Research Article

Predicting Software Projects Cost Estimation Based on Mining Historical Data

Hassan Najadat,¹ Izzat Alsmadi,² and Yazan Shboul¹

¹ Computer Information Systems Department, Jordan University of Science and Technology, P.O. Box 3030, Irbid 22110, Jordan

² Computer Information Systems Department, Yarmouk University, P.O. Box 566, Irbid 21163, Jordan

Correspondence should be addressed to Izzat Alsmadi, alsmadi@gmail.com

Received 18 November 2011; Accepted 16 January 2012

Academic Editors: J. Cao and O. Greevy

Copyright © 2012 Hassan Najadat et al. This is an open access article distributed under the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

In this research, a hybrid cost estimation model is proposed to produce a realistic prediction model that takes into consideration software project, product, process, and environmental elements. A cost estimation dataset is built from a large number of open source projects. Those projects are divided into three domains: communication, finance, and game projects. Several data mining techniques are used to classify software projects in terms of their development complexity. Data mining techniques are also used to study association between different software attributes and their relation to cost estimation. Results showed that finance metrics are usually the most complex in terms of code size and some other complexity metrics. Results showed also that games applications have higher values of the SLOCmath, coupling, cyclomatic complexity, and MCDC metrics. Information gain is used in order to evaluate the ability of object-oriented metrics to predict software complexity. MCDC metric is shown to be the first metric in deciding a software project complexity. A software project effort equation is created based on clustering and based on all software projects' attributes. According to the software metrics weights values developed in this project, we can notice that MCDC, LOC, and cyclomatic complexity of the traditional metrics are still the dominant metrics that affect our classification process, while number of children and depth of inheritance are the dominant from the object-oriented metrics as a second level.

1. Introduction

Software companies are interested in determining the software development cost in the early stages to control and plan software tasks, risks, budgets, and schedules. In order to analyze software cost, we have to gather metrics from the current and the previous projects and draw similarities and analogies to be able to come up with predictions regarding the current project.

It is important for software companies to benefit from the company historical software data in order to use it to estimate the cost and the schedule of the new ordered projects. So they can manage the budget, the development staff, and the schedules for the work process.

It is essential for both developers and customers to get accurate software cost estimation by accurately estimating the new project cost. Project managers can provide the customers with an accurate deadline for their projects and

debate some of the contract negotiation's issues, so that customers can expect actual development costs to be in line with estimated cost, and these estimations can be used by developers to generate reports and proposals and to determine what resources are needed to commit to the project and how well these resources will be used. Accurate software cost estimation makes projects management easier to be managed and controlled as resources are better matched to the real needs.

Many researchers in software engineering field have studied in depth how to predict the software project cost which is important for the project managers and software development organizations. Cost estimation or what is called by other researchers "effort prediction" is the process of estimating the cost of the software system development. This estimation can generally be estimated through three methods: experts' judgment, algorithmic model, and by analogy [1].

FIGURE 1: McCabe graph and its pseudocode.

In this work, a new approach is proposed to predict the cost and the complexity of building new software projects using the data mining classification techniques based on the metrics similarity. We have built a dataset from open source projects classified according to three domains (communication, finance, and games) Important software related features are extracted from them using a tool developed by Alsmadi and Magel [2]. This work builds a channel between both the data mining and the software engineering approaches.

It is important for software companies to specify the current new ordered project cost in a bid to manage the project cost, schedule, and to determine the staff and time table of the project development.

Most of the previous works have suffered from some problems of accuracy and building their models depending on their own datasets [3]. In this work, we built a dataset or a dictionary of prediction where users can relate their projects to a domain in the dictionary and then use such information to make the cost estimation.

A little research has been done on estimating the effort of object-oriented software that applies the use of case model [4]. Also, there are few researches that analyze the effort prediction of open source code projects.

2. Background and Related Work

Software metrics are the software features (measures) and characteristics. Since software measurements are essential in software engineering, there have been many researches over the last four decades to provide a comprehensive measure of software complexity and to use it in software cost estimation and software analysis.

Although the first software metrics book was published in 1976 [5], the history of software metrics researches went back to the 1960s, when the lines of code (LOC) metric was used to measure the productivity of the programmer and software complexity and quality. LOC was used as a main key in effort prediction for some prediction models such as [6, 7].

