pictorial representation of the factors affecting Error Potential

There are two types of scores that are considered in the Error Potential calculations.

- ***Direct scores:*** These are the scores that entered by the analysts and have a score from one to five (EX3, DO3, US3, UC3, CL3, FR3, DT3, RM3, RA3, AM3, CX3, DI3, SS3).
- ***Composite Scores:*** These are the scores that are computed based on the weighted average of the direct scores (Development (DV2), Process (PR2), Software Characteristic (SC2)).

The score for Error Potential (EP1) is obtained by the weighted average of the composite scores.

We used the sanitized names for the 12 projects as X1, X2, .., X12. Software components of these projects were assigned direct scores ranging from 1 to 5 by the domain experts and the composite scores were obtained as a weighted sum of the direct scores. We implemented clustering for all the components of the projects based on these SILAP scores. Since the Consequence and Error Potential scores are obtained as a weighted sum of the direct scores, we implement clustering at two levels of granularity on the components of the projects.

The two levels of Granularity at which we implement clustering are

- At a higher level we implemented clustering on the components of each project using the weighted average scores of Consequence (CO1) and Error Potential (EP1) scores.
- At a lower level we implemented clustering on the components of the project using the direct score attributes which were assigned by the analysts that is the factors in the leaves of the tree representation in Figure 4 and Figure 5.

4.2 Implementation of Agglomerative clustering on projects

We implemented clustering on the SILAP scores of the 12 projects X1, X2, …, X12. Due to space limitation we present the results obtained for two projects X9 and X10 here.

4.2.1 Clustering results based on weighted Consequence and Error Potential scores

Project X9

We implemented clustering using the hierarchical Wards method for each project with the weighted consequence and error potential scores. The clustering tree that we obtained for project X9 is as shown in Figure 6.

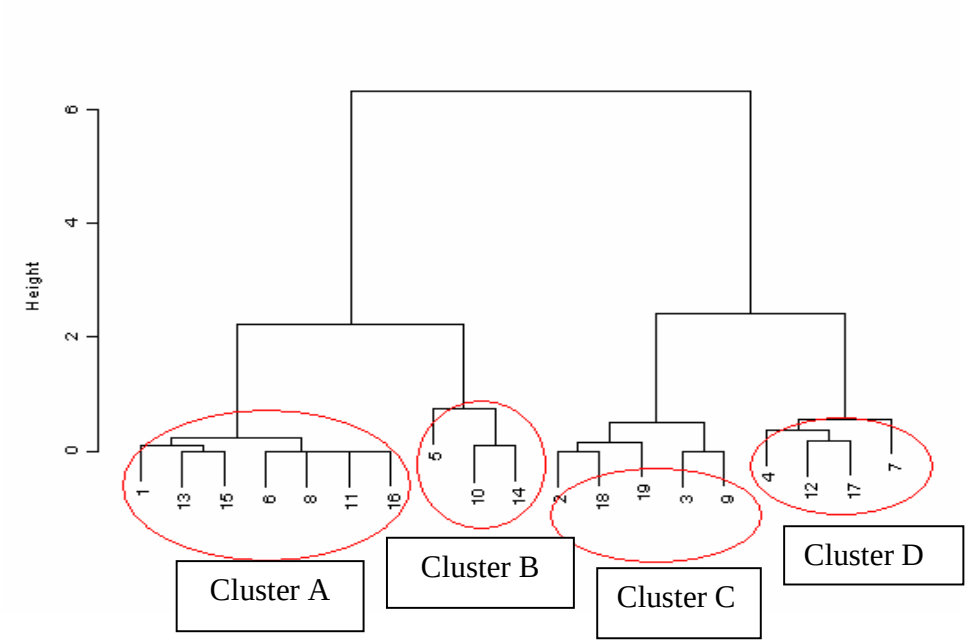

Figure 6. Clustering tree of project X9 obtained using the Consequence and Error Potential scores (Agglomerative Coefficient: 0.979)

In hierarchical clustering we can decide the number of clusters, by analyzing the output. From Figure 6 it is evident that the 19 components of project X9 form 4 distinct clusters, A, B ,C and D. The Agglomerative coefficient value obtained was 0.979, which indicates a good quality of clustering.

Table 3 shows the four distinct clusters, the components in each cluster and their CO1 and EP1 values.

Table 3. Clusters of the components of project X9 based on Consequence and Error Potential scores

Cluster and components		Consequence	Error Potential
Cluster	**Components**	**CO1**	**EP1**
A	1	2.30	2.49
	13	2.30	2.58
	15	2.30	2.58
	6	2.30	2.68
	8	2.30	2.68
	11	2.30	2.68
	16	2.30	2.68
B	5	1.65	2.58
	10	1.00	2.58
	14	1.00	2.49
C	2	3.60	2.58
	18	3.60	2.58
	19	3.70	2.68
	3	3.35	2.77
	9	3.35	2.77
D	4	4.35	2.68
	12	4.65	2.68
	17	4.65	2.49
	7	5.00	2.68

From the Table 3, it is evident that the components that have similar characteristics are in one cluster. For instance, components in cluster A have moderate consequence and moderate error potential scores, while components of cluster B have low consequence and moderate error potential scores, components of cluster C have relatively high consequence and moderate error potential scores and components of cluster D have very high consequence and moderate error potential scores. Here, in this project as the error potential scores are close to each other clustering were being guided by consequence scores.

Project X10

The clustering tree obtained for project X10 using the weighted score of CO1 and EP1 is shown in Figure 7. From Figure 7, we see that there are 3 distinct clusters. The agglomerative coefficient value obtained was 0.904, which indicates a good quality of clustering.

Table 4 shows the clusters of project X10, its components and CO1 and EP1 values of project X10.

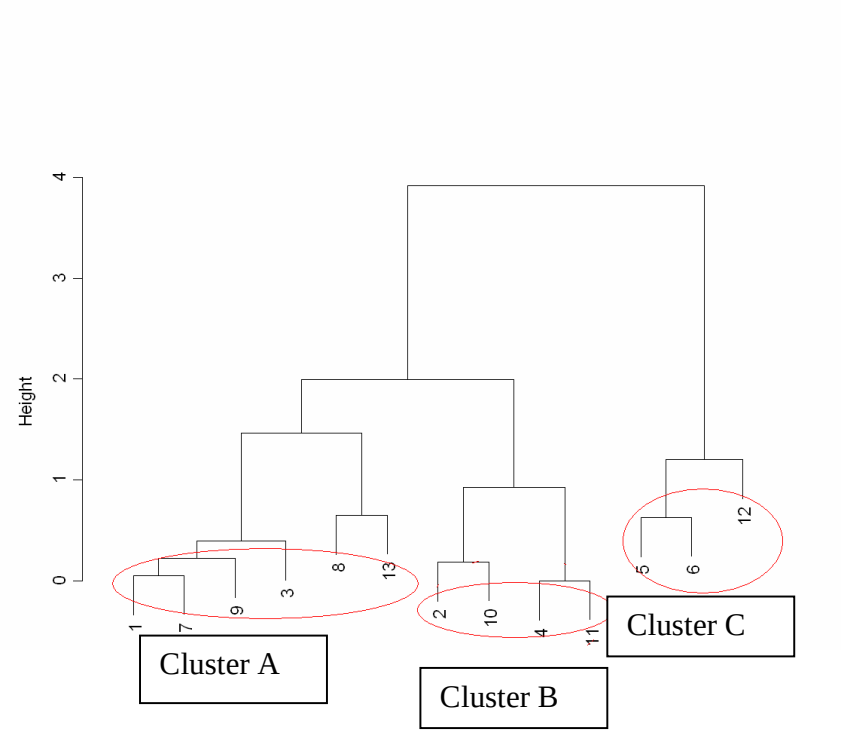

Figure 7. Clustering tree of project X10 obtained using the Consequence and Error Potential scores (Agglomerative coefficient – 0.904).

Table 4. Clusters of components of project X10 based on Consequence and Error Potential scores

Clusters and components		**Consequence**	**Error Potential**
Cluster	**Components**	**CO1**	**EP1**
A	1	2.3	1.74
	3	2.0	1.75
	7	2.35	1.74
	8	2.30	2.64
	9	2.30	1.93
	13	1.65	2.64
B	2	1.65	1.74
	10	1.65	1.55
	4	1.00	1.74
	11	1.00	1.74
C	5	3.0	1.74
	6	3.6	1.55
	12	4.3	1.93

From Table 4, it is evident that the components that have similar characteristics are clustered together. Cluster A has moderate consequence and error potential values. Cluster B has relatively low consequence and moderate error potential. Cluster C has relatively high consequence and moderate error potential.

Similarly we implemented clustering on the other 11 projects based of the CO1 and EP1 scores and used the Agglomerative Coefficient values for their validation.

4.2.2 Clustering results based on direct scores

We implemented clustering using the Hierarchical Wards clustering method on the 12 projects with the 3 direct scores HS2 , AS2 and PF2 that affect Consequence and the 13 direct scores EX3, DO3, US3, UC3, CL3, FR3, DT3, RM3, RA3, AM3, CX3, DI3, SS3 that affect the Error Potential of the components for each project.

