Utility of CK Metrics in Predicting Size of Board-Based Software Games

NOSHEEN SABAHAT*, ALI AFZAL MALIK**, AND FAROOQUE AZAM*

RECEIVED ON 05.02.2017 ACCEPTED ON 21.02.2017

ABSTRACT

Software size is one of the most important inputs of many software cost and effort estimation models. Early estimation of software plays an important role at the time of project inception. An accurate estimate of software size is, therefore, crucial for planning, managing, and controlling software development projects dealing with the development of software games. However, software size is unavailable during early phase of software development. This research determines the utility of CK (Chidamber and Kemerer) metrics, a well-known suite of object-oriented metrics, in estimating the size of software applications using the information from its UML (Unified Modeling Language) class diagram. This work focuses on a small subset dealing with board-based software games. Almost sixty games written using an object-oriented programming language are downloaded from open source repositories, analyzed and used to calibrate a regression-based size estimation model. Forward stepwise MLR (Multiple Linear Regression) is used for model fitting. The model thus obtained is assessed using a variety of accuracy measures such as MMRE (Mean Magnitude of Relative Error), Prediction of x(PRED(x)), MdMRE (Median of Relative Error) and validated using K-fold cross validation. The accuracy of this model is also compared with an existing model tailored for size estimation of board games. Based on a small subset of desktop games developed in various object-oriented languages, we obtained a model using CK metrics and forward stepwise multiple linear regression with reasonable estimation accuracy as indicated by the value of the coefficient of determination ($R^2 = 0.756$).Comparison results indicate that the existing size estimation model outperforms the model derived using CK metrics in terms of accuracy of prediction.

Key Words: **Accuracy Measures, Chidamber and Kemerer Metrics, Game Sizing, Multiple Linear Regression, Open Source, Project Management, Project Planning, Simple Linear Regression, Software Size Estimation, K-fold Cross Validation**

1. INTRODUCTION

A number of attributes can be used to describe and judge various aspects of software. Some of these attributes (e.g. quality) are external while others (e.g. complexity) are internal. Software size is classified as an internal attribute [1]. Once the software has been developed, its size can be easily measured by using any of a number of different applications called code counters [2]. The real challenge however, is

Corresponding Author (E-Mail: nosheen.sabahat@ceme.nust.edu.pk)
* Department of Computer Engineering, National University of Sciences & Technology, Islamabad.
** Department of Computer Science, National University of Computer & Emerging Sciences, Lahore.

estimating the size of software before its development. Unfortunately, software size is unavailable during early phases of software development. So we need to develop a size estimation model that accurately predicts the size of desktop-based software games.

Estimation of software size is challenging but the rewards for accurate prediction of size are extremely high. This is primarily because software size is one of the key inputs of many effort and cost estimation models [3]. Some of these models, such as the widely used COCOMO-II (Constructive Cost Model) [4], use both direct size measures e.g. SLOC (Source Lines of Code) [5]) as well as indirect functional size measures such as function points [6]. An accurate estimate of either SLOC or the programming-language independent function points is, therefore, crucial in planning and managing software development projects.

Software development projects using the object-oriented paradigm [7] use class diagrams [8] for modeling purposes. Analysis class diagrams are used to model the problem space while design class diagrams model the solution space [9]. Apart from being used as a tool for improving understanding of the problem and solution spaces, these class diagrams can also be used as a tool for estimating the size of the final software application [10].

Application design information, whether obtained from the class diagrams or extracted by reverse engineering [11] code written in an object-oriented programming language, is a good candidate for being used as an input to a size estimation model. This is primarily because design is the last phase before implementation. Therefore, a wealth of information is available at the end of the design phase. This research attempts to take advantage of this fact.

In this work, we describe the derivation and validation of a regression-based model developed to predict the size of board-based software games. This model uses the design information provided by CK metrics [12]. It is a suit of most validated and most reliable metrics. It is used to evaluate object-oriented design [13] as input and generates an estimate of software size as output. A dataset comprising around 60 open source board-based software games is used to calibrate this model. Once calibrated, this model is assessed using different accuracy measures and is validated using K-fold cross validation [14]. The accuracy of this model is also compared to that of an existing size estimation model.

The next section discusses relevant past work in this area. Section-III presents an overview of the CK metrics and section-IV illustrates their calculation via a small case study. Our research methodology is described in detail in section-V which also summarizes the results of model assessment and validation and presents a comparison withan existing size estimation model. Threats to validity of this research are discussed in section-VI. Section-VII highlights our major contributions and finally section-VIII proposes some directions for further work in this area.

2. LITERATURE REVIEW

Researchers have explored different ways of estimating software size using information contained in the class diagrams of object-oriented systems. One of the earliest works in this area was done by Mišic and Tešic [15]. They used the number of classes and the number of methods of classes in the class diagram to predict the size of a software system via OLS (Ordinary Least Squares) regression. Since, we do not use number of methods and number of classes for size estimation, we only adapted OLS regression technique.

More recently, Harizi [16] also proposed a software size estimation model that relied on various parameters of a class diagram. However, along with such parameters, we support a more comprehensive set of well-known suite of object-oriented CK metrics, in estimating the size of software applications using the information from its UML class diagram. Zhou et. al. [10] looked at the accuracy of code size estimation approaches empirically using the information contained in UML class diagrams to predict the size of object-oriented systems. Different modeling techniques were used to check the accuracy of models and it was finally concluded that object-oriented metrics can be used to estimate the size of software. However, their findings are based on Java systems only. Whereas our work validates the findings using systems developed in various object-oriented programming languages such as C#, C++, Python and Java etc.

A slightly different approach was adopted by Costagliola et. al. [17]. This approach used class points - a functional size measure similar to function points but specifically designed to estimate the size of object-oriented applications. First unadjusted and then adjusted class points were calculated. The size of the software was predicted on the basis of these class points. Since calculation of class points do not add any significance to this work, we did not use them.