In 1984, Basili and Perricone made an analysis of the relationship between module size and error proneness [8]. They analyzed some modules with LOC less than 200 lines of FORTRAN language project, and they concluded that the modules with less LOC have the greater bug density. Another research article published in 1985 made an analysis on the software of Pascal, PL/S, and assembly language [9]. In their research Shen et al. [9] concluded that the higher bug density occurs with large modules when LOC is greater than or equals 500. In his research of Ada program in 1990, Withrow [10] validated the concave relationship between software size and bug density where the bug density increases when LOC value is more than 250 and decreased when LOC value is less than 250.

In the mid of the 1970s, the interest in software complexity increased when graph theoretical complexity is discussed by McCabe in [11]. He developed a mathematical technique for program modularization. Some definitions of the graph theory were used in order to measure and control the number of paths through a software program that is called the Cyclomatic Complexity metric. Then this metric has been used for complexity measurements instead of size metrics.

McCabe Cyclomatic Complexity metrics compute the number of paths that may be executed through the program using the graph theory. The nodes of the graph represent the source code lines of the software program, and the directed edges between nodes represent the second source code lines that may be executed. Figure 1 represents an example of the McCabe Cyclomatic Complexity graph.

According to McCabe [12], the Cyclomatic Complexity metric value is measured by the following formula:

$$V(G) = e - n + 2, \quad (1)$$

where e refers to edges and n refers to the nodes. For example, in the above graph $V(G) = 7 - 6 + 2 = 3$.

In their research in 1984, Basili and Perricone [8] found a correlation between McCabe Cyclomatic Complexity and module sizes. They discovered that large modules have high complexity.

Halstead introduced other software metrics in 1977 [13]. These metrics have been developed in order to estimate the programming effort. The Halstead metrics are measured using some statistic numbers. These statistics are

- (i) $n1$ = number of unique or distinct operators which appear in that implementation;
- (ii) $n2$ = number of unique or distinct operands which appear in that implementation;
- (iii) $N1$ = total usage of all of the operators which appear in that implementation;
- (iv) $N2$ = total usage of all of the operands which appear in that implementation.

The above statistics contain operators and operands. The operators are used to specify the manipulation to be performed while the operands are used as logic units to be operated. From the above statistics Halstead complexity metrics are defined as

- (i) the vocabulary $n = n1 + n2$,
- (ii) the program length $N = N1 + N2$.

In 2004, Fei et al. in their article [14] proposed an improvement on the Halstead complexity metrics. They added weights to the Halstead metrics. They gave different operators and operands different weights.

Six object-oriented design metrics were developed and evaluated by Chidamber and Kemerer in 1994 [15]. These object-oriented metrics are called CK metrics. The CK metrics that resulted from Chidamber and Kemerer are weighted methods per class (WMC), depth of inheritance tree (DIT), number of children (NOC), coupling between object classes (CBO), response for a class (RFC), and lack of cohesion in methods (LCOM).

2.1. Cost Estimation. Software systems and applications are the most expensive part of the computer systems due to the human effort that is used in producing software systems. This reason (expansive software development) motivated many researchers to focus on this aspect of research.

Software cost estimation or what is called by other researchers “effort prediction” is the process of estimating software cost accurately. Over the past 30 years, many studies have been conducted in the software cost estimation field [16] where two main types of cost estimation methods have been discussed including the algorithmic and nonalgorithmic methods. Some cost estimation models use data from previous projects in order to derive cost formulas; these models are called empirical models like COCOMO model [17], while other models depend on global assumptions to derive their cost formulas. These models are called analytical models; for example, Putnam in his paper [7] proposed an approach to accurately estimate effort, cost, and time of software projects. Leung and Fan in their article [16] gave an

overview of the software cost estimation, and they highlighted the importance of the accuracy in estimating the cost of the software.

2.2. Cost Estimation Models. Cost estimation models can be classified into two categories according to the approach and the procedures used to measure the software cost. There are two major categories which are algorithmic and nonalgorithmic models.

2.2.1. Nonalgorithmic Models. There are many non-algorithmic models used in software cost estimation. The following are some of these models used in the literature.