Project X9

The clustering tree obtained after implementing clustering on the X9 project is as shown in Figure 8. As seen in the Figure 8, clustering tree there are 3 distinct clusters. The Agglomerative coefficient value obtained was 0.938, which indicates a good quality of clustering. Table 5 shows the clusters, components in each cluster and their direct score values

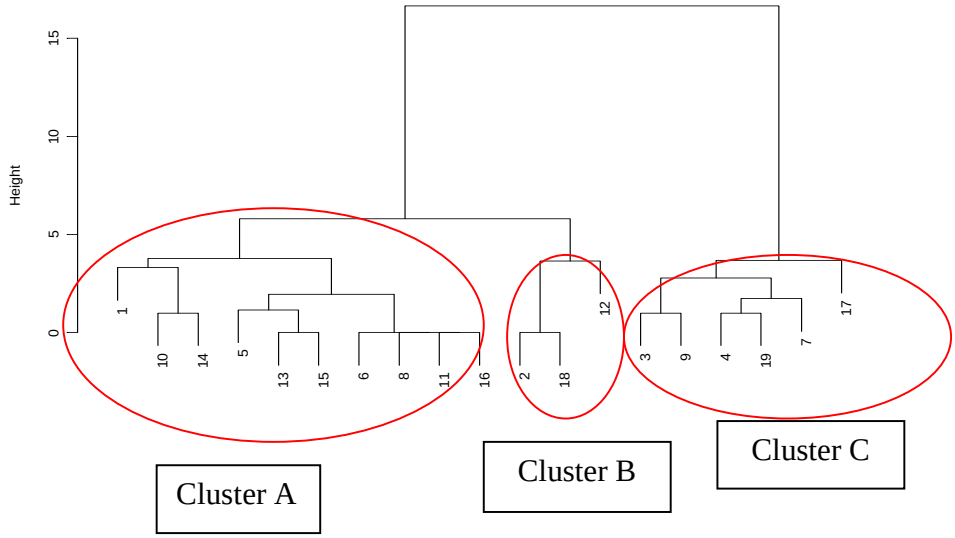

Figure 8. Clustering tree of project X9 obtained using the direct scores (Agglomerative Coefficient -0.938)

Table 5. Clusters of components of project X9 based on the direct scores

Cluster and components		Consequence			Error potential												
Cluster	**Components**	**HS2**	**AS2**	**PF2**	**EX3**	**DO3**	**US3**	**UC3**	**CL3**	**FR3**	**DT3**	**RM3**	**RA3**	**AM3**	**CX3**	**DI3**	**SS3**
A	1	2	1	3	3	2	3	1	3	2	2	2	5	1	1	1	4
	10	0	1	1	3	2	3	1	3	2	2	2	5	1	2	1	4
	14	0	1	1	3	2	3	1	3	2	2	2	5	1	1	1	4
	5	0	1	2	3	2	3	1	3	2	2	2	5	1	2	1	4
	13	0	1	3	3	2	3	1	3	2	2	2	5	1	2	1	4
	15	0	1	3	3	2	3	1	3	2	2	2	5	1	2	1	4
	6	0	1	3	3	2	3	1	3	2	2	2	5	1	3	1	4
	8	0	1	3	3	2	3	1	3	2	2	2	5	1	3	1	4
	11	0	1	3	3	2	3	1	3	2	2	2	5	1	3	1	4
	16	0	1	3	3	2	3	1	3	2	2	2	5	1	3	1	4
B	2	0	1	5	3	2	3	1	3	2	2	2	5	1	2	1	4
	18	0	1	5	3	2	3	1	3	2	2	2	5	1	2	1	4
	12	0	4	5	3	2	3	1	3	2	2	2	5	1	3	1	4
C	3	5	4	3	3	2	3	1	3	2	2	2	5	1	4	1	4
	9	4	4	3	3	2	3	1	3	2	2	2	5	1	4	1	4
	4	5	5	4	3	2	3	1	3	2	2	2	5	1	3	1	4
	19	5	5	3	3	2	3	1	3	2	2	2	5	1	3	1	4
	7	5	5	5	3	2	3	1	3	2	2	2	5	1	3	1	4
	17	5	4	5	3	2	3	1	3	2	2	2	5	1	1	1	4

Comparing Table 3 and Table 5, we observe that components of project X9 were grouped in a different way. When clustering was based on weighted average scores CO1 and EP1 than they were clustered in a different when compared to based on the direct score attributes. This difference in clustering could be due to loss of information because of weighting as the scores for CO1 and EP1 are the weighted average scores.

Project X10

The clustering tree obtained after implementing clustering on project X10 is shown in Figure 9. As seen clustering tree has three distinct clusters, A, B, C. The Agglomerative Coefficient value 0.704 indicates a good quality of clustering. Table 6 shows the clusters, components in each cluster and the direct scores for project X10.

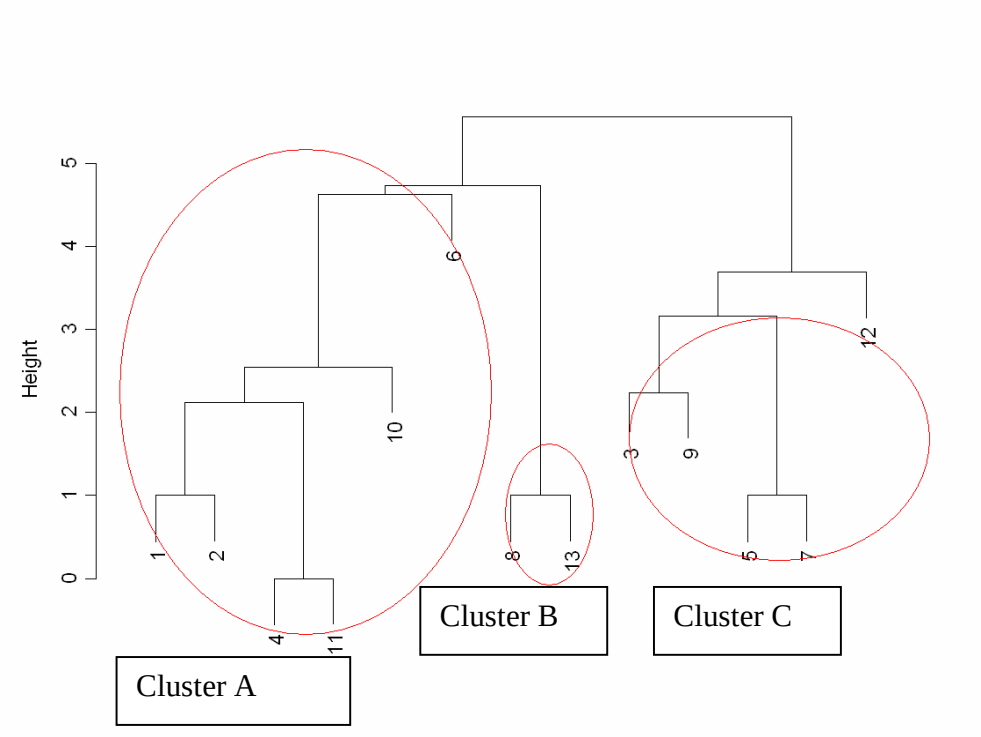

Figure 9. Clustering tree of project X10 obtained after using the direct scores (Agglomerative coefficient – 0.704)

.

Table 6. Clusters of components of project X10 based on the direct scores

Cluster and components		Consequence			Error Potential												
Cluster	**Components**	**HS2**	**AS2**	**PF2**	**EX3**	**DO3**	**US3**	**UC3**	**CL3**	**FR3**	**DT3**	**RM3**	**RA3**	**AM3**	**CX3**	**DI3**	**SS3**
A	1	0	1	3	1	4	1	1	3	1	1	1	2	3	3	1	3
	2	0	1	2	1	4	1	1	3	1	1	1	2	3	3	1	3
	4	0	1	1	1	4	1	1	3	1	1	1	2	3	3	1	3
	11	0	1	1	1	4	1	1	3	1	1	1	2	3	3	1	3
	10	0	1	2	1	4	1	1	3	1	1	1	2	3	1	1	3
	6	0	1	5	1	4	1	1	3	1	1	1	2	3	1	1	3
B	8	0	1	3	3	4	1	1	3	1	2	1	1	3	3	1	2
	13	0	1	2	3	4	1	1	3	1	2	1	1	3	3	1	2
C	3	0	2	2	1	4	1	1	3	1	1	1	2	2	4	1	2
	9	0	1	3	1	4	1	1	3	1	1	1	2	3	5	1	3
	5	0	3	3	1	4	1	1	3	1	1	1	2	3	3	1	3
	7	0	3	5	1	4	1	1	3	1	1	1	2	3	5	1	3
	12																

Comparing Table 4 with Table 6 we find that components of project X10 were grouped in a different way. This difference in the clustering could be due to the loss of information due to weighting as the scores for CO1 and EP1 are the weighted average scores.

Similarly we implemented clustering on all the other 11 projects using the direct scores as attributes. For all the projects, when the clustering tree for the components of each project obtained with the weighted average scores CO1 and EP1 and the clustering tree obtained with the direct scores was compared, they were different. Hence we could infer that weighting causes loss of information and the behavior of the components could be understood better using the direct scores, as there would be no loss of information. We need domain expert's knowledge to further validate our results.

4.3 Implementation of Expected Maximization Clustering and obtaining decision trees

In addition to the hierarchical clustering technique we also implemented EM clustering on the 12 projects, using the weighted average and the direct SILAP scores.