Another attempt in this area was made by Tan et. al. [18]. They proposed a size estimation method for information systems. Multiple linear regression was used to obtain a size estimation model that relied on information present in the class diagrams (such as relationships/associations between classes, attributes of classes) to feed the independent variables (i.e. inputs). It was concluded that using class diagram metrics provides better results in predicting the size of the data intensive systems. However, this approach attempts to estimate the size of information systems only which the authors refer to as database application that supports business processes. Moreover, it relies only on the information of association and attributes of classes from UML class diagram.

Various other metrics have also been used to estimate the size of an object-oriented system such as class diagram related metrics [19], object-oriented function points [20], fast&&serious class points [21] etc. All above mentioned approaches use various class diagram related metrics only to estimate the size of software. The assessment of model accuracy and validation of most size estimation approaches have not been carried out. Secondly the data set is limited to few systems hence statistical conclusions were hard to determine.

This study focuses on the size estimation of object-oriented board-based software games using the design information provided by CK metrics. These well-known metrics are used to build a regression-based model which is assessed for accuracy, validated using K-fold cross validation, and compared to an existing size estimation model.

3. OVERVIEW OF CK METRICS SUITE

Chidamber et. al. [22] proposed a set of metrics in 1991 specifically for object-oriented software programs. It was improved in 1994 [12] and is now used widely. This suite consists of 6 metrics viz. WMC (Weighted Methods/ Class), DIT (Depth of Inheritance Tree), NOC (Number of Children), CBO (Coupling Between Objects), RFC (Response For a Class) and LCOM (Lack of Cohesion in Methods). The values of these metrics are determined for each class in anobject-oriented software. Brief descriptions of these metrics [12] are given below:

Weighted Methods Per Class: In its simplest form, WMC is a count of all methods in a class. It provides an indication of the complexity of a class.

Depth of Inheritance Tree: DIT refers to the “depth of inheritance of the class”. In case of multiple inheritance, DIT is “the maximum length from the node to the root of the tree". As a class gets deeper in the hierarchy, it inherits more methods and variables.

Number of Children: NOC refers to the "number of immediate subclasses subordinated to a class in a class hierarchy" [18]. Unlike DIT, it measures the breadth of the class hierarchy.

Coupling between Object Classes: CBO for a class is defined as "a count of the number of other classes to which it is coupled". Two classes are said to be coupled "when methods declared in one class use methods or instance variables defined by the other class".

Response for a Class: RFC refers to "a set of methods that can potentially be executed in response to a message received by an object of that class".

Lack of Cohesion in Methods: LCOM"provides a measure of the relative disparate nature of methodsin the class". It indicates how closely local methods are related to local instance variables in a class. In other words, it is a number of disjoint set of local methods [23].

4. CASE STUDY

Consider a board game called "X and O". The objective of this two-player game is to win by getting three X's or O's in a row. This game is played on a three by three game board. The players take turns in placing X's and O's on the board till one of them wins or the board gets filled. The rules of the game are listed below:

- Game Start:
  - Any player can start the game.
- Moves:
  - The players choose their signs as X or O and then move accordingly.
  - A player can place X or O at any empty place of the grid on his turn.
- Draw:
  - The game is said to be drawn if none of the player succeeds in getting three in a row and the board gets filled.
- Win:
  - The game is won by the player who gets three in a row first.

A software version of this game was developed using Java (row 57 of Table 2) and made available on an open source repository. Fig. 1 shows the design class diagram of the software version of this game. This diagram was reverse engineered from the source code of this game using the Understand tool [24]. The same tool was also used to extract the values of the CK metrics for the design of this game. Table 1 depicts the values of CK metrics for each of the design classes shown in Fig. 1. The last row of this table sums up the values of each CK metric for all individual classes. These sums are then used as inputs for our size estimation model.

5. RESEARCH METHODOLOGY

This research extends our earlier work on the size estimation of board-based software games [25,26]. In our earlier work, we defined eight potential predictors to estimate the size of desktop-based board games viz. NRUL (Number of Rules), NPLY (Number of Players), ANIM (Animation), 3DVI (3D Visualization), COPP (Computer Opponent), MSKI (Multi Skills), NTVA (Number of Type of Variants) and MGOP (Miscellaneous Game Options) as input of our model. In this work, we present the derivation, assessment, and validation of a new size estimation model based on CK metrics. The accuracy of this new design-based model is compared with the accuracy of an early size estimation model proposed in one of our earlier works [25]. Fig. 2 illustrates our research methodology. It depicts the major steps carried out in this research.

