Received April 6, 2021, accepted May 26, 2021, date of publication June 3, 2021, date of current version June 14, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3085752

Improving the Accuracy of Early Software Size Estimation Using Analysis-to-Design Adjustment Factors (ADAFs)

MARRIAM DAUD AND ALI AFZAL MALIK, (Senior Member, IEEE)
Department of Computer Science, FAST School of Computing, National University of Computer and Emerging Sciences (NUCES), Lahore 54700, Pakistan
Corresponding author: Marriam Daud (marriamdaud@gmail.com)

ABSTRACT Early software size estimation is a challenging task since limited information is available at the time of project inception. Additional information, however, is gradually added as development progresses. The goal of this research is to quantitatively capture the impact on early software size estimation of this additional information introduced especially when transitioning from the analysis phase to the design phase by comparing the analysis class diagram (ACD) and the design class diagram (DCD). We introduce a new class of metrics called analysis-to-design adjustment factors (ADAFs) to accomplish this goal. ADAFs are calculated for four different class diagram metrics – number of classes (NOC), number of attributes (NOA), number of methods (NOM), and number of relationships (NOR) – used in different class diagram-based software size estimation models. We use practical, theoretical, and empirical validation methods to evaluate the applicability of these ADAFs. To assess the utility of these ADAFs in early software size estimation, we compare the accuracy of existing early software size estimation models before and after the application of ADAFs. Results indicate a marked improvement in the accuracy of these models after the application of ADAFs. Furthermore, regression-based models employing problem domain metrics have also been built to predict these ADAFs. All of these models are statistically significant (p-values < 0.05) with R^2 values between 0.42 and 0.88.

INDEX TERMS Analysis-to-design adjustment factors (ADAFs), class diagram, early software size estimation, empirical validation, multiple linear regression models.

I. INTRODUCTION

Estimation of software size, development effort, and cost are essential for project planning, which, in turn, is an important component of software project management [1]. Software size estimation is a crucial task as many software cost and effort estimation models take software size, measured in Lines of Code (LOC) [2] and/or Function Points (FPs), [2], [3] as input. Inaccurate software size estimation (overestimation or underestimation), therefore, leads to incorrect effort and cost estimation of a software project which, in turn, may lead to project failure.

In the past, researchers have used metrics extracted from different analysis and design models (e.g. use case diagram, ER diagram, sequence diagram, class diagram, etc.) to predict software size [4]–[18]. This research focuses on the use of class diagrams for this purpose, since class diagrams are one of the most widely used models [19]. A class diagram [20], [21] is a static class-based model that depicts the overall structure of the software. The basic building block of a class diagram is a class which has different attributes and operations. Classes are connected to each other via relationships e.g. association, aggregation, generalization, etc.

Initially, during the analysis phase, an analysis class diagram (ACD) is created which focuses on the problem domain only. It does not contain aspects of the solution. Later, during the design phase, this ACD is transformed into a design class diagram (DCD) which contains aspects of the solution as well. The level of abstraction, therefore, reduces while moving from ACD to DCD [21].

Figure 1 illustrates this difference by showing the analysis and design versions of a single class named Product, which may be a core class in domains such as ecommerce, inventory, and point-of-sale (POS).

One can easily determine the values of metrics such as number of attributes (NOA) and number of methods (NOM)

The associate editor coordinating the review of this manuscript and approving it for publication was Francisco J. Garcia-Penalvo.