4.3.1 Clustering results based on weighted Consequence and Error Potential scores

We implemented EM clustering on the 12 projects with the CO1 and EP1 scores. EM clustered the components of the project X9 as shown in Table 7.

Table 7. Cluster of components of project X9 using Expected Maximization Clustering based on the Consequence and Error Potential scores.

Cluster	Components
A (class0)	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

As seen from Table 7, all the 19 components of project X9 were classified as a single class (class0), which implies that all the components behave similarly, when clustered based on CO1 and EP1 scores.

Similarly we ran the WEKA tool on all the other 11 projects using the weighted average scores CO1 and EP1 and obtained the clusters. All the results obtained were different from the results obtained in hierarchical clustering.

4.3.2 Clustering results based on direct score attributes

We implemented EM clustering on the components of the project using the direct score attributes, that is HS2, AS2, PF2, EX3, DO3, US3, UC3, CL3, FR3, DT3, RM3, RA3, AM3, CX3, DI3, SS3. The clusters that we obtained after we ran the WEKA tool on the X9 project is shown in Table 8.

Table 8. Cluster of components of project X9 using Expected Maximization Clustering based on the direct scores.

Cluster	Components
A (class0)	1 2 5 6 8 10 11 12 13 14 15 16 18
B (class1)	3 4 7 9 17 19

As seen from Table 8, when we implemented the EM cluster using the direct scores HS2, AS2, PF2, EX3, DO3, US3, UC3, CL3, FR3, DT3, RM3, RA3, AM3, CX3, DI3, SS3 of the components for the project X9, components were classified into two clusters (class0 and class1). We see that components 1,2,5,6,8,10,11,12,13,14,15,16,18 are clustered as one cluster (class0) and components 3,4,7,9,17,19 are clustered as another cluster (Class1) as shown in Table 8.

Similarly we implemented EM Clustering on the other 11 projects using the direct score factors and obtained the clusters. They clustered the components in a different way when compared to the clusters obtained using the weighted score factors (CO1 and EP1). All the results obtained using EM Clustering were different from the results obtaimed using hierarchical clustering.

4.3.3 Implementation of J48 classifier to obtain decision trees

We ran the J48 classifier using WEKA and obtained the J48 pruned decision tree for the components of all the 12 projects that were clustered (EM cluster) using the direct score. The J48 decision tree tells us which attribute causes the components to behave and hence cluster differently. We obtained J48 pruned decision tree when we ran the WEKA tool using J48 classifier for all the projects. The J48 decision tree obtained for project X9 is as shown in Figure 10.

HS2 <= 2: class0 (13.0)

HS2 > 2: class1 (6.0)

Figure 10. J48 decision tree for project X9

This implies that the factor HS2 is the factor that has the highest information gain and decides the cluster. If the HS2 score is less than or equal to 2 then it's classified as a cluster, class0, otherwise it's classified as another cluster (class1). There are 13 instances correctly classified as class0 and 6 instances correctly classified as class1.

Similarly we ran the j48 classifier to obtain the decision trees for all the other 11 projects based on the clusters that were obtained when they were clustered using direct score factors (EM Clustering). Table 9 shows the decision trees that were obtained for all the 12 projects.

Table 9. Decision trees for the 12 projects[1]

Project	10 Fold Cross Validation			Decision Trees
X1	Correctly Classified Instances	43	97.7273 %	EX3 <= 2 | PF2 <= 2: class3 (8.0) | PF2 > 2: class1 (17.0) EX3 > 2: class2 (19.0/1.0)
	Incorrectly Classified Instances	1	2.2727 %	
X2	Correctly Classified Instances	42	100 %	DO3 <= 4: class0 (15.0)

[1] Considering only the components of the project that have values for all the factors affecting consequence and error potential.

	Incorrectly Classified Instances 0 0 %	DO3 > 4 | UC3 <= 1: class1 (15.0) | UC3 > 1: class2 (12.0)
X3	Correctly Classified Instances 20 83.3333 % Incorrectly Classified Instances 4 6.6667 %	DO3 <= 4 | DI3 <= 2: class2 (19.0) | DI3 > 2: class1 (2.0) DO3 > 4: class0 (3.0)
X4	Correctly Classified Instances 9 81.8182 % Incorrectly Classified Instances 2 18.1818 %	EX3 <= 2: class1 (3.0/1.0) EX3 > 2: class0 (8.0)
X5	Correctly Classified Instances 11 73.3333 % Incorrectly Classified Instances 4 26.6667 %	US3 <= 2 | EX3 <= 1: class3 (2.0) | EX3 > 1 | | PF2 <= 3: class1 (7.0) | | PF2 > 3: class2 (4.0) US3 > 2: class0 (2.0)
X6	Correctly Classified Instances 25 100 % Incorrectly Classified Instances 0 0 %	DO3 <= 1: class1 (15.0) DO3 > 1: class2 (10.0)
X7	Correctly Classified Instances 137 99.2754 % Incorrectly Classified Instances 1 0.7246 %	DO3 <= 2 | HS2 <= 1: class2 (77.0) | HS2 > 1 | | EX3 <= 1: class1 (10.0) | | EX3 > 1: class0 (6.0) DO3 > 2: class1 (45.0/1.0)
X8	Correctly Classified Instances 24 92.3077 % Incorrectly Classified Instances 2 7.6923 %	SS3 <= 1 | US3 <= 1: class5 (3.0) | US3 > 1: class4 (7.0) SS3 > 1 | EX3 <= 2: class1 (5.0) | EX3 > 2: class2 (11.0/1.0)
X9	Correctly Classified Instances 17 89.4737 % Incorrectly Classified Instances 2 10.5263 %	HS2 <= 2: class0 (13.0) HS2 > 2: class1 (6.0)
X10	Correctly Classified Instances 10 76.9231 % Incorrectly Classified Instances 3 23.0769 %	EX3 <= 1 | AS2 <= 1: class0 (7.0)

		| AS2 > 1: class1 (4.0/1.0) EX3 > 1: class2 (2.0)
X11	Correctly Classified Instances 33 100 % Incorrectly Classified Instances 0 0 %	US3 <= 2 | AS2 <= 3: class1 (16.0) | AS2 > 3: class0 (11.0) US3 > 2: class2 (6.0)
X12	Correctly Classified Instances 37 86.0465 % Incorrectly Classified Instances 6 13.9535 %	UC3 <= 1 | EX3 <= 1 | | DO3 <= 3: class3 (7.0) | | DO3 > 3: class4 (2.0) | EX3 > 1: class5 (29.0/3.0) UC3 > 1: class0 (5.0)

From Table 9, it is evident that each project is different and different attributes have highest information gain for each project.

J48 Decision tree obtained for all the twelve projects

We implemented the EM clustering on all twelve projects together using the direct score factors and then we ran the j48 classifier using WEKA tool to obtain the decision tree. The decision tree obtained for all the twelve projects together is as shown in Figure 11.

DT3 <= 1
| RM3 <= 2: class1 (14.0)
| RM3 > 2
| | EX3 <= 3
| | | HS2 <= 1: class6 (68.0)
| | | HS2 > 1: class2 (24.0/1.0)
| | EX3 > 3: class7 (45.0)
DT3 > 1
| US3 <= 3
| | CL3 <= 3
| | | AS2 <= 4
| | | | AM3 <= 1

| | | | | RM3 <= 1
| | | | | | UC3 <= 1: class1 (11.0)
| | | | | | UC3 > 1
| | | | | | | US3 <= 2
| | | | | | | | DO3 <= 3: class0 (2.0/1.0)
| | | | | | | | DO3 > 3: class4 (8.0)
| | | | | | | US3 > 2
| | | | | | | | HS2 <= 1
| | | | | | | | | CX3 <= 3: class0 (13.0/1.0)
| | | | | | | | | CX3 > 3: class3 (3.0/1.0)
| | | | | | | | HS2 > 1: class3 (2.0)
| | | | | RM3 > 1
| | | | | | HS2 <= 0: class0 (89.0/3.0)
| | | | | | HS2 > 0
| | | | | | | AS2 <= 2
| | | | | | | | DI3 <= 1: class0 (6.0/1.0)
| | | | | | | | DI3 > 1: class5 (2.0)
| | | | | | | AS2 > 2: class3 (3.0)
| | | | AM3 > 1
| | | | | FR3 <= 1: class1 (2.0)
| | | | | FR3 > 1
| | | | | | AM3 <= 2
| | | | | | | PF2 <= 1: class5 (2.0)
| | | | | | | PF2 > 1: class3 (10.0/1.0)
| | | | | | AM3 > 2: class5 (6.0)
| | | AS2 > 4: class3 (19.0)
| | CL3 > 3
| | | UC3 <= 1
| | | | RA3 <= 3: class5 (3.0/1.0)
| | | | RA3 > 3: class1 (28.0/1.0)
| | | UC3 > 1
| | | | FR3 <= 1| | | | | DO3 <= 4: class4 (21.0)
| | | | | DO3 > 4: class5 (3.0)
| | | | FR3 > 1: class5 (30.0)

| US3 > 3: class8 (19.0)

Figure 11. J48 decision tree for all the twelve projects together

Observing the decision trees obtained in Table 9 and Figure 11, we find that different attributes have highest information gain in different projects. Another observation from Figure 11 is that, the attribute Use of Defect Tracking System (DT3) has highest information gain for all the twelve projects together. This way implementing Clustering and running the classifiers could contribute to the classification of the software components and revealing the attribute that has the highest information gain in the decision trees very early in the software life cycle.