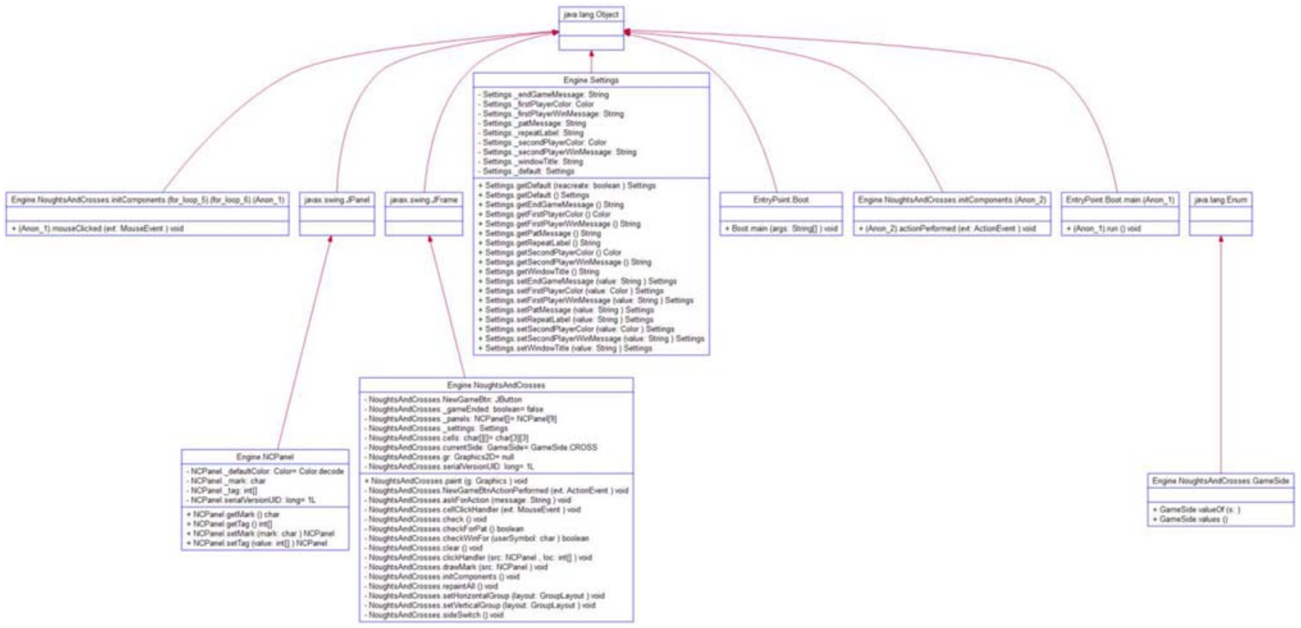
java lang Object
Engine Settings
- Settings _endGameMessage: String
- Settings _firstPlayerColor: Color
- Settings _firstPlayerWinMessage: String
- Settings _patMessage: String
- Settings _repeatLabel: String
- Settings _secondPlayerColor: Color
- Settings _secondPlayerWinMessage: String
- Settings _windowTitle: String
- Settings _default: Settings
+ Settings getDefault (reacreate: boolean) Settings
+ Settings getDefault () Settings
+ Settings getEndGameMessage () String
+ Settings getFirstPlayerColor () Color
+ Settings getFirstPlayerWinMessage () String
+ Settings getPatMessage () String
+ Settings getRepeatLabel () String
+ Settings getSecondPlayerColor () Color
+ Settings getSecondPlayerWinMessage () String
+ Settings getWindowTitle () String
+ Settings setEndGameMessage (value: String) Settings
+ Settings setFirstPlayerColor (value: Color) Settings
+ Settings setFirstPlayerWinMessage (value: String) Settings
+ Settings setPatMessage (value: String) Settings
+ Settings setRepeatLabel (value: String) Settings
+ Settings setSecondPlayerColor (value: Color) Settings
+ Settings setSecondPlayerWinMessage (value: String) Settings
+ Settings setWindowTitle (value: String) Settings
Engine NoughtsAndCrosses initComponents (for_loop_5) (for_loop_6) (Anon_1)
+ (Anon_1) mouseClicked (evt: MouseEvent) void
javax swing JPanel
javax swing JFrame
EntryPoint Boot
+ Boot main (args: String[]) void
Engine NoughtsAndCrosses initComponents (Anon_2)
+ (Anon_2) actionPerformed (evt: ActionEvent) void
EntryPoint Boot main (Anon_1)
+ (Anon_1) run () void
java lang Enum
Engine NoughtsAndCrosses
- NoughtsAndCrosses NewGameBtn: JButton
- NoughtsAndCrosses _gameEnded: boolean= false
- NoughtsAndCrosses _panels: NCPanel[]= NCPanel[9]
- NoughtsAndCrosses _settings: Settings
- NoughtsAndCrosses cells: char[][]= char[3][3]
- NoughtsAndCrosses currentSide: GameSide= GameSide CROSS
- NoughtsAndCrosses gr: Graphics2D= null
- NoughtsAndCrosses serialVersionUID: long= 1L
+ NoughtsAndCrosses paint (g: Graphics) void
- NoughtsAndCrosses NewGameBtnActionPerformed (evt: ActionEvent) void
- NoughtsAndCrosses askForAction (message: String) void
- NoughtsAndCrosses cellClickHandler (evt: MouseEvent) void
- NoughtsAndCrosses check () void
- NoughtsAndCrosses checkForPat () boolean
- NoughtsAndCrosses checkWinFor (userSymbol: char) boolean
- NoughtsAndCrosses clear () void
- NoughtsAndCrosses clickHandler (src: NCPanel , loc: int[]) void
- NoughtsAndCrosses drawMark (src: NCPanel) void
- NoughtsAndCrosses initComponents () void
- NoughtsAndCrosses repaintAll () void
- NoughtsAndCrosses setHorizontalGroup (layout: GroupLayout) void
- NoughtsAndCrosses setVerticalGroup (layout: GroupLayout) void
- NoughtsAndCrosses sideSwitch () void
Engine NCPanel
- NCPanel _defaultColor: Color= Color decode
- NCPanel _mark: char
- NCPanel _tag: int[]
- NCPanel serialVersionUID: long= 1L
+ NCPanel getMark () char
+ NCPanel getTag () int[]
+ NCPanel setMark (mark: char) NCPanel
+ NCPanel setTag (value: int[]) NCPanel
Engine NoughtsAndCrosses GameSide
+ GameSide valueOf (s:)
+ GameSide values ()

FIG. 1. DESIGN CLASS DIAGRAM OF THE GAME "X AND O" USING 'UNDERSTAND'TOOL [24]

TABLE 1. CK METRICS FOR THE GAME "X AND O"