(i) *Analogy Model.* Prediction by analogy is one of the most non-algorithmic methods used in effort prediction. Using analogy prediction depends on previous completed projects where we can predict efforts using actual existing projects cost values. In prediction by analogy, the software project is characterized through variables, and then Euclidean distance is measured in n -dimensional space. A prediction tool to find an analogous of the current software project from the set of completed projects called ANGEL is developed, where it is flexible to return three analogous [1].

(ii) *Expert Judgment.* Usually, more than one expert's opinions are involved in the estimation process. Therefore, deriving the software cost estimation is neither explicit nor repeatable. In fact, experts consider some techniques such as Delphi technique or PERT [18, 19] mechanisms in a bid to reach a consensus between each other resolve the inconsistencies in the estimation. In the Delphi mechanisms, a coordinator asks each expert to fill a form to record estimations, and then the coordinator prepares a summary of all the estimations from the experts [16].

Parkinson Model. The cost is determined by the available resources rather than based on an objective assessment according to Parkinson's principle [20].

Algorithmic models. The algorithmic model estimates the software cost through some formulas that depend mainly on the size of the project which is measured in terms of function point, object point, and lines of code (LOC). Beside the size of the project, there are a number of variables that are involved in the algorithmic model function such as:

$$\text{Effort} = f(y_1, y_2, y_3, \dots, y_n), \quad (2)$$

where Effort is a cost estimation measure that is usually measured by (person-month), and f refers to the function form, and $(y_1, y_2, y_3, \dots, y_n)$ refers to the cost factors.

Boehm introduced the first version of COCOMO as a model for estimating the effort, cost, and schedule; this COCOMO version was called COCOMO 81 [6] where 63 projects ranging from 2000 to 100000 lines of code are used in that study. In 1997, Boehm enhanced his first version of COCOMO and introduced another model called COCOMO 2 [21]. This model provides more support for modern

software development processes. In COCOMO models, LOC is used as a software code size and given in thousands to measure the effort which is measured in person-month.

3. Goals and Approaches

A software metrics tool called CodeMetrics is built in order to extract software metrics from source codes which are used to build software metrics data set. CodeMetrics is extended from SWMetrics which was built by Alsmadi and Megal [2]. This tool can extract metrics from C#, C++, and Java source code. CodeMetrics can build and classify the software data set according to their domains (e.g., communication, finance, and games). CodeMetrics can extract some selected CK metrics as well as the traditional metrics like the metrics extracted from SWMetrics [15]. The metrics extracted are (Lines, LOC, SLOC, SLOCmath, MCDC, MaxNest, CComplexity, Averaged method per class, averaged method CComplexity, Max Inheritance, Coupling, and Number of Children).

Our metrics tool parses all the source code files of the selected language only. For example, for C#, C++, and Java languages projects, CodeMetrics parses all the files of the extension types `***.cs`, `***.cpp`, and `***.java`, respectively. Through the parsing process, a counter for each software metrics is used to compute the metric value.

Before parsing the source code we should select the language; after completing the metrics counting we can specify the project domain and save it under that domain class. CodeMetrics can be extended to more than three domains types within the domains list such as educational and scientific systems which can be added to the domains list.

3.1. Open Source Code Dataset. We have built our data set using open source projects collected from different open source web sites which are

- (i) <http://www.codeproject.com/>,
- (ii) <http://www.codeplex.com/>,
- (iii) <http://www.sourceforge.com/>.

We have classified our source code projects into communication, finance, and games domains. All collected projects are of C# code language only. Table 1 shows the number of projects for each domain, 38.1% of the gathered projects are from the games domain, 35.7% of the projects are Finance applications, and 26.2% of the projects are from communication domain. Communication applications are related to email, file transfer, client-server, and chat applications. Finance applications are related to stock and billing systems and so forth. An example of such projects is NopCommerce and LinqCommerce. NopCommerce is an open source e-commerce solution with comprehensive features that is easy to use for new online businesses and is included within finance domain. LinqCommerce was created by JMA Web Technologies Inc. in order to be a part of e-commerce solutions.

TABLE 1: Source code projects domain.