4.4 Proposed Algorithm for ranking clusters in a Project

We propose a way to rank the clusters based on Consequence and Error Potential. We ordered the direct scores that affect consequence and error potential based on the weights assigned by domain experts as shown in Figure 12.

Consequence	Error Potential
HS2 (0.0)	EX3 (0.828)
PF2 (0.65)	CX3 (0.547)
AS2 (0.35)	DI3 (0.351)
	AM3 (0.242)
	RA3 (0.226)
	DO3 (0.172)
	FR3 (0.1119)
	SS3 (0.102)
	UC3 (0.0962)
	US3 (0.0955)
	DT3 (0.0873)
	CL3 (0.0764)
	RM3 (0.0647)

Figure 12 Order of importance for Consequence and Error Potential

Algorithm for Ranking n clusters based on Consequence

Let Co_i represent the factors that affect consequence.

1. *Select the Co_i that has the highest order of importance.*
2. *Let $Comax_{i-j}$ be the maximum value of selected Co_i for cluster j.*
3. *Sort the clusters based on their $Comax_{i-j}$ values for the selected Coi in descending order and store in array sort[].*
4. *If (Comax $_{i-m}$ of sort[k]) > (Comax $_{i-h}$ of sort[k+1])*

   Then rank cluster m higher rank

   Repeat step 4 for next value of k in sort[] having the same Co_i checked.

   Else

   {

   - Check the sequence of next values in sort[] till the $Comax_{i-j}$ values of index where sort[index] !=sort[k] for same Co_i

   Repeat

   {

   If (! all Co_i's are checked for the sequence of sort[k] to sort[index-1])

   {

   - Select the next Co_i according to the order of importance for these sequence of clusters in sort[k] to sort[index-1]

   - Resort sort[] only for these values from sort[k] to sort[index-1] in descending order based on the $Comax_i$ values of selected Co_i of these clusters.

   - Repeat step 4 for these sequence of clusters from sort[k] to sort[index-1]

   }

Else

{

Sequence of clusters from sort[k] to sort[index-1] are ranked the same.

}

} Until all clusters in sort[k] to sort[index-1] are assigned a rank

}

Ranking based on Error Potential: is done similarly, except that we consider the ordering of the direct scores that affect error potential

4.4.1 Ranking for Project X9

Ranking of clusters of project X9 based on Consequence

As shown in Table 10, the maximum HS2 score of cluster B is higher than the maximum HS2 score of cluster A (5>2), cluster B is ranked higher than cluster A based on consequence.

Table 10 Ranking of the Clusters of project X9 based on Consequence

Cluster	Components	Maximum HS2	Rank
A	1 2 5 6 8 10 11 12 13 14 15 16 18	2.0	II
B	3 4 7 9 17 19	5.0	I

Ranking of clusters of project X9 based on their Error Potential

As seen in the Table 11

Maximum EX3 value of cluster A = maximum EX3 value of cluster B

Maximum CX3 value of cluster B > maximum CX3 value of cluster A

Hence, Cluster B is ranked higher based on error potential

Table 11 Ranking of clusters of project X9 based on error potential

Cluster	Components	EX3	CX3	Rank
A (class0)	1 2 5 6 8 10 11 12 13 14 15 16 18	3.0	3.0	II
B (class1)	3 4 7 9 17 19	3.0	4.0	I

Similarly, we ranked the clusters for all the other projects. This way we could rank the clusters of components very early in the software life cycle with clustering.

Chapter 5: Classification of Software Components Based on Reliability and Maintainability Based Risk in the Architectural Level

We implemented clustering using metrics obtained early in the software life cycle, when the requirement specifications and design details are available. In this chapter, we present the clustering results obtained on the CM1 case study using the reliability based risk metrics such as Complexity, Severity and Fan-Out [9] and the maintainability based risk metrics such as Change Probabilities, Normalized Maintenance Impact Fan-Out and Normalized maintenance Impact Fan-In [1], [2], [3] .

5.1 CM1 Case Study

CM1 is a software component of a data processing unit used in an instrument which exploits data, to probe the early universe. This case study is from the Data Metrics Program [33]. CM1 has 12 components and 9 scenarios [4]. Reliability based risk metrics and Maintainability based metrics were obtained for CM1 using methodology presented in [1], [4], [9] early in the software life cycle. We implemented clustering on this case study using these metrics and studied the behavior of the components of CM1 early in the software life cycle.

5.2 Reliability-based Risk Metrics

Reliability based risk is defined as an unexpected result originated from a wrong system behavior, which is out of the feasible space defined from the functional requirements. In this case the source of failure is a violation of some functional requirement. It takes into account that the probability that a software product will fail in the operational environment and the adversity of the failure. In [4], [9] a methodology for assessing reliability based risk in early phases of a software cycle was developed.

Description of the methodology used in the previous works [4], [9] and the definitions of the reliability based metrics has been explained for the sake of completeness. Information necessary for estimation of reliability based risk was obtained using unified modeling language (UML) [6] and the commercial modeling environment Rational Rose RT (RoseRT) [34]. From the UML diagrams for each component and

connector in the software architecture a heuristic risk factor was obtained. Scenario risk factors were computed using Markov model. Risk factors for usecases were obtained by averaging the scenarios risk factors. In order to obtain the overall system risk factor the independent use cases risk factors were weighted with the probability of their execution. Reliability based risk metrics such as Dynamic Complexity, Severity and Fan-Out were obtained using as dynamic specification metrics from UML

- ***Dynamic Complexity***

  As there is a correlation between the number of faults found in a software component and its complexity, in [4] the dynamic complexity of state charts was obtained as a dynamic metric for components. Dynamic coupling between components was computed as a dynamic metric related to fault proneness for connectors. Component's Dynamic Complexity was obtained based on the UML state charts that are available during the early stages of software life cycle [4]. A number of states and transition between these states in the state chart of each component i describes the dynamic behavior of the component.

  Dynamic Complexity is defined as follows [4]

- For a scenario S_x, a subset of all states of component i are visited in the scenario and a subset of all the transitions are traversed. If C_i^x denotes the subset of states for a component *i* visited in the scenario S_x and with T_i^x as the subset of transitions traversed in the state chart of component *i* in that scenario. The subset of states C_i^x and the corresponding transitions T_i^x were mapped into a control graph. $c_i^x = \left|C_i^x\right|$ and $t_i^x = \left|T_i^x\right|$ denotes the number of nodes in that graph (cardinality of C_i^x) and number of edges in that graph (cardinality of T_i^x) respectively. Dynamic Complexity doc_i^x of component *i* in scenario S_x is defined as [4]

  $$doc_i^x = t_i^x - c_i^x + 2. \qquad (5.1)$$

Normalizing the Dynamic Complexity doc_i^x with respect to the sum of complexities for all active components gives Normalized Dynamic Complexity DOC_i^x of a component *i* in a scenario S_x [4]. Using this definition, Dynamic Complexity metrics were obtained for the components of CM1 [4].

- ***Severity metric:*** Severity metric measures the severity of the consequences of potential failures [4]. To get this metric value domain experts play a major role. Based on hazard analysis [23] [4], the severity classes were identified as follows:
  Catastrophic: A failure that could cause death or total system loss
  Critical: A failure that could cause severe injury, major property damage, major system damage, or major loss of production.
  Marginal: A failure that could cause minor injury, minor property damage, minor system damage, or delay or minor loss of production.
  Minor: A failure that is not serious enough to cause injury, property damage, or system damage, but could result in unscheduled maintenance or repair.
  In [4] severity indices of 0.25, 0.50, 0.75 and 0.95 were assigned to minor, marginal, critical and catastrophic severity classes respectively.
  Experts make an estimate of the severity of the components and connectors based on their experience with other systems in the same field. The components of CM1 were assigned a score based on their severity class [4].
- ***Fan*-Out:** The Fan-out metric value was obtained from the UML diagrams for each component [4] early in the software life cycle.

We implemented clustering on the components of CM1 using the Reliability based metrics , Dynamic Complexity, Severity and Fan-Out scores that were assigned to the components of CM1 according to the definitions defined above[4].

5.2.1 CM1 Case Study Results based on Reliability-based-Risk

We clustered the components of CM1 based on reliability based metrics per scenario and also fusion of all scenarios (Simple sum, Weighted Sum, Worst Case analysis) [17] [13]. We implemented hierarchical clustering methods (Single Link, Complete Link, Average

Link, Wards method) on the case study and found that Wards method had the highest agglomerative coefficient values than the other methods.

- ***Per-Scenario Results for CM1***

  We implemented hierarchical Wards method as it had the highest agglomerative coefficient value, using the Euclidean dissimilarity measure on each of the 9 scenarios [4] with the reliability based metrics such as Dynamic Complexity, Severity and Fan-Out for all components. The results for the scenario HouseKeeping (HK) using Wards method are as shown in Figure 13.