Classes	LCOM	DIT	CBO	NOC	RFC	WMC
Engine.Panel	75	2	0	0	5	5
Engine.Noughts and Crosses	82	2	3	0	16	16
Engine.Noughts and Crosses GameSide	100	1	1	0	2	2
Engine.Settings	89	1	0	0	18	18
EntryPoint.Boot	0	1	0	0	1	1
EntryPoint.Boot.main.(Anon_1)	0	1	2	0	1	1
Engine.NoughtsAndCrosses.initComponents.(for_loop_5).(for_loop_6).(Anon_1)	0	1	1	0	1	1
Engine.NoughtsAndCrosses.initComponents(Anon_2)	0	1	1	0	1	1
Total	346	10	8	0	45	45

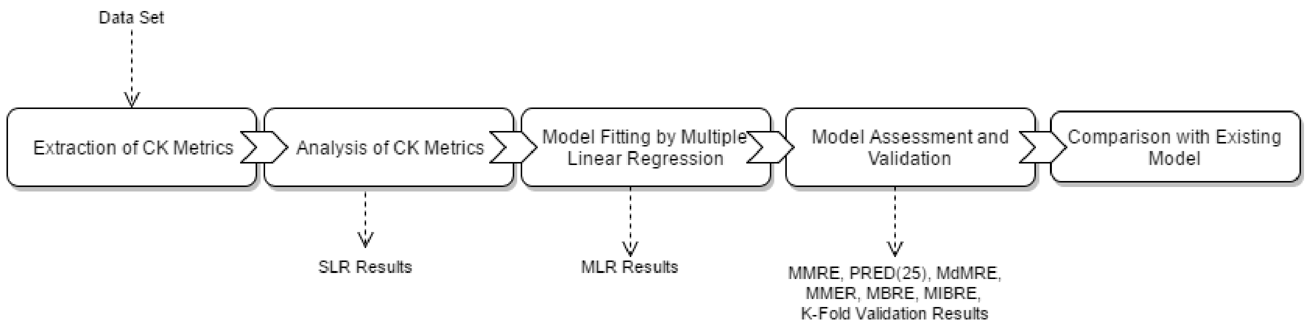
Data Set
Extraction of CK Metrics
Analysis of CK Metrics
Model Fitting by Multiple Linear Regression
Model Assessment and Validation
Comparison with Existing Model
SLR Results
MLR Results
MMRE, PRED(25), MdMRE, MMER, MBRE, MIBRE, K-Fold Validation Results

FIG. 2. BASIC STRUCTURE OF METHODOLOGY

5.1 Extraction of CK Metrics

The same dataset (comprising 67 board-based games) that was used for calibrating our previous model [25] was used as a starting point. However, games that were not developed using an object-oriented programming language were pruned from the original dataset. As shown in Table 2, this pruning resulted in a slightly smaller subset comprising 59 games all of which were developed using an object-oriented programming language.

The values of the six CK metrics (i.e. LCOM, DIT, CBO, NOC, RFC, and WMC) for each of these games were extracted using a commercial static analysis tool called Understand [24]. This tool provides the values of the six CK metrics for each class. The values for the entire game are obtained by summing up the values of each class used in the game. SLOC (Source Lines of Code) for each game were counted by using USC-CSSE Unified Code Count tool [27]. From Table 2 we can see that most of the games have single SLOC whereas few games have multiple SLOC. This is because those games are developed in more than one language and hence all those languages contribute to the FP (Function Points).

QSM's language-specific GF (Gearing Factors) [28] are used for converting the SLOC to the programming-language independent FP. The average values of the language-specific GFs provided in the QSM FP Languages Table 2 [29] are used for this conversion. Similarly we have mentioned GF for multiple languages in the table. Using multiple GF we converted multiple SLOC into FP.

5.2 Analysis of CK Metrics

First, SLR [30] was used to determine the predictive strength of each CK metric individually. Later, forward stepwise MLR [31] was used to obtain our size estimation model which uses FP as the response variable and the six CK metrics as the six predictor variables. Regression analyses were done using the IBM SPSS tool [32].

Table 3 shows the results of SLR. It lists the predictors in descending order of predictive strength. As is evident from these results, WMC is the strongest individual predictor of software size (FP) with R^2 value of 0.709 whereas DIT is the weakest. The size estimation model obtained using MLR is presented in the next subsection.

5.3 Model Fitting by MLR

Before building the size estimation model, outliers had to be eliminated. Cook's distance [33] was used for this purpose. Four games were identified as outliers and hence eliminated from the dataset since their Cook's distance was greater than ((4/n)*3) where n represents the total number of games.

The data set consisting of 55 games now (after eliminating the four outliers) was used to build the final size estimation model. Forward stepwise MLR was used for model fitting. This technique is helpful in discovering the set of significant predictor variables which best explains the relationship between the response variable (FP) and the independent/predictor variables (LCOM, NOC, DIT, RFC, CBO, and WMC).

In this technique, the model is gradually built by adding one independent variable at a time. At each step, the value of coefficient of determination (R^2) is assessed to check if the recently added variable increased the value of R^2 or not. Besides this, only those combinations of predictors are selected in which all predictors are statistically significant at ±-value of 0.05. Moreover, the presence of multicollinearity among the predictors is also checked using the value of VIF (Variance Inflation Factor) [34]. In our case, all of the predictors in the MLR models obtained had VIF values close to 1. The presence of multicollinearity was, therefore, ruled out.