Domain	Number of application	percentage
Communication	33	26.2%
Finance	45	35.7%
Game	48	38.1%

3.2. Data Mining for Cost Estimation. We have used information gain method as a subset selection method in order to know the best subset of the metrics that play a major role in the classification process. A decision tree method (J48) is used as a classification method for the software data set depending on the software metrics. A clustering technique (K-mean) is used to classify the data set and grouping the similar projects within the same cluster.

3.3. Attribute Selection. There are many methods used for attribute subset selection such as information gain, gain ratio, and gini index methods. An attribute subset selection method is used to select the best separates of the attributes. In this work, we have used information gain method to determine the best splitting attributes. This method computes the information gain value for each attributes, and the attribute with the highest information gain is the best attribute that can identify the class label of a tuple in the data set. The following formulas are used to compute the information gain values:

$$\text{Info}(D) = - \sum_{i=0}^n p_i * \log(p_i), \quad (3)$$

where $\text{Info}(D)$ is the average amount of information needed to identify the class label and p_i is the probability that an tuple in the data set belongs to a class label

$$\text{Info}_A(D) = \sum_{k=0}^n \frac{|D_k|}{|D|} * \text{Info}(D_k), \quad (4)$$

where the $\text{Info}_A(D)$ is the expected information needed to classify a tuple based on an attribute A $|D_k|/|D|$ is the weight of the k partition

$$\text{Gain}(A) = \text{Info}(D) - \text{Info}_A(D), \quad (5)$$

where $\text{Gain}(A)$ refers to the amount of information required to classify the data set of attribute A.

3.4. J48 Decision Tree. J48 decision tree is one of the classification methods that construct a decision tree classifier. J48 classifier works as follows.

- (1) Take the data set, attribute list, and the attribute selection method as input.
- (2) Check if the tuples in the data set are all with the same class label or not.
- (3) Check if the attribute list is empty or not.
- (4) Apply attribute selection method to find the best separate attribute.

- (5) Remove the splitting attribute from the attribute list.
- (6) For each outcomes of the tree, grow subtrees for each partition.

We have used decision tree classifier as a knowledge discovery method because it can handle high-dimensional data like the data type of our data set. Another reason to use the decision tree classifier is that the construction of the decision tree does not need any domain knowledge.

3.5. Clustering by K-mean. Clustering techniques used to put the similar data object within the same cluster and the dissimilar objects in other clusters. We used K-mean clustering method to put the similar projects within the same cluster. In the K-mean clustering method we first select the number of clusters (K) and the distance method. K-means method works as follows

- (1) Randomly select K of the object, which represents a cluster center.
- (2) Assign each object to the cluster to which the object is the most similar.
- (3) Calculate the mean value of the objects for each cluster.
- (4) Reassign each object to the cluster to which the object is the most similar.
- (5) Repeat steps 3-4 until no changes.
- (6) Assign the cluster number as a class label for each object.

After applying the K-mean clustering method, each projects of the data set is labeled with the cluster number which refers to a complexity level for each project as will be described later.

3.6. Using WEKA Data Mining Tool. In order to analyze our data set using data mining techniques, we used one of the most data mining tools that contain implementations of the data mining techniques. We used WEKA 3.6 version tool. WEKA is one of the most popular tools that are used by many researchers to analyze their data. WEKA supports data mining tasks such as data preprocessing, data classification, data clustering, and attributes selection.

4. Implementation and Experimental Work

We first analyzed our dataset considering the domain types of the projects and concluding general characteristics for each domain type. We also grouped the projects using K-mean clustering method that is implemented through WEKA tool and concluding some analysis data.

4.1. Source Code Domains Analysis. This section discusses software source code according to their domains. We have analyzed software metrics within each domain and have extracted some information about each domain and made a comparison between the studied domains applications. In

this work, we have studied three types of software application domains which are the communication, the finance, and the games applications. The following two sections analyze the traditional metrics and the object-oriented metrics within domains.

4.1.1. Traditional Metrics Analysis. Collected traditional metrics show different values for each domain application. Traditional metrics collected using the developed tool includes LOC, SLOCmath, MCDC, MaxNesting, and Cyclomatic Complexity. The following graphs show some characteristics for each domain application.

The size metrics are lines, LOC, SLOC, MCDC, Slocmath, Maxnest, and Cyclomatic complexity. We have found that finance applications have higher source size codes than the other application types.