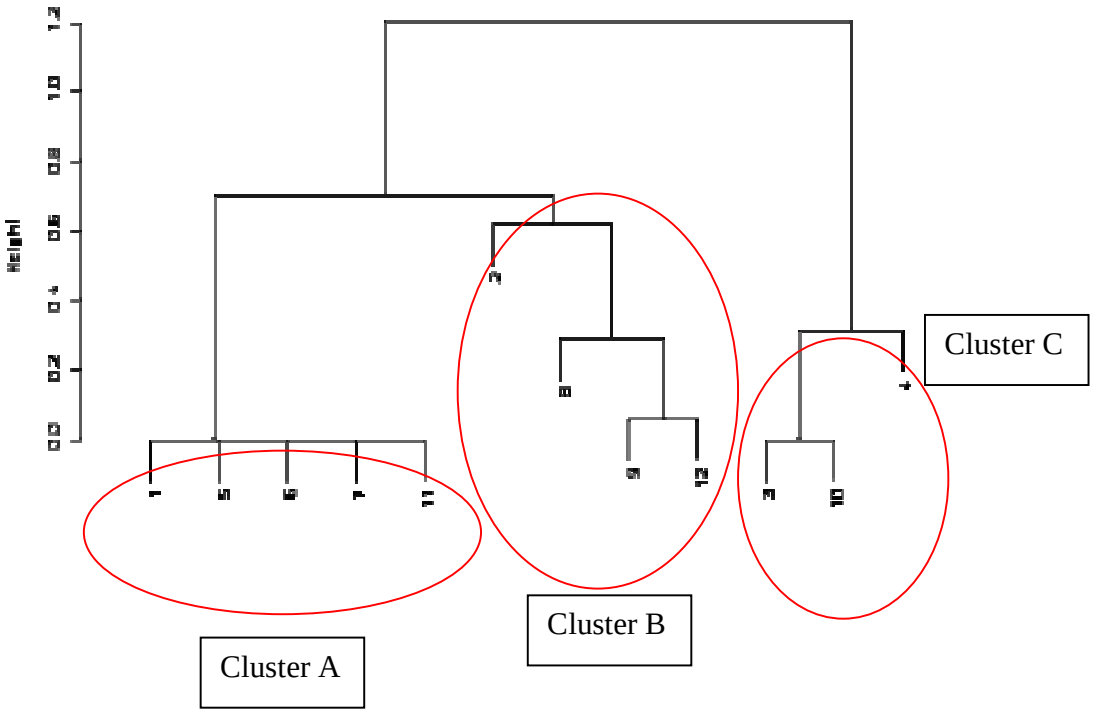

Figure 13. Clustering tree of CM1 obtained for the HouseKeeping (HK) scenario based on Reliability based risk metrics (Agglomerative Coefficient AC 0.90)

As shown in Figure 13, if we look for 3 clusters then the components 1, 5, 6, 7, 11 are the most similar to each other forming a cluster (cluster A) , components 2, 8 ,9, 12 form the second (Cluster B) and components 3, 10 and 4 form the third cluster(Cluster C) for the scenario HouseKeeping(HK) [4]. The AC value was the highest when clustering was implemented using the hierarchical clustering Wards method for all the 9 scenarios. This

indicates that the quality of clustering structure found using the Wards method was better than any other method for our dataset. For the HouseKeeping(HK) scenario explained above the Agglomerative coefficient(AC) value was 0.90 with the Wards method.

Similarly, Clustering was implemented for the other 8 scenarios of CM1. Wards method performed well for all the scenarios. Much inference could not been drawn about the behavior of the components, as Clustering tree obtained for each scenario was different., indicating that the components behaved differently in different scenarios.

We then experimented ways to combine the reliability based metrics of components across all the nine scenarios and implement clustering in order to get better interpretation of results.

- ***Fusion of Reliability based risk metrics across all scenarios***

We implemented clustering on the CM1 components using techniques like the simple sum scores fusion, weighted sum scores fusion and worst case value (Maximum value) [17] [13] across all the scenarios.

- We obtained clustering results using the hierarchical Wards method as it had the highest AC value, with the simple sum scores fusion that is clustering the components using the simple sum of the metrics across all the 9 scenarios. Also implemented the weighted sum scores fusion, that is clustering components using the weighted sum of metrics across all scenarios (weighted by the probability of occurrence of each scenario) [13] [17].

For Risk, the worst case values are considered to be the most important. We looked at the metric values in all the scenarios for each component and selected the worst case values (i.e. Maximum value) [13]. We implemented hierarchical Wards clustering method as it had the highest Agglomerative Coefficient value. The clustering tree obtained for clustering using the worst case values are as shown in Figure 14. Figure 15 shows its corresponding Banner plot. From the clustering tree in figure 14, it is evident that components 1, 6, 4, 11 form one cluster, components 2, 9, 12, 10, 3, 7, 5 form the second and component 8 forms the third. Agglomerative Coefficient value when Wards clustering was implemented using the worst case values was 0.81. This indicated good quality of clustering. From Table 12, we observe that most of the components in cluster A, that is components 1, 6, 4, 11 have relatively low Fan-Out, relatively low Complexity

and moderate severity values. Most of the components in the cluster B, that is components 2, 9, 13, 10, 3, 7, 5 have moderate Complexity, moderate Fan-Out and relatively high Severity values. Cluster C, that is components 8 has relatively higher Complexity, higher Severity and higher Fan-Out values than most of the other components. Domain experts ranked the components of CM1 based on their knowledge and experience, starting from the most critical to least critical as 8, 3, 10, 7, 12, 9, 2, 5, 11, 6, 1, 4 [33]. The clustering results we obtained in Figure 14 were in accordance with the ranking given by the domain experts.

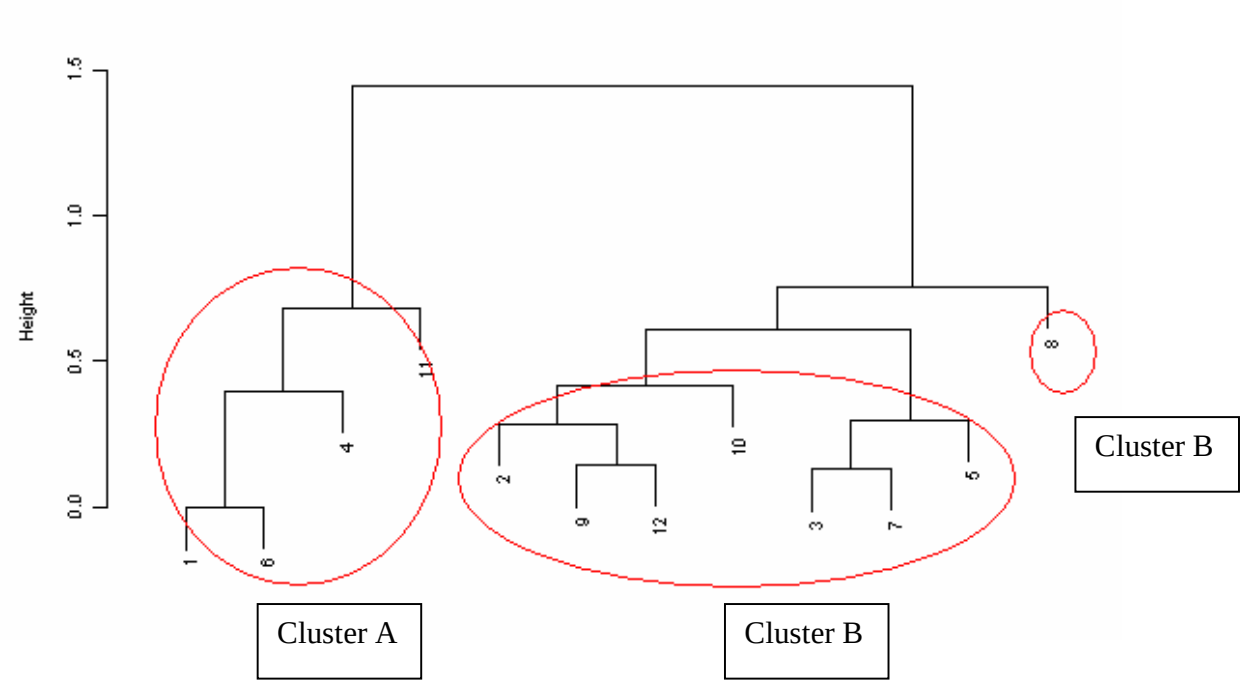

Figure 14. Clustering Tree of CM1 obtained for the worst case values of Reliability Based Risk metrics (Agglomerative coefficient 0.81)

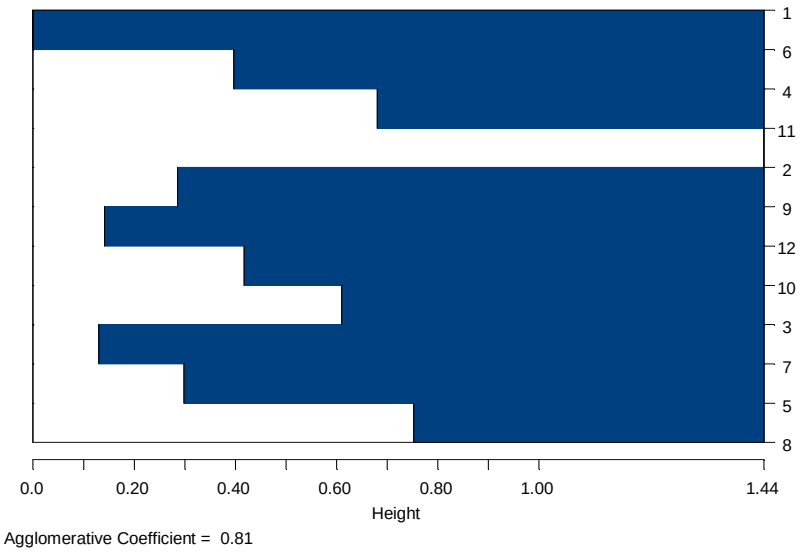

Figure 15. Banner plot of CM1 obtained for the worst case values of Reliability based risk metrics

Table 12. Clusters components of CM1 based on Reliability based risk

CLUSTER	Components	Complexity	Severity	FanOut
A	1	0.08	0.5	0.06
	6	0.08	0.5	0.06
	4	0.21	0.25	0.25
	11	0.67	0.50	0.06
B	2	0.31	0.99	0.50
	9	0.50	0.99	0.50
	12	0.60	0.99	0.60
	10	0.40	0.99	0.20
	3	0.36	0.75	0.38
	7	0.40	0.75	0.40
	5	0.40	0.50	0.50
C	8	0.50	0.99	1.00

Table 12 shows the clusters A, B and C of components of CM1 corresponding to Figure 14 and Figure 15.