TABLE 2. DATA SET

No.	Games	Language	SLOC	GF	FP	LCOM	DIT	CBO	NOC	RFC	WMC
1.	3D Tic Tac Toe	C#	2851	54	52.796	2310	15	475	2	782	380
2.	C# Tic Tac Toe	C#	531	54	9.833	91	1	20	0	50	32
3.	Chess Bin	C#	2312	54	42.815	641	4	134	0	275	149
4.	Clu	C#	8388	54	155.333	1554	6	417	0	1001	659
5.	Connect 4 ab	C#	521	54	9.648	168	1	17	0	43	25
6.	Dot Chess	C#	3155	54	58.426	1620	27	503	12	1069	363
7.	Draughts Master	C#	1196	54	22.148	831	15	152	3	400	174
8.	DraughtsDotNet	C#/JS/ASP	12353/3174/73	54/47/51	297.723	8355	289	1665	113	5519	1677
9.	Four in a Row	C#	659	54	12.204	403	2	53	0	165	75
10.	Grid	C#	3004	54	55.63	286	1	102	13	203	110
11.	Learn Connect 4	C#	3729	54	69.056	991	10	155	7	674	321
12.	LudoDP	C#	2103	54	38.944	577	2	85	3	204	105
13.	Ludo.net	C#	11355	54	55.056	10032	175	1930	46	4915	2065
14.	Multi Revrsi	C#	1843	54	34.13	92	1	26	0	41	32
15.	Othello Reversi	C#	2302	54	42.63	1549	17	243	10	664	290
16.	Quixo 3D	C#	3227	54	59.759	1772	56	495	19	1095	443
17.	RAE TTT	C#	825	54	15.278	249	9	79	2	200	76
18.	Reversi	C#	2464	54	45.63	612	6	131	0	200	110
19.	Reversi 2008	C#	1434	54	26.556	1063	9	160	2	317	148
20.	Scrabble for wpf	C#	3170	54	58.704	3588	39	648	26	1495	652
21.	Scrabble Nights	C#	2976	54	55.111	1604	12	358	2	543	288
22.	Sharp Checkers	C#	1588	54	29.407	1011	3	259	7	467	224
23.	Snake board game	C#	1442	54	26.704	179	3	60	0	92	56
24.	Sr Chess for WPF	C#	8318	54	154.037	2625	30	776	9	1388	716
25.	Stoopid Chess	C#	2796	54	51.778	1026	5	209	1	336	192
26.	TTT Network	C#	1444	54	26.741	257	2	66	0	88	52
27.	Blockem	C++	8811	50	176.22	1705	12	154	4	518	449
28.	Brutal Chess	C++	5987	50	119.74	1849	18	123	15	673	438
29.	Ethereal Chess	C++	14771	50	295.42	1643	0	59	0	816	816
30.	Gamebit (Chess)	C++	6463	50	129.26	2173	35	317	22	782	558
31.	Hotel	C++/C#	5452/7323	50/54	244.651	2913	31	546	13	1929	1009
32.	Maxit	C++	674	50	13.48	387	3	12	0	55	54
33.	Nero Othello	C++	3881	50	77.62	830	16	22	5	469	328
34.	Pentobi	C++/Python	17721/316	50/24	367.587	5662	86	561	54	2178	1526
35.	Piskvork	C++/Pascal	6670/124	50/91	134.763	30	0	0	0	5	5
36.	Pouet Chess	C++/Python	14272/352	50/24	300.287	3934	56	404	43	3602	1393
37.	Snakes &Ladders	C++	621	50	12.429	307	6	8	0	122	101
38.	The Genius	C++	3473	50	69.46	638	8	21	0	156	140
39.	Tsuro	C++	8002	50	160.04	4167	110	458	75	4187	1086
40.	Win Chess	C++	9014	50	180.28	900	0	57	0	52	52
41.	Capa Chess	Java	5498	53	103.736	1172	31	154	7	189	175
42.	Cluedo	Java	11188	53	211.094	5069	194	556	28	1150	981
43.	Domination	Java	6323	53	119.302	2125	64	142	7	788	594
44.	Dots and Boxes	Java	879	53	16.585	630	20	26	0	96	96
45.	EBF Checkers	Java	5371	53	101.34	2505	120	191	15	804	574
46.	Get Four	Java	515	53	9.717	304	25	38	0	70	70
47.	GoBang (5 in row)	Java	1863	53	35.151	620	39	67	1	184	160
48.	Java Checkers	Java	695	53	13.113	341	11	14	0	88	88
49.	Java Frittle Chess	Java	1170	53	22.075	1241	47	79	4	258	200
50.	Java Tic Tac Toe	Java	467	53	8.811	92	21	19	0	49	49
51.	JO Chess	Java	3886	53	73.321	2858	83	198	6	357	297
52.	Jscrabble	Java	4895	53	92.358	3715	161	335	20	694	597
53.	KludgopolB	Java	14461	53	272.849	5407	260	704	43	1720	1138
54.	Scotland Yard	Java	1171	53	22.094	653	19	36	2	115	105
55.	Two Player Chess	Java	1525	53	28.774	1283	37	94	6	306	204
56.	Tyrannus (chess)	Java	3013	53	56.849	1581	58	129	0	298	298
57.	X and O	Java	267	53	5.038	346	10	8	0	45	45
58.	Scrabble 3D	Pascal	14326	91	157.429	3689	104	119	0	857	659
59.	GlParchis (Ludo)	C++/Python	4667/2747	50/24	208.478	1375	28	40	11	686	481

Table 4 summarizes the results of applying forward stepwise MLR on the data set. Each row in this table represents an MLR model. A check mark (✓) represents inclusion while a dash (-) indicates exclusion of the corresponding predictor variable. Using different combinations of predictors, a total of six models were obtained in which all predictors were statistically significant at α-value of 0.05. It is important to note that all of these six models had only two predictor variables. This is because none of the MLR models obtained using more than two predictors had all statistically significant predictors (at α-value of 0.05). The best model (highlighted in bold in Table 4) has an R^2 value of 0.756. The two predictors it uses are CBO and WMC. The following Equation (1), in which EST_FP (Estimated Function Points) denotes the estimated value of the response variable (FP), formalizes this model:

TABLE 3. SLR RESULTS USING CK METRICS

No.	Predictors	R2
1.	WMC	0.709
2.	RFC	0.508
3.	LCOM	0.460
4.	NOC	0.332
5.	CBO	0.210
6.	DIT	0.196

5.4 Model Assessment and Validation

The accuracy of an estimation modelmust be formally assessed using different accuracy measures. In this research, we have used the two de-facto standards i.e. MMRE [35] and PRED(x) [35-36]. MMRE is defined as the average of all MREs calculated as a result of regression. On the other hand, PRED(x) is defined as, “the percentage of relative error deviation that lies within x”. The most commonly used values of x in literature are 25 and 30 with little difference in results [37]. We use PRED(25) in our work (i.e. x = 25) which is a more stringent metric than PRED(30). According to Conte et. al. [37] for a model to have reasonably good estimation accuracy, MMRE $\leq$ 0.25 and PRED(25) $\geq$ 0.75.