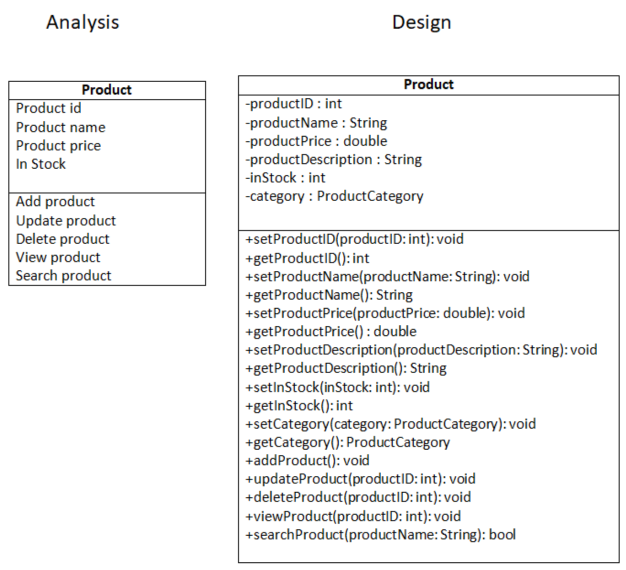

Analysis
Design
Product
Product id
Product name
Product price
In Stock
Add product
Update product
Delete product
View product
Search product
Product
-productID : int
-productName : String
-productPrice : double
-productDescription : String
-inStock : int
-category : ProductCategory
+setProductID(productID: int): void
+getProductID(): int
+setProductName(productName: String): void
+getProductName(): String
+setProductPrice(productPrice: double): void
+getProductPrice() : double
+setProductDescription(productDescription: String): void
+getProductDescription(): String
+setInStock(inStock: int): void
+getInStock(): int
+setCategory(category: ProductCategory): void
+getCategory(): ProductCategory
+addProduct(): void
+updateProduct(productID: int): void
+deleteProduct(productID: int): void
+viewProduct(productID: int): void
+searchProduct(productName: String): bool

FIGURE 1. Analysis and design versions of the same class (adapted from [22]).

for both versions of this class. In the analysis version of this Product class, NOA is 4 and NOM is 5 while, in the design version of the same class, NOA is 6 and NOM is 17. It is apparent from even this simple example that more detailed information is available in the DCD as compared to the ACD. Our research exploits this additional information and proposes a method to quantitatively capture the impact of this additional information by introducing a new class of metrics called the analysis-to-design adjustment factors (ADAFs).

ADAFs are calculated for four different class diagram metrics – number of classes (NOC), number of attributes (NOA), number of methods (NOM), and number of relationships (NOR) – used in different class diagram-based software size estimation models. The applicability of these ADAFs is evaluated through practical, theoretical, and empirical validation methods proposed in [23]–[25]. To assess the utility of these ADAFs in early software size estimation, we compared the accuracy of existing early software size estimation models before and after the application of ADAFs. Results reveal that improvement in the quality of the inputs (resulting from the application of ADAFs) enhances the accuracy of these early software size estimation models. We have also built regression-based models using problem domain metrics (which are available in the early phases) to estimate these ADAFs. Results show that all of these models are statistically significant (p-values < 0.05) with R^2 values between 0.42 and 0.88.

The rest of the paper is structured as follows. Section II describes the metrics used in this study. Section III discusses related work and explains how our work is different. Section IV describes our research methodology, while evaluation and empirical validation of ADAFs is explained in Section V. Section VI presents ADAFs prediction models. Section VII illustrates the usage and importance of the proposed methodology using a worked-out example. Section VIII sheds some light on the threats to validity of this study, while Section IX summarizes the main conclusions and provides some directions for future work in this area.

II. PROBLEM DOMAIN METRICS

Table 1 lists the entire problem domain metrics used in this study, along with their brief definitions. These metrics were chosen because they are widely used for estimating software size [4], [7]–[18], [28]. Besides this, values of these metrics can be extracted from artifacts prepared in the early phases of the software development life cycle (SDLC) i.e. (graphical) use case diagram, (structured text-based) use case descriptions, and ACD.