Results showed that finance applications have higher nesting values than other applications domains. Results showed also that games applications have higher values of the SLOCmath and MCDC metrics, while lower values of these three metrics are measured within communication source code applications. It clearly shows that games applications have more complex source code than finance and communication applications, while the finance applications are the least complex applications.

The above analysis and graphs conclude that these software application domains have some differences and that each application type has general characteristics. The analysis of our data set that is gathered from open sources software and classified into three domains (communication, finance, and games) showed that finance applications require the largest number of software size which is measured by computing source code lines of code. Finance applications have the deepest nesting levels which increase the complexity of the source codes.

Games applications have higher values of SLOCmath and MCDC metrics. This result is expected for the games applications because these types of software applications deal with the players' choices and the players' probabilities through playing the game. Also games applications have the highest complexity values of the Cyclomatic Complexity metric which means that games applications have larger number of execution paths relative to communication and finance applications. Results showed also that finance applications have limited number of execution paths. This result refers to the stability of the finance applications which depend on clear and static business rules.

4.1.2. Object-Oriented Metrics Analysis. Object-oriented metrics information measures the quality of the software application and helps in accurately estimating the software cost. Results for "Average Cyclomatic Complexity for methods within a class" metric showed that games applications methods are more complex than other domains applications methods, while lower complexity levels appeared within communication source code methods. In terms of the metric "average number of methods per class", results showed also that games applications classes have much more methods

than other classes at other domains, while finance application's classes are the lowest ones.

We analyzed the inheritance depth of the classes within the three domains, and we found that the depth of inheritance for the finance applications is deeper than that of communication applications, while the depth of inheritance for the games applications is the deepest of the three.

Coupling of a class measures how many methods and method calls exist in a call to all other classes. In coupling, results showed that games objects within any class are invoked many times by methods in different classes more than those in finance or communication applications. Results showed also the average number of children classes over the three analyzed domains (i.e., Games, Finance, and communication). It is clearly noticed that games application's classes have much more children classes than applications in other domains.

The above analysis shows that applications of the game domain are more complex than other software applications within other domains. Average Cyclomatic Complexity per methods shows that games applications have the highest values of this metrics which can increase the software cost (i.e., building and maintenance effort) for games applications. Games programs require to interact with the player's options and reactions, so games programmers need to include all the probabilities of the game and the interactions with the games players' options which explain the high values of object-oriented metrics for the game source codes.

4.2. Software Metrics Selection. In this section, we have analyzed the software metrics and have assigned weights for the software metrics in order to specify the metrics that determine the software domain (the dominant metric). Attribute subset selection method from data mining is used in a bid to find the dominant metrics that specify the software domain. In order to know each metric weight, we have used the information gain method (InfoGain) that is implemented by WEKA tool. In a bid to know the effects of object-oriented metrics on the analysis, we have made two methods of analysis: on the traditional metrics only (i.e., without object-oriented metrics) and on all available software metrics.

4.2.1. Traditional Dominant Metrics. Applying information gain method on the traditional metrics shows that MCDC metric is the dominant metric since its value can often determine the software domain. Table 2 shows the information gain values for other metrics. The second metric that plays an important role in determining software domain is the cyclomatic complexity with a value 0.45 which is the nearest value to the MCDC metric. LOC and SLOCmath are in the third and fourth positions with values close to each other. The MaxNest metric is the last metric that may affect the software domain.

The decision tree (J48) clearly shows the heuristic for selecting the splitting criterion metrics. Tree 2 shows that MCDC metric is at the root of the tree which means that MCDC has a major role in splitting the tree. J48 classifier has

TABLE 2: Metrics information gain value.

Metrics number	Metrics	Information gain value
1	MCDC	0.479
2	CComplexity	0.45
3	LOC	0.336
4	Slocmath	0.322
5	maxNest	0.299

TABLE 3: J48 confusion matrix.

Domain/classified as	Communication	Finance	Game
Communication	23	10	0
Finance	0	44	1
Game	1	0	47

a tree with 13 leaves and 7 levels depth with 90.4762% prediction accuracy. Table 3 shows the confusion matrix of J48 classifier. The main mismatches are within communication software where 10 communication projects are classified as finance projects. While finance and games projects have very high percentages of correctly classified projects.