Despite the fact that MMRE and PRED(x) are the de-facto standards of estimation accuracy, they have been criticized due to their limitations [38]. Therefore, to overcome these limitations, we have used additional accuracy metrics as well i.e. MdMRE, MMER (Mean Magnitude of Error Relative) to estimate, MBRE (Mean of Balanced Relative Error), and MIBRE (Mean of Inverted Balanced Relative Error).

TABLE 4. MLR RESULTS USING CK METRICS

No.	LCOM	DIT	CBO	NOC	RFC	WMC	R^2	VIF
1.	-	-	✓	-	-	✓	**0.756**	CBO = 2.091
								WMC = 2.091
2.	✓	✓	-	-	-	-	0.501	LCOM = 3.094
								DIT = 3.094
3.	✓	-	-	-	✓	-	0.550	LCOM = 2.429
								RFC = 2.429
4.	-	✓	✓	-	-	✓	0.732	LCOM = 4.715
								WMC = 4.715
5.	-	✓	✓	-	-	-	0.270	DIT = 1.336
								CBO = 1.336
6.	-	-	-	-	✓	✓	0.733	RFC = 6.427
								MC = 6.427

Table 5 summarizes the values of these different accuracy metrics for our CK metrics-based size estimation model. The formulas for calculating these accuracy metricsare also given in Table 5. The values of these accuracy metrics clearly indicate that this size estimation model does not have a reasonably good estimation accuracy. MMRE and other error-related means are more than 0.25.Similarly, PRED(25) is much less than 0.75.

TABLE 5. VALUES OF ACCURACY METRICS

No.	Metric	Based On	Formula	Value
1.	MMRE	$MRE_i = \frac{\lvert Y_i - \hat{Y}_i \rvert}{Yi}$	$MMRE = \frac{1}{N}\sum_{i=1}^{N} MRE_i$	0.69
2.	MdMRE	$MRE_i = \frac{\lvert Y_i - \hat{Y}_i \rvert}{Yi}$	MdMRE = MEDIAN (i=1, N) [MREi]	0.32
3.	MMER	$MRE_i = \frac{\lvert Y_i - \hat{Y}_i \rvert}{\hat{Y}_i}$	$MMER = \frac{1}{N}\sum_{i=1}^{N} MER_i$	0.56
4.	MBRE	$BRE_i = \frac{\lvert \hat{Y}_i - Y_i \rvert}{\hat{Y}_i}, \hat{Y}_i - Y_i < 0$ $BRE_i = \frac{\lvert \hat{Y}_i - Y_i \rvert}{Y_i}, \hat{Y}_i - Y_i \geq 0$	$MBRE = \frac{1}{N}\sum_{i=1}^{N} BRE_i$	0.34
5.	MIBRE	$IBRE_i = \frac{\lvert \hat{Y}_i - Y_i \rvert}{Y_i}, \hat{Y}_i - Yi < 0$ $IBRE_i = \frac{\lvert \hat{Y}_i - Y_i \rvert}{\hat{Y}_i}, \hat{y}_i - y_i \geq 0$	$MIBRE = \frac{1}{N}\sum_{i=1}^{N} IBRE_i$	0.91
6.	PRED(25)	$MRE_i = \frac{\lvert Y_i - \hat{Y}_i \rvert}{Yi}$	$PRED(x) = \frac{1}{N}\sum_{i=1}^{N} \begin{cases} 1 \text{if } MRE_i \geq x \\ 0 \text{ Otherwise} \end{cases}$	0.36

TABLE 6. K-FOLD CROSS VALIDATION RESULTS

K	Data Points	Size Estimation Model	R2	MMRE	MdMRE	MMER	MBRE	MIBRE	PRED (0.25)
1.	01-11	17.6531 + (0.2676 * WMC) + (-0.1251 * CBO)	0.73	0.18	0.12	0.19	0.18	0.15	0.63
2.	12-22	25.3752 + (0.2582 * WMC) + (-0.1264 * CBO)	0.73	1.34	1.30	0.52	1.34	0.52	0.09
3.	23-33	26.4727 + (0.2508 * WMC) + (-0.1370 * CBO)	0.74	1.84	1.32	0.49	1.84	0.49	0.27
4.	34-44	14.3837 + (0.2948 * WMC) + (-0.0816 * CBO)	0.78	0.53	0.36	0.47	0.53	0.34	0.18
5.	45-55	8.7118 + (0.2676 * WMC) + (-0.1508 * CBO)	0.78	0.35	0.19	2.26	0.35	0.32	0.54
Average				0.84	0.65	0.78	0.84	0.36	0.34

The terms Y and $\hat{Y}_i$ in Table 5, denote the actual and the predicted values respectively obtained after regression.

K-fold cross validation [14] was used to validate this model. The data set comprising 55 games was divided into 5 random subsets (i.e. K = 5), each subset comprising 11 games placed randomly. In each iteration, 4 folds were used to train the model and the remaining 5th fold was used to validate the model. Table 6 summarizes the results of this K-fold cross validation exercise. The last row of Table 6 shows the average vales of accuracy metrics. As indicated by these average values, the performance of the model is not up to the mark.