5.3 Maintainability-Based Risk Metrics

According to NASA standard on software safety [27], risk is defined as a function of the anticipated frequency of occurrence of an undesired event, the potential severity of the resulting consequences and the uncertainties associated with the frequency and severity. Risk assessment is an integral part of software risk management. Several types of risks are ushered into the system when it undergoes maintenance, like project risk, usability risk and maintainability risk [20].

- Project risk basically concerns that the maintenance project cannot be completed within the budget or timeframe because of an unproductive maintenance process or deficiency of personnel and maintenance resources.
- Usability risk focuses that the maintenance conducted on the system will trigger problems and failures. It takes into account the functionality, performance and software failure risk.
- Maintainability based risk answers the question how complex will it be to maintain the system in future because of the way we handled maintenance task [2].Maintainability based risk is defined as the product of probability of performing maintenance task and the cost of performing this task. This can be used to identify the most risky parts of the system. More than 65% of the life cycle of a software project is spent in maintenance [1].In accordance with NASA-STD-8719 standard maintainability based risk is defined as the product of the probability of carrying out maintenance tasks and the impact of these tasks [27].If the software system has good maintainability it can be easily modified to fix faults.

In [1] architecture-level maintenance risk assessment methodology has been presented for assessing the maintainability based risk into the context of corrective maintenance early in the software life cycle. We present brief details of the methodology used in [1] for sake of completeness. Corrective software maintenance deals with fixing defects that escape detection before release and that which manifest as field failures [3].The methodology proposed in [1] for estimating the maintainability based risk depended on the architectural artifacts such as system requirements, system design and their evolution through the life cycle of the system. In this methodology, the requirements

maturity was first estimated and mapped into the components stability. Then the initial change probabilities of the components were estimated based on the maintenance type and the data available. The initial change propagation probabilities and the change propagation probabilities between them were used to get the unconditional probability of change of the components of the system. To get the Impact of maintenance tasks, the size of change of change between the components of the system was used. Finally, the product of the unconditional change probability and the maintenance impact was used to obtain the maintainability-based component risk factor [1], [4].

Requirements Maturity Index*:*

Requirements Maturity Index is estimated by analyzing their evolution across the releases of the system [1], [4]. The IEEE 982 standard suggested software maturity index to quantify properties of the requirements evolution [26]. In [5] Software Maturity Index was adapted to Requirements Maturity Index (RMI) to measure the requirements stability. In [1] the Use Case Maturity Index (UCMI) was adapted and function points were used as a size measure for the usecases.

UCMI for the usecase uc_i was given by

$$\text{UCMI} = \frac{U_T - U_C}{U_T} \qquad (5.2)$$

Where U_T is the function point of usecase uc_i in the current release

U_C is the function point size of the change in the usecase uc_i in the next release from the current release due to requirements change of change scenario.

Initial Change Probabilities:

In [1] the Sequence Diagrams were used to identify the set if components that contributed to each use case. Use case stability was then mapped to component stability and Initial Change Probability of system components was consequently estimated. For components that were part of multiple scenarios, the maximum ICP, that is the worst case scenario was considered.

Change Probabilities:

Change Propagation probability CP= [$cp_{i/j}$] is the conditional probability that change originating in one component of the architecture requires changes in other components to be made [4]. Initial Change Probabilities vector of the components were multiplied by the conditional change probabilities vector obtained from the system architecture in order to account for the dependency among the components of the system [1][4].

Size of Change:

Size of change is defined as the ratio between the number of affected methods of the receiving component that was caused by the changes in the interface of the providing components and the total number of methods in the receiving components [1][4].

Impact of Maintenance task:

The impact of maintenance task was obtained by using the size of change between pairs of the system components.

5.3.1 CM1 Case study Results based on Maintainability-based risk

We applied Wards clustering method as it has the highest agglomerative coefficient and works better than the other methods (Single Link, Complete Link, Average Link, Wards Method) for all the components of CM1 system taking into consideration the maintainability based risk using parameters such as Change Probabilities, Normalized maintenance Impact Fan-out, Normalized Maintenance Impact Fan-In [1], [2], [3] obtained for the CM1 system as a whole. Figure 16 shows the classification of components of CM1 based on the maintainability risk using Wards method. Figure 17 shows the corresponding Banner Plot. Table 13 shows the clusters formed and the components in them. Agglomerative coefficient value obtained was 0.85, which indicates that the strength of the cluster is good.

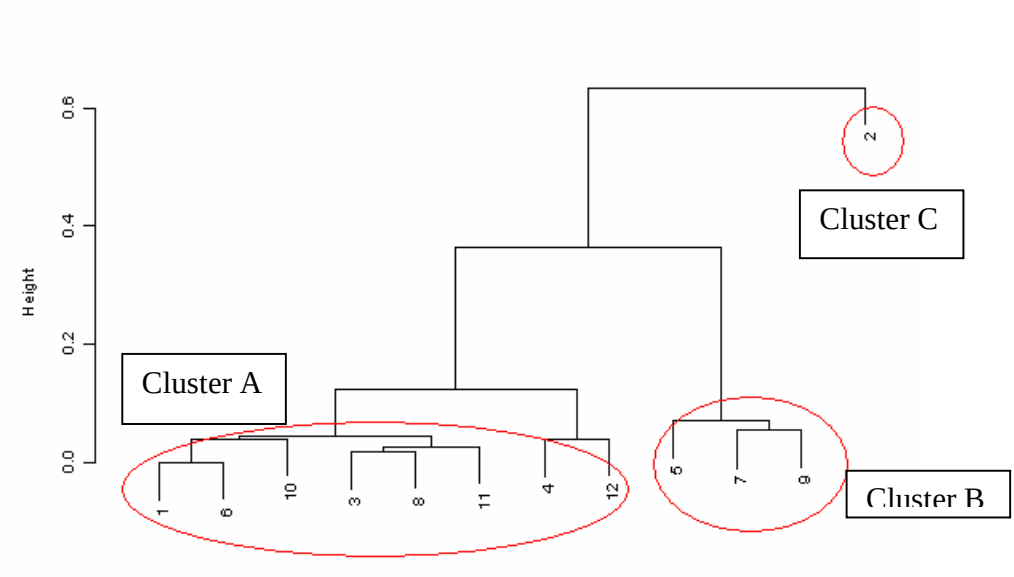

Figure 16. Clustering tree of CM1 obtained based on Maintainability based risk metrics (Agglomerative coefficient – 0.85)

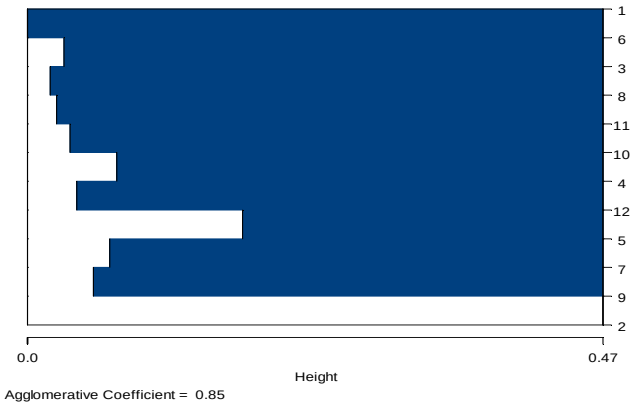

Figure 17. Banner Plot of CM1 obtained based on maintainability based risk metrics

Table 13. Clusters of components of CM1 based on Maintainability-based risk

Cluster	Components	Change Probabilities	Normalized Maintenance Impact FanOut	Normalized Maintenance Impact FanIn
A	1	0.09	0.13	0.17
	6	0.09	0.13	0.17
	10	0.11	0.10	0.17
	3	0.11	0.12	0.17
	8	0.11	0.14	0.17
	11	0.13	0.13	0.17
	4	0.13	0.17	0.17
	12	0.17	0.19	0.17
B	5	0.23	0.32	0.19
	7	0.20	0.27	0.17
	9	0.20	0.25	0.22
C	2	0.42	0.42	0.46

When Figure 14 and Figure 16 are compared, it is obvious that the components of the CM1 were classified in a different way based on Reliability based risk and Maintainability based risk. This implies that the components behave differently when we different attributes are considered. Hence components were clustered differently when clustered based on reliability and maintainability based risk metrics.