Results showed that all the traditional metrics play a role in determining the software domains, and there are different ends with a domain type. The shortest path in the above tree is the path that includes two metrics only which are MCDC and MaxNest that end with 18 games projects. The longest path ends with two finance and two communication projects. This path includes the following metrics:

$$\begin{aligned} &\text{MCDC} \rightarrow \text{CComplexity} \rightarrow \text{LOC} \\ &\rightarrow \text{MaxNest} \rightarrow \text{MaxNest} \rightarrow \text{LOC}. \end{aligned} \quad (6)$$

The path that ends with the biggest number of projects is the path that ends with 47 financial projects with 10 projects mismatching the domain, and so 37 of the finance projects are found through the same path which refers to the high similarities between finance projects. The tree 3.1 shows that all the financial projects have MCDC value less than 10221, while most games projects have MCDC values larger than 10221.

4.2.2. Traditional-CK Dominant Metrics. In this section, we have added several CK metrics to the traditional metrics of the software. We have also found that MCDC metric is the most significant metric that can determine the source code domain by using information gain method as shown in Table 4.

The above table shows the metrics arranged decreasingly according to the information gain value. The CK metrics have less effect on splitting the projects except the CBO metric which has the third position within the software metrics.

The J48 classifier shows a high accuracy value (97.619%) of using traditional software metrics plus CK metrics where 123 projects are correctly classified. Table 5 shows the confusion matrix of applying J48 on traditional plus CK metrics.

TABLE 4: Traditional-CK metric information gain value.

Metrics number	Metric	Information gain value
1	MCDC	0.479
2	CComplexity	0.45
3	CBO	0.363
4	LOC	0.336
5	SLOCmath	0.322
6	MaxNest	0.299
7	avgMethod per class	0.151
8	Childern	0.147

TABLE 5: Traditional CK metrics confusion matrix.

Domain/classified as	Communication	Finance	Game
Communication	33	0	0
Finance	1	43	1
Game	1	0	47

This high accuracy value when adding CK metrics refers to the enhancement of the prediction process when considering CK-metrics in addition to the traditional metrics. Results showed that traditional CK metrics has 97.619% of accuracy while traditional metrics only has 90.4762% of accuracy.

5. Assumptions

- (i) Grouping the software projects into 3 groups using data mining clustering methods, each group represents a cost factor for the projects within each group where we assume (Low, Mid, and High) for the three grouping approach.
- (ii) Using one of the attribute subset selection methods in order to get metrics weights, we have used the information gain method and have assumed our effort formula according to the linear approach divided by 1000.

Our collected data set contains traditional metrics as well as some CK metrics. All of these metrics are numerical, and so in order to cluster the data set into similar groups we used one of the data mining approaches. We used K-mean method which is one of the known clustering techniques. In the following sections, K-means is applied on our data set in a bid to put the collected software projects within clusters according to the similarity between clustered projects.

6. Conclusion

Cost estimation remains a complex problem that attracts researchers to study and to try different approaches and methodologies to solve it. In this thesis, a new cost estimation approach for analyzing software projects is presented in order to help project managers to take their decisions. Our approach utilizes data mining techniques in the analysis process, and we have used K-mean in a bid to classify

the gathered projects into clusters and find the similarities between each cluster's projects. In order to gather the data for our dataset a software metric tool called CodeMetrics is built to extract traditional metrics as well as some of the object-oriented metrics (CK metrics).

A dataset of software metrics is established according to the projects domain types, and, to our knowledge, there is not any datasets that contain software metrics classified according to the domain type.

The following points refer to the final experimental results of this work.