5.5 Comparison of Results

In this subsection, the accuracy of the design-based size estimation model obtained using CK metrics is compared with that of an early size estimation model proposed in one of our earlier works [25]. The early size estimation model uses four inputs (i.e. number of game rules, number of players, animation, and miscellaneous game options) and produces an estimate of FP as output. Table 7 shows a comparison of the accuracy of these two models by depicting values of different accuracy metrics. This comparison clearly indicates that the model built using CK metrics has much lower accuracy than the early size estimation model proposed earlier. The early size estimation model has higher values of R^2 and PRED(25) and lower values for MMRE and other error-related means.

A comparison of the results of K-fold cross validation shown in Table 8 also corroborates the same conclusion. The early size estimation model clearly outperforms the design-based size estimation model built using CK metrics. This may appear counterintuitive since more information is present at the time of design. It must be kept in mind, however, that these two models use different predictors. It is quite possible that the predictors of the early size estimation model (for instance, number of rules) capture crucial information about the requirements which may have been missed by the CK metrics.

6. THREATS TO VALIDITY

A couple of factors may have an impact on the validity of our results. The first of these is related to the comparison of the size estimation model obtained using CK metrics and the size estimation model proposed earlier. The earlier size estimation model was calibrated using 65 open source board-based games. Since all of these games were not

TABLE 7. COMPARISON OF ACCURACY METRICS

No.	Metric	CK Metrics Model	EarlySize Estimation Model [25]
1.	R2	0.75	0.91
2.	MMRE	0.69	0.24
3.	MdMRE	0.32	0.18
4.	MMER	0.56	0.22
5.	MBRE	0.34	0.32
6.	MIBRE	0.91	0.21
7.	PRED(25)	0.36	0.75

TABLE 8. COMPARISON OF K-FOLD CROSS VALIDATION RESULTS

No.	Metric	CK Metrics Model	EarlySize Estimation Model [25]
1.	MMRE	0.84	0.23
2.	MdMRE	0.65	0.18
3.	MMER	0.78	0.25
4.	MBRE	0.84	0.34
5.	MIBRE	0.36	0.21
6.	PRED(25)	0.34	0.71

developed using object-oriented languages, the exact same dataset could not be used for building a size estimation model based on the CK metrics. A slightly smaller subset comprising games developed using only object-oriented programming languages was used instead.

The second factor is related to measuring the values of CK metrics. In this research, we have used a single static analysis tool i.e. Understand [24]. Since different static analysis tools may use slightly different conventions for calculating the values of the CK metrics, a change of tool may lead to a slight variance in results.

7. CONCLUSION

This paper has presented the results of an investigation conducted to determine the utility of CKmetrics in estimatingsoftware size. A dataset comprising around 60 open source board-based desktop games was used as a starting point. The values of CK metrics of each game were extracted using a commercial static analysis tool. These values were then used for model fitting using forward stepwise multiple linear regression. The fitted model was assessed via commonly used accuracy parameters and validated using K-fold cross validation. Assessment and validation results indicate that the accuracy of this model does not meet the recommended standards suggesting that CK metrics alone may have little utility in estimating software size.

The prediction accuracy of this model was also compared to that of an existing size estimation model developed specifically for estimating the size of board-based games. The existing model relies on information available at the time of inception while CK metrics, on the other hand, are available only after the completion of detailed design. Therefore, even though one may expect the model using CK metrics to be more accurate, comparison results indicate that the existing model outperforms the model based on CK metrics with respect to prediction accuracy. This unexpected behavior is, perhaps, the result of not capturing crucial game requirements such as the number of rules of the game.

8. FUTURE WORK

This work can be extended in a variety of ways. So far, we have focused on a small subset dealing with just board-based desktop games. Other types of games (e.g. card-based) may also be considered. Similarly, games made for other platforms (e.g. mobile) may also be explored. Thirdly, the utility of other suites of object-oriented metrics may be investigated. Last but not the least, a hybrid-model using a combination of object-oriented design-based software metrics and requirements-based metrics may be built to get the best of both worlds.

ACKNOWLEDGEMENT

Authors are thankful to National University of Sciences & Technology, Islamabad, Pakistan, and FAST-National University of Computer & Emerging Sciences, Lahore, Pakistan, for their joint collaboration in the research work.

REFERENCES

[1] Fenton, N.E., and Pfleeger, S.L., “Software Metrics: A Rigorous and Practical Approach”, 2nd Edition, PWS Publishing Co., pp. 435-445, Boston USA, 1998.

[2] QSM Code Counters, http://www.qsm.com/resources/code-counters/, [Last Accessed: 30th December 2016].

[3] Pfleeger, S.L., Wu, F., and Lewis, R., “Software Cost Estimation and Sizing Methods: Issues and Guidelines”, RAND Corporation, pp. 1-7, California USA, 2005.

[4] Boehm, B.W., Abts, C., Brown, A.W., Chulani, S., Clark, B.K., Horowitz, E., Madachy, R., Reifer, D.J., and Steece, B., “Software Cost Estimation with COCOMO-II”, Prentice Hall, New Jersey USA, 2000.

[5] Laird, L.M., and Brennan, M.C., “Software Measurement and Estimation: A Practical Approach”, John Wiley & Sons, New Jersey, USA, 2006.

[6] Albrecht, A.J., “Measuring Application Development”, Proceedings of the IBM Applications Development Joint SHARE/ GUIDE Symposium, pp. 83–92, California, USA, 1979.

[7] Meyer, B., “Object-Oriented Software Construction”, Prentice Hall, Volume. 2, pp. 331-410. New York USA, 1988.

[8] Booch, G., ”Object-Oriented Analysis and Design”, Addison-Wesley, California USA, 1980.