The Table 13 shows the three distinct clusters A, B and C components which form the cluster and corresponding change probabilities, Normalized Maintenance Impact FanOut and Normalized Maintenance Impact FanIn values., One cluster is formed by components 1, 6, 10, 3, 8, 11, 4, 12, the other is formed by 5, 7, 9 and component 2 alone forms another cluster when clustering is performed, based on Maintainability Risk. Components in cluster A, that is 1, 6, 10, 3, 8, 11, 4, 12 have relatively lower change probabilities, lower Normalized maintenance Impact FanOut and lower Normalized Maintenance Impact FanIn. Components in Cluster B, 5, 7, 9 have moderate change probabilities, moderate Normalized Maintenance Impact FanOut and moderate Normalized Maintenance Impact FanIn. Component 2 has relatively higher change probabilities, higher Normalized Maintenance Impact FanOut and higher Normalized Maintenance Impact FanIn than the other components.

Hence, its evident from the Table 13 that the component 2 is very dissimilar from others when classified according to maintainability based risk and is the most critical

component when classified based on maintainability based risk as it has higher values for change probabilities, normalized maintenance impact fan out and normalized impact fan in. But, according to the Table 12 component 8 is dissimilar from others. Hence, when classifying the components in the early life cycle both reliability and maintainability based risk should be considered.

We used Wards method to cluster as it gave the highest agglomerative coefficient value of 0.85, compared to the other methods, indicating that the strength of cluster obtained by Wards method is better than the others.

This way clustering could be used in the early software life cycle for the classification of software components based on reliability based risk and maintainability based risk.

Chapter 6: Classification of Software Components at the Operational Stage

We implemented hierarchical Wards Clustering on the Indent case study [36] [37] , using the component level measurements, Expected Visit Counts and the Component Entropy, that are derived in the operational stage from the raw and aggregated measures of visit counts[10].

6.1 Indent case study

Indent is an open source software project [36] [37], which consists of about 9 files, totaling about 7000 lines of code, used to beautify the C code. When Indent is ran on a C program, it has no effect on the functionality of the code, but makes the code more readable and aesthetically pleasing. Appearance of C programs could be changed in many ways such as

- Adding or removing white space
- Changing the indentation of blocks , declarations and parenthesis
- Stylish parameters could be altered

Indent has ten versions of source code, multiple CVS logs, many source code change logs and a regression test suite along with a test driver and an oracle. The latest version of Indent has about 11,000 lines of code, but the earliest version had only about 7000 lines of code. There are two change logs with 66 entries for all the ten versions of Indent [10].

We used the component level measurements, Expected visit counts and the Component Entropy to implement clustering on the Indent case study. A methodology to estimate these metrics on Indent was presented in [10]. We present a brief description of the methodology used in [10] for better understanding of the metrics used in clustering.

Profiling Software:

Information about the execution path of a program and the number of times parts of program are executed is stored in tools called Profilers. They can store information at the basic block level, line level, or the function level. Profilers can be sample-based tools or event-based tools. Sample-based tools collect data periodically based on the sampling

time period. Event-based tools collect data for every event that occurs. Sample based tools have less overhead but are less accurate, as they could miss events that occur between sample periods. Event based tools introduce more overhead but are much more accurate as they can not miss the events between the sample periods.

In [10] information on software executions were collected with the sample based profiler, Gprof. It provides two types of profiles: a call graph and a flat profile [10]. A call graph represents the control flow and the information in it, describes the call tree of the program and it is sorted by amount of time spent in each function and its children. The Flat profile lists all functions called, the number of times each was called and how long each execution took. In [10] the Indent software was instrumented with the Gprof profiler, and the information needed was extracted from the call graph as the model used in [10] depended on the flow of control transfer. 158 test cases were run, while profiling them, which gave 158 profiles [10].

Transition Probabilities:

The data in the call graph obtained from the Gprof profiler, representing the transition counts from a function f to another g was studied to calculate the transition probabilities [10]. The transition probability matrix was calculated [10] using the equation

$$P_{ij} = \frac{n_{ij}}{\sum_{j} n_{ij}} \qquad (6.1)$$

Where P_{ij} represents the probability that component *i* calls component *j*. The probability of component *i* calling component *j* is equal to the number of times component *i* calls j (n_{ij}) divided by the sum of the number of times component *i* calls any other component (n_i).

Fault Identification:

In [10] a methodology for the identification of the location of each fault has been presented. Firstly, all the test cases were run on the earliest version of Indent, version 2.2.0. The failed test cases were re run on the remaining 9 versions of the software. Thus, the release in which the fault was fixed was identified. Also, general time period of when

the fix was made was known. Once this was known, all the changes in the changelog for the time period the bug would have been fixed was searched and read. By looking at the testcase, the diff files, the output and the expected output the reason for the test case failure could be known and the description of that bug could be found in the change logs. This method of mapping failures to fixes was successful for 30 of the 34 failed test cases [10].

6.2 Dynamic metrics for Indent

The dynamic metrics expected visit counts and the component Entropy were used to implement clustering on the Indent case study. The way these metrics were derived in [10] are as follows.

Component Entropy:

An approach presented in [14] was used in [10] for the uncertainty analysis based on the concept of entropy. The theory of entropy was used to calculate the amount of uncertainty in a Discrete Time Markov chain (DTMC). The entropy of a component *i* is defined as the conditional entropy and is given by

$$H_i = -\sum_j p_{ij} \log(p_{ij}) \qquad (6.2)$$

where, p_{ij} represents the probability that the control transfers from component *i* to component *j*.

The transition probabilities were used to estimate the system uncertainty, the expected execution rates and the uncertainty of each component [10]. The entropy of the component *i*, would be higher if it transfers the control to more components and the transition probabilities are equiprobable [14]. Hence, components with higher entropy are considered critical as they affect larger part of the system [14].

Expected Visit Counts:

Expected visit counts values for the Indent that was computed in [10] was used for clustering. The methodology in [22] [11] was used to compute the expected visit counts for a component [10]. It was assumed the control of the system is transferred among

modules based on a Markov process [22]. There is an associated reliability with each module that gives the probability that the module would operate correctly when called and would transfer control successfully when finished. Eventually, the system would either fail or complete its task successfully and enter a terminal state. The expected visit counts, v_i represents the expected number of visits to a state *i* that is the expected number of executions of a component *i*.

6.3 Clustering Results for Indent

We implemented clustering on the file level, for 9 components of the Indent using the derived dynamic metrics, Component Entropy and the Expected Visit Counts which are derived from the raw and aggregated visit counts [10]. The Figure 18 shows the clustering tree obtained on the 9 components of the Indent Case Study. The Agglomerative Coefficient value (AC) obtained was 0.923. This indicates a good quality of clustering.

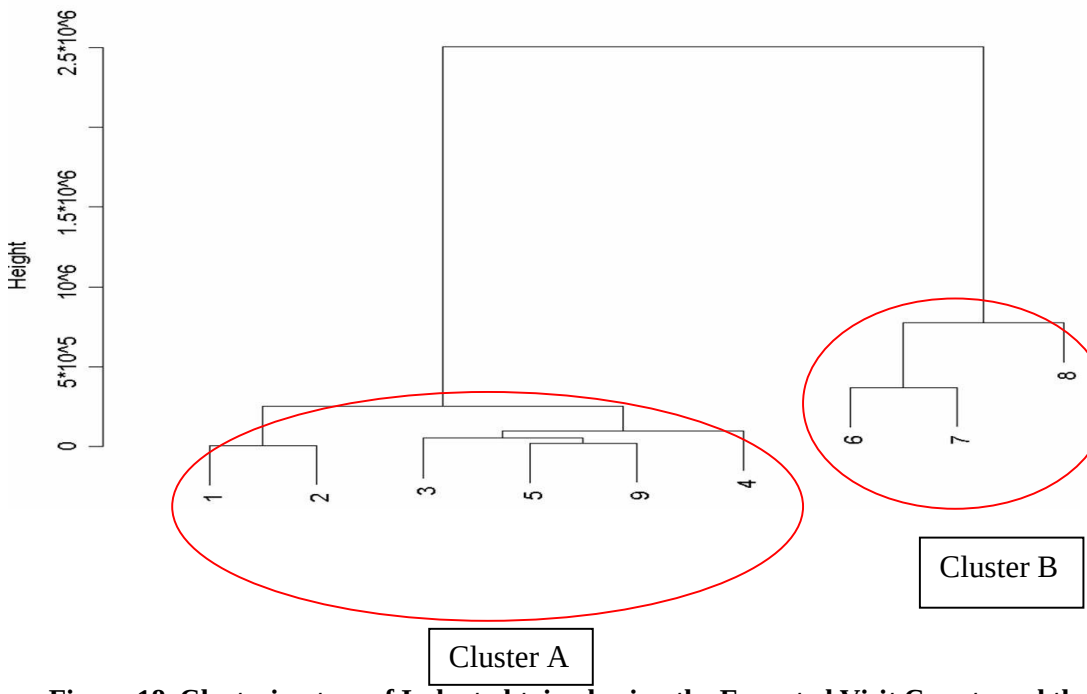

Figure 18. Clustering tree of Indent obtained using the Expected Visit Counts and the Component Entropy values

As seen in Figure 18 we find that there are two distinct clusters of components, Cluster A (Components 1, 2, 3, 5, 9 and 4) and Cluster B (Components 6, 7 and 8). Table 14 shows the Clusters, the components in each cluster and the number of failed test cases that required a fix in the component. Components 6, 7 and 8 are the three most frequently executed components and they had high number of test cases that required a fix in the component. Thus, the components that were the most frequently executed and that had maximum number of test cases that required a fix were clustered together as cluster B. This way, Clustering could group the components of Indent into meaningful clusters based on the metrics available late in the software life cycle.