Metrics 1–11 can be easily obtained from ACD. Most of these are simple counts while others are simple averages or sums derived from these counts. Metrics 12–17 are related to use case diagrams and use case descriptions and are based on the work done by Karner [7] which presented a method to calculate the size of software in terms of use case points (UCP). In this method, four variables – unadjusted actor weights (UAW), unadjusted use case weights (UUCW), technical complexity factors (TCF), and environmental complexity factors (ECF) – are required to calculate UCP. Weights of actors and use cases are based on three complexity groups i.e. simple, average, and complex. The use case description is utilized to count the success transactions (s_t) and alternate transactions (a_t) of a use case. The total transactions (T_i) of a use case i are the sum of s_t_i and a_t_i. UUCW is the sum of all weighted use cases. Even though UAW, TCF, and ECF contribute to software size, we have used only UUCW since, according to past research by Silhavy *et al.* [8] it contributes more to the size of the system.

The last two metrics (18 and 19) are obtained from the work done by Nassif [9]. This work introduced a new method to calculate software size from use case descriptions. This method proposes six complexity groups (i.e. very low, low, normal, high, very high, and extreme high) for use cases and assigns each group a different weight. The total transactions (T_i) of use case i are defined as the sum of s_ti and (a_ti/2) while the total size (NassifSize) of a project is obtained using the weighted sum of the number of use cases in each complexity group.

III. RELATED WORK

Many researchers have tried to estimate software size by using different metrics extracted from UML class diagrams. One of the earliest studies on class diagram-based size estimation was reported by Mišic' and Tešic' [10]. They proposed ordinary least squares (OLS) regression-based software size estimation models constructed using 7 C++ industrial projects and showed the importance of two class diagram metrics (i.e. NOC and NOM) in software size estimation. The relationship between the same metrics extracted from DCD and source code was also explored which provided

TABLE 1. Problem domain metrics.

S #	Metric	Definition
1	NOC	The total number of classes in a class diagram
2	NOA	The total number of attributes in a class diagram
3	NOM	The total number of methods in a class diagram
4	AvgNOA	The average number of attributes per class AvgNOA= NOA/NOC
5	AvgNOM	The average number of methods per class AvgNOM = NOM/NOC
6	NODep	The total number of dependency relationships
7	NOAss	The total number of association relationships
8	NOComp	The total number of composition relationships
9	NOAgg	The total number of aggregation relationships
10	NOGen	The total number of generalization relationships
11	NOR	The total number of relationships in a class diagram NOR = NODep + NOAss + NOComp + NOAgg + NoGen
12	NOAct	The total number of actors in a use case diagram
13	NOUseC	The total number of use cases in a use case diagram
14	ST	The sum of success transactions of all use cases in a use case diagram $\mathrm{ST} = \sum_{i=1}^{n} \mathrm{s\_t}_i$ where n is the total number of use cases in a use case diagram.
15	AT	The sum of alternate transactions of all use cases in a use case diagram $\mathrm{AT} = \sum_{i=1}^{n} \mathrm{a\_t}_i$ where n is the total number of use cases in a use case diagram.
16	Trans	The sum of total transactions of all use cases in a use case diagram $\mathrm{Trans} = \sum_{i=1}^{n} \mathrm{s\_t}_i + \mathrm{a\_t}_i$ where n is the total number of use cases in a use case diagram.
17	UUCW [7]	Unadjusted use case weight $\mathrm{UUCW} = \sum_{i=1}^{3} n_i * w_i$ where n_i is the number of use cases of group i and w_i is the complexity weight of the group to which the use cases belong. The <u>three</u> complexity groups are: simple, average, and complex.
18	NassifTrans	The (weighted) sum of total transactions of all use cases in a use case diagram $\mathrm{NassifTrans} = \sum_{i=1}^{n} \mathrm{s\_t}_i + \frac{\mathrm{a\_t}_i}{2}$ where n is the total number of use cases in a use case diagram.
19	NassifSize [9]	$\mathrm{NassifSize} = \sum_{i=1}^{6} n_i * w_i$ where n_i is the number of use cases of group i and w_i is the complexity weight of the group to which the use cases belong. The <u>six</u> complexity groups are: very low, low, normal, high, very high, and extreme high.

models with R^2 values between 0.83 and 0.98. Despite these ground-breaking results, this study had some limitations e.g. a small dataset was used, other class diagram metrics (CDM) contributing to software size (e.g. NOA, NOR, etc.) were not explored, and most importantly, the models proposed were built using design-level metrics which are not available during the early phases of SDLC. Our research, on the other hand, enables early software size estimation by using information that is available in the early phases of SDLC coupled with ADAFs.