[9] Blaha, M., and Rumbaugh, J., “Object-Oriented Modeling and Design with UML”, Pearson Education, Upper Saddle River, USA, 2005.

[10] Zhou, Y., Yang, Y., Xu, B., Leung, H. and Zhou, X., "Source Code Size Estimation Approaches for Object-Oriented Systems from UML Class Diagrams: A Comparative Study", Information and Software Technology, Volume 56, No. 2, pp. 220-237, 2014.

[11] Chikofsky, E.J. and Cross, J.H., "Reverse Engineering and Design Recovery: A Taxonomy", IEEE Software, Volume 7, No. 1, pp. 13-17, 1990.

[12] Chidamber, S.R. and Kemerer, C.F., "A Metrics Suite for Object-Oriented Design", IEEE Transactions on Software Engineering, Volume 20, No. 6, pp. 476–493, 1994.

[13] Chidamber, S.R., Darcy, D.P. and Kemerer, C.F., "Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis", IEEE Transactions on Software Engineering, Volume 24, No. 8, pp. 629-639, 1998.

[14] Picard, R.R., Cook, R.D., "Cross-Validation of Regression Models", Journal of the American Statistical Association, Volume 79, No. 387, pp. 575-83, 1984.

[15] Mišic, V.B., Tešic, D.N., "Estimation of Effort and Complexity: An Object-Oriented Case Study", Journal of Systems and Software, Volume 41, No. 2, pp. 133-143, 1998.

[16] Harizi, M., "The Role of Class Diagram in Estimating Software Size", International Journal of Computer Applications, Volume 44, No. 5, pp. 31-33, 2012.

[17] Costagliola, G., Ferrucci, F., Tortora, G., Vitiello, G., "Class point: An Approach for the Size Estimation of Object-Oriented Systems", IEEE Transactions on Software Engineering, Volume 31, No. 1, pp.52-74, 2005.

[18] Tan, H.B.K., Zhao, Y. and Zhang, H., "Conceptual Data Model-based Software Size Estimation for Information Systems", ACM Transactions on Software Engineering and Methodology, Volume 19, No. 2, pp. 1-37, 2009.

[19] Antoniol, G., Lokan, C., Fiutem, R., "Object-Oriented Function Points: An Empirical Validation", Empirical Software Engineering, Volume 8, No. 3, pp. 225-254, 2003.

[20] Antoniol, G., Lokan, C., Caldiera, G., and Fiutem, R., "A Function Point-like Measure for Object-Oriented Software", Empirical Software Engineering, Volume 4, No. 3, pp. 263-287, 1999.

[21] Carbone, M., Santucci, G., "Fast&&Serious: A UML Based Metric for Effort Estimation",Proceedings of the 6th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, pp. 35-44, 2002.

[22] Chidamber, S.R. and Kemerer, C.F., "Towards a Metrics Suite for Object-Oriented Design", Proceedings of 6th OOPSLA Conference, ACM, pp. 197-211, 1991.

[23] Bakar, N.S., "The Analysis of Object-Oriented Metrics in C++ Programs". Lecture Notes on Software Engineering, Volume 4, No. 1, pp. 48, 2016.

[24] Understand tool, https://www. scitools.com/, [Last Accessed: 25thDecember 2016].

[25] Sabahat, N., Malik, A.A., Azam, F., "A Size Estimation Model for Desktop Games", Submitted to IEEE Access, 2017.

[26] Sabahat, N., Malik, A.A., Azam, F., "Size Estimation of Open Source Board-based Software Games", Presented at the 9th IEEE International Conference on Open Source Systems and Technologies, 2015.

[27] UCC, Unified Code Counter, http://sunset.usc.edu/ucc_wp/ , [Last Accessed: 30thOctober 2016].

[28] QSM Gearing Factors, "A Flexible Sizing Approach, Quantitative Software Management. Inc. http://www.qsm.com/Blog/Gearing%20Factors.pdf, [Last Accessed: 30thDecember 2016].

[29] QSM, Function Point Languages Table Version 4.0, Quantitative Software Management, http://www.qsm.com/resources/function-pointlanguages-table/index.html, [Last Accessed: 30th December 2016].

[30] Weisberg, S., "Applied Linear Regression", Wiley, 1980.

[31] Fritz, M., and Berger, P.D., "Improving the User Experience Through Practical Data Analytics", Morgan Kaufmann, pp. 239-269, 2015.

[32] IBM Corp. Released 2013. IBM SPSS Statistics for Windows, Version 22.0, IBM Corp, New York, 2013.

[33] Cook, R.D., "Detection of Influential Observations in Linear Regression", Technometrics, Volume 19, pp.15-18, 1977.

[34] O'brien, R.M., "A Caution Regarding Rules of Thumb for Variance Inflation Factors", Quality and Quantity, Volume 41, No.5, pp.673-690, 2007.

[35] Foss, T., Stensrud, E., Kitchenham, and B., Myrtveit, I., "A Simulation Study of the Model Evaluation Criterion MMRE", IEEE Transactions on Software Engineering, Volume 29, No. 11, pp. 985-95, 2003.

[36] Port, D., and Korte, M., "Comparative Studies of the Model Evaluation Criterions MMRE and PRED in Software Cost Estimation Research", Proceedings of the Second ACMIEEE International Symposium on Empirical Software Engineering and Measurement, 2008.

[37] Conte, S.D., Dunsmore, H.E., and Shen, V.Y., "Software Engineering Metrics and Models", BenjaminCummings Publishing Co.,Redwood, 1986.

[38] Myrtveit, I., Stensrud, E., and Shepperd, M., "Reliability and Validity in Comparative Studies of Software Prediction Models", IEEE Transactions on Software Engineering, Volume 31, No. 5, pp. 380-391, 2005.