- (1) Finance projects require a larger number of lines and nesting relative to games and communication projects.
- (2) Slocmath, mcddc, and complexity metrics for games projects have higher values relative to finance and communication projects, while the complexity of the communication projects is larger than finance projects.
- (3) Applications of games type have higher values of object-oriented metrics relative to the other projects (method CComplexity, averaged method per class, inheritance depth, coupling, children).
- (4) Using data mining techniques for attribute subset selection we have found that MCDC plays a big role in determining source code domains more than LOC metric as all cost estimation models depend on it.
- (5) Adding object-oriented metrics to the metrics list increases the prediction accuracy where the accuracy becomes (97.619%) instead of (90.4762%) (using J48 classifier).
- (6) Our results conclude that MCDC, LOC, MaxNets are the major metrics that affect the classification of the software projects while most of the previous studies consider LOC metric as the major metric that plays a role in effort estimation.
- (7) Adding object-oriented metrics to the metrics list increases the correlation between our model and COCOMO model where the correlation becomes 0.958219 for effort instead of 0.939904104, and the correlation of the development time becomes 0.995454 instead of 0.985798364.

References

- [1] M. Shepperd, C. Schofield, and B. Kitchenham, "Effort estimation using analogy," in *Proceedings of the 18th International Conference on Software Engineering*, pp. 170–178, Berlin, Germany, 1996.
- [2] I. Alsmadi and K. Magel, "Open source evolution analysis," in *Proceedings of the 22nd IEEE International Conference on Software Maintenance (ICSM '06)*, Philadelphia, Pa, USA, 2006.
- [3] B. Boehm, B. Clark, E. Horowitz, R. Madachy, R. Shelby, and C. Westland, "Cost models for future software life cycle process: COCOMO 2.0," in *Annals of Software Engineering Special Volume on Software Process and Product Measurement*,

- J. D. Arther and S. M. Henry, Eds., vol. 1, pp. 45–60, J.C. Baltzer AG, Science Publishers, Amsterdam, The Netherlands, 1995.
- [4] K. Ribu, *Estimating Object-Oriented Software Projects with Use Cases*, M.S. thesis, University of Oslo Department of Informatics, 2001.
 - [5] T. Gilb, *Software Metrics*, Chartwell-Bratt, 1976.
 - [6] B. W. Boehm, *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1981.
 - [7] L. H. Putnam, “A general empirical solution to the macro software sizing and estimating problem,” *IEEE Transactions on Software Engineering*, vol. 4, no. 4, pp. 345–361, 1978.
 - [8] V. R. Basili and B. T. Perricone, “Software errors and complexity: an empirical investigation,” *Communications of the ACM*, vol. 27, no. 1, pp. 42–52, 1984.
 - [9] V. Y. Shen, T. J. Yu, S. M. Thebaut, and L. R. Paulsen, “Identifying error-prone software—an empirical study,” *IEEE Transactions on Software Engineering*, vol. 11, no. 4, pp. 317–324, 1985.
 - [10] C. Withrow, “Error density and size in Ada software,” *IEEE Software*, vol. 7, no. 1, pp. 26–30, 1990.
 - [11] T. J. McCabe, “A complexity measure,” *IEEE Transactions on Software Engineering*, vol. SE-2, no. 4, pp. 308–320, 1976.
 - [12] J. Han and M. Kamber, *Data Mining: Concepts and Techniques*, Morgan Kaufmann, San Francisco, Calif, USA, 2th edition, 2006.
 - [13] M. H. Halstead, *Elements of Software Science*, Elsevier North Holland, 1977.
 - [14] Y. Y. Fei, Z. Zhi, and Z. S. Chao, “Improvements about Halstead model in software science,” *Journal of Computer Applications*, pp. 130–132, 2004.
 - [15] S. R. Chidamber and C. F. Kemerer, “Metrics suite for object oriented design,” *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.
 - [16] H. Leung and Z. Fan, *Software Cost Estimation*, Department of Computing, The Hong Kong Polytechnic University, 2002.
 - [17] T. Xie, S. Thummalapenta, D. Lo, and C. Liu, “Data mining for software engineering,” *Computer*, vol. 42, no. 8, pp. 55–62, 2009.
 - [18] H. Leung and Z. Fan, *Software Cost Estimation*, Department of Computing, The Hong Kong Polytechnic University, 2002.
 - [19] J. D. Aron, “Estimating Resource for Large Programming Systems,” NATO Science Committee, Rome, Italy; October 1969.
 - [20] G. N. Parkinson, *Parkinson’s Law and Other Studies in Administration*, Houghton-Mifflin, Boston, Mass, USA, 1957.
 - [21] B. W. Boehm et al., *The COCOMO 2.0 Software Cost Estimation Model*, American Programmer, 1996.