A pilot study was conducted by Antoniol *et al.* [11] using 8 C++ industrial systems to inspect the relationship between Object-Oriented Function Points (OOFP) of a system and its size in LOC. Later, Antonial *et al.* [12] re-investigated the relationship between OOFP and software size in LOC using a large dataset (29 C++ industrial subsystems) which was refined after identifying and controlling the factors most affecting software size estimation. They found that the CDM-based model was significantly better than the OOFP-based model. Chen *et al.* [13] used a dataset comprising 14 student projects to develop linear, logarithmic, exponential, and polynomial models to estimate software size in SLOC. Their results revealed that software size in SLOC was well correlated with NOC and number of external use cases.

Bianco and Lavazza [14] used 12 Java student projects to build and compare software size estimation models based on OOFP approach and CDM (i.e. NOA and NOM). Their findings were similar to a previous study [12] i.e. the model based on OOFP was not better than the model based on CDM. Later, Bianco and Lavazza [15] used 5 open source projects to investigate whether correlation similar to the one for student projects existed between: (1) LOC and CDM and (2) OOFP and CDM. They did not find the same correlation for (1) but results were better for (2). Our research uses their size estimation model (to determine the impact of ADAFs) since it was also developed using student projects and is based on CDM.

Tan and Zhao [16] built and validated regression-based size estimation models using NOC (i.e. entity types), NOA, and NOR extracted from the entity-relationship diagram of the system. Later, Tan *et al.* [17] proposed a method for early software size estimation using a conceptual data model of information systems. They built and validated regression-based size estimation models based on three metrics (i.e. NOR, NOC, and AvgNOA) and identified that these simple CDM had good capability to predict software size. Conceptual data models in this study, however, were mostly extracted through reverse engineering the source codes.

Harizi [18] proposed a method for estimating software size from its class diagram. He proposed a list of class-based parameters along with their weights. For each class, parameters were counted and multiplied by their weights, and the resulting values were summed up to get the final size of a class. The final size of the system was calculated by adding size of all the classes in the class diagram. In this study, empirical validation for weight assignment criteria was not performed, design-level metrics were used, which are not available during the early phases of SDLC, and double counting was done in terms of relationships.

Lazic *et al.* [26] presented a study to compare existing regression-based size estimation models with their proposed models. Our research uses their size estimation model (to determine the impact of ADAFs) since it was developed using CDM (i.e. NOC, NOA and NOM) and software size as SLOC. Later, Alashhb and Lazic [27] showed the usefulness of surface response modeling analysis to identify predictors contributing more to size estimation model accuracy.

Zhou *et al.* [28] compared the accuracy of class diagram-based source code size estimation approaches. Based on a large dataset (100 open-source Java projects), they built size estimation models utilizing 10 CDM, 5 function point metrics, 8 different modeling techniques and 2 transformations. Their results revealed that a model built using object-oriented project size (Oops) metric and OLS regression with a logarithmic transformation attained the highest accuracy than other models.

Ayyildiz and Koçyigit [29]–[31] investigated the correlation between the problem domain metrics and solution domain metrics. In study [31] they used a dataset consisting of 14 Java industrial systems to find the correlation between two problem domain metrics – number of distinct nouns (NDN) and number of distinct verbs (NDV) extracted from the documentation of system – and two solution domain metrics – NOC and NOM – extracted from the source code of the system. Their results showed that problem domain metrics had a good ability to predict solution domain metrics and software final size. A small dataset was used in this study and other CDM contributing to software size (e.g. NOA, NOR, etc.) were not explored.