Table 14. Clusters of Components for the Indent

Cluster	Component numbers	Number of Test cases that required a Fix in the Component
A	1 2 3 5 9 4	0 0 10 0 1 0
B	6 7 8	10 7 2

Chapter 7: Conclusion

In this thesis we have presented how clustering could be used for the classification of software components throughout the software life cycle. The basic assumption was that components that have similar metric values behave similarly. As clustering group's components into homogeneous clusters, it would be possible to rank the clusters and assign similar activities to all the components in a cluster.

We implemented clustering on the software components of several case studies using metrics derived in different phases of software life cycle. We used hierarchical clustering methods, the Expectation Maximization clustering method and also ran the J48 classifier to obtained decision trees for the components of twelve real NASA projects. We also implemented hierarchical clustering method on a case study, CM1 that is derived from the Data Metrics Program and another case study, Indent, open source software. Clustering results obtained have been presented in this thesis and several observations were made.

- Clustering results obtained on the components of the twelve real NASA projects during the requirements specification based on the Process/Product metrics obtained from the Software Integrity and Level Assessment Process (SILAP) helped us draw several conclusions.
    - One observation was that the Wards and the EM clustering results obtained with the direct scores and the weighted scores (Consequence and Error Potential) from SILAP were different. This implies that there is loss of information because of weights assigned in SILAP.
    - The decision trees obtained for each project were different. Different attribute had highest information gain in different projects. This reveals that each project is different.
- Clustering results obtained on the components of case study CM1, based on the reliability and maintainability based risk metrics at the architectural level helped us draw a few conclusions
    - Wards method works the best for small sample datasets, as it has the highest agglomerative coefficient than any other hierarchical clustering method.

    - Another observation was that clustering results were the best when the fusion of the reliability based risk metrics across all scenarios was done using the worst case values. This is because for risk worst case is considered the most important.
    - Components were clustered in accordance with the ranking based on criticality given by the NASA domain experts when clustered based on the reliability based risk metrics.
    - Clustering results obtained with the reliability based risk metrics and the results obtained with the maintainability based risk metrics were different. This difference in the grouping of the components is because Components behave differently when different attributes are considered.
- Clustering results obtained using the code metrics obtained at the operational stage for an open source software, Indent, clustered the most frequently executed components together.

All the results obtained, revealed that clustering helps in the classification of the software components into homogeneous clusters through out the software life cycle.

Chapter 8: References

1. W.Abdelmoez, K.Goseva-Popstojanova, and H.Ammar, "Methodology for Maintainability-Based Risk Assessment", *2006 Annual Reliability and Maintainability Symposium (RAMS 2006)*, Newport Beach, CA, January 2006.
2. W.Abdelmoez, K.Goseva-Popstojanova ,and H.Ammar, "Maintainability Based Risk Assessment in Adaptive Maintenance Context", *2nd International Predictor Models in Software Engineering Workshop (PROMISE 2006)*, Philadelphia, September 2006.
3. W.Abdelmoez, K.Goseva-Popstojanova, and H.Ammar, "Using Maintainability-based Risk Assessment and Severity Analysis in Prioritizing Corrective Maintenance Tasks", *Supplemental Proc. 17th IEEE International Symposium on Software Reliability Engineering (ISSRE 2006)*, Raleigh, NC, November 2006.
4. W.Abdelmoez, "Model based Risk Assessment", Master Thesis, West Virginia University, 2006.
5. S.Anderson, M.Felici, "Quantitative Aspects of Requirements Evolution". *In the Proceedings of 26th Annual International Conference on Computer Software and Applications Conference (COMPSAC 2002), IEEE Computer Society*, August 2002, pp 27-32.
6. G.Booch, I.Jacobson, and J.Rumbaugh, "The unified Modelling Language Guide", *Addison Wesley*, 1998.
7. W.Dickinson,D.Leon, and A.Podgurski, "Finding Failures by cluster analysis of Execution Profiles", *Proc. 23rd International Conference on Software Engineering*, 2001, pp. 339 – 348.
8. W.Dickinson, D.Leon, and A.Podgurski, "Pursuing Failure: The Distribution of Program Failures in a Profile Space", *Proc. 10th European Software Engineering Conference and 9th ACM SIGSOFT Symposium on Foundations of Software Engineering*, 2001, pp.246-255.
9. K.Goseva-Popstojanova, A.Hassan, A.Guedem, W.Abdelmoez, D.Nassar, H.Ammar and A. Mili, "Architectural Level Risk Analysis using UML", *IEEE Transactions on Software Engineering*, Vol.29, No.10, 2003, pp. 946-960.

10. M. Hamill, “Empirical Analysis of Software Reliability”, Masters Thesis, West Virginia University, 2006.
11. M.Hamill and K.Goseva-Popstajanova, “Architecture based Software Reliability: Why only a Few Parameters Matter”, *COMPSAC* (submitted).
12. A.K. Jain, M.N. Murthy and P.J. Flynn, “Data Clustering: A Review”, *ACM Computing Surveys*, Vol. 31, No. 3, 1999, pp. 264 – 323.
13. A.K.Jain, K.Nandakumar, A.Ross, “Score Normalization in Multimodal biometric systems”, *Pattern recognition*, Vol.38, No.12, pp. 2270-2285, December 2005.
14. S.kamavaram and K.Goseva-Popstajanova, “Entropy as a Measure of Uncertainity in Software Reliability”, *Supplemental proc. 13’th Int’l Symp. On Software Reliability Engineering*, Nov 2002, pp. 209-210.
15. G.Punj and D.W.Stewart, “Cluster Analysis in Marketing Research: Review and Suggestions for Applications” ,*Journal of Marketing*, Vol. 20, 1983, pp. 134-48.
16. G.Punj and D.W .Stewart, “Cluster Analysis in Marketing Research: Review and Suggestions for Applications” , *Journal of Marketing*, Vol. 20, 1983, pp 134-48.
17. A.Ross and A.Jain, “Information Fusion in Biometrics”, *Pattern Recognition Letters*, Vol. 24, Issue 13, Sep 2003, pp. 2115 – 2125.
18. P.J.Rousseeuw, “A Visual Display for Hierarchical Classification”, *In E.Diday, Y.Escoufier, L.Lebart, J.Pages, Y.Schektman, R.Tomassone. (Eds). Data Analysis and Informatics*, Vol.4, Amsterdam : New Holland ,1986, pp. 743-48.
19. P.J.Rousseeuw and L.Kaufman, “Finding Groups in Data: An Introduction to Cluster Analysis”, *New York: John Wiley & Sons*, Inc, 2005.
20. S.Sherer, “Using Risk Analysis to Manage Software Maintenance”, *Software Maintenance: Research and Practice*, Vol. 9, 1997, pp.345-364.
21. T. Shizhong, M. Khoshgoftaar, and N. Seliya, “Analyzing software measurement data with clustering techniques”, *IEEE Intelligent Systems*,Vol.19, No 2, 2004, pp.20-27.
22. K.Siegrist, “Reliability of System with Markov Transfer of Control”, *IEEE Trans. Reliability*, Vol.14 No.7, 1988, pp.1049-1053.
23. C.Sundarajan, “Guide to Reliability Engineering, Data Analysis, Applications, Implementations, and Management”, Van *Nostrand Reinhold*, 1991.

24. I.H. Witten, E.Frank, “Data Mining: Practical Machine Learning tools and Techniques”, *Morgan Kaufmann*, 2005.
25. I.H. Witten, E. Frank, L. Trigg, M. Hall, G. Holmes, and S.J. Cunningham, “WEKA: Practical Machine learning tools and techniques with Java implementations”, *Morgan Kaufmann*, 1999.
26. IEEE Std 982.1 – IEEE Standard Dictionary of Measures to Produce Reliable Software.
27. NASA-STD-8719.13A, “Software Safety NASA Technical Standard”, Sep.15, 1997.
28. “S plus 6 for Windows: Guide to statistics”, Insightful, Vol. 2, July 2001.
29. http://grb.mnsu.edu/grbts/doc/manual/Expectation_Maximization_EM.html
30. http://grb.mnsu.edu/grbts/doc/manual/J48_Decision_Trees.html
31. http://www.cs.toronto.edu/~roweis/csc2515-2003/notes/lec7x.pdf
32. http://www.r-project.org/
33. Metrics Data Program, NASA IV & V Facility http://mdp.ivv.nasa.gov/.
34. Rational Rose Real-Time. http://www.rational.com/products/rosert/index.jtmpl
35. http://www2.chass.ncsu.edu/garson/PA765/cluster.htm
36. http://www.gnu.org/software/indent/indent.html
37. http://www.xs4all.nl/~carlo17/indent/