Kiewkanya and Surak [32] developed an automated tool for regression-based software size estimation of C++ projects. They used 8 structural complexity metrics extracted from DCD. Like some other studies mentioned earlier, the main limitation of this study was also the use of design-level metrics which are not available during the early phases of SDLC.

So far, to the best of our knowledge, no work has been done to quantitatively capture the additional information introduced when moving from ACD to DCD using ADAFs and to investigate its impact on class diagram-based early software size estimation. This research attempts to fill this gap. An additional contribution of this research lies in investigating the efficacy of problem domain metrics in predicting ADAFs.

IV. RESEARCH METHODOLOGY

Figure 2 gives a pictorial summary of our research methodology which consists of the following steps:

A. STEP 1: DETERMINE SOFTWARE SIZE IN SLOC

The first step of this process is to extract software size. Software size of Java, C++, and VB.NET projects is automatically extracted using a software tool called Understand 5.1 [33]–[35]. Only non-comment and non-blank physical source lines of code (SLOC) are considered.

B. STEP 2: EXTRACT DESIGN CLASS DIAGRAM (DCD)

The DCD is automatically extracted through reverse engineering the source code of a project using Understand 5.1.

C. STEP 3: CALCULATE INPUT METRICS VALUES FROM DCD: DM1, . . . , DM4

Values of four DCD metrics (i.e. DM1 = NOC, DM2 = NOA, DM3 = NOM, and DM4 = NOR) are obtained from the reverse-engineered DCD obtained in step 2. Values of NOC, NOA and NOM are automatically extracted using Understand 5.1. For Java projects, values of NOR are extracted automatically by using two software tools, i.e. Astah UML [36] and SDMetrics [37]. For C++ projects, on the other hand, values of NOR are determined manually by the first author of this study. For industrial projects implemented in VB.NET, values of NOR are provided by the software house (see Section V.C below).

D. STEP 4: CALCULATE INPUT METRICS VALUES FROM ACD: AM1, . . . , AM4

Values of the same four input metrics are extracted from the ACD available in the SRS of the project. All of these ACD metrics (i.e. AM1 = NOC, AM2 = NOA, AM3 = NOM, and AM4 = NOR) are extracted manually by the first author of this study.

E. STEP 5: CALCULATE ADAF FOR EACH INPUT METRIC: ADAF1, . . . , ADAF4

The next step is to calculate ADAF for each input metric as shown in (1) below. $\mathrm{DMx_i}$ is an input metric (e.g. NOC) extracted from the DCD of project i and $\mathrm{AMx_i}$ is the same metric (i.e. NOC) extracted from the ACD of project i. Values of x range from 1 to 4 since we use only 4 input metrics (see previous steps).

$$\mathrm{ADAFx_i} = \frac{\mathrm{DMx_i}}{\mathrm{AMx_i}} \tag{1}$$

F. STEP 6: CALCULATE MEAN ADAF FOR EACH INPUT METRIC: MADAF1, . . . , MADAF4

Mean ADAF (MADAF) for each input metric is calculated using the formula given in (2) where n represents the total number of projects.

$$\mathrm{MADAFx} = \frac{1}{n} \sum_{i=1}^{n} \mathrm{ADAFx_i} \tag{2}$$

G. STEP 7: ADJUST EACH INPUT METRIC: MADAF1*AM1, . . . , MADAF4*AM4

The extra information captured by ADAFs is now used to adjust the values of input metrics obtained from ACD to improve the accuracy of early size estimation. This adjustment is done by multiplying each ACD input metric (e.g. NOC) with the mean ADAF of that input metric (calculated earlier in Step 6) to obtain the adjusted ACD metric (AAM) as shown in (3) below.

$$\mathrm{AAMx_i} = \mathrm{MADAFx} * \mathrm{AMx_i} \tag{3}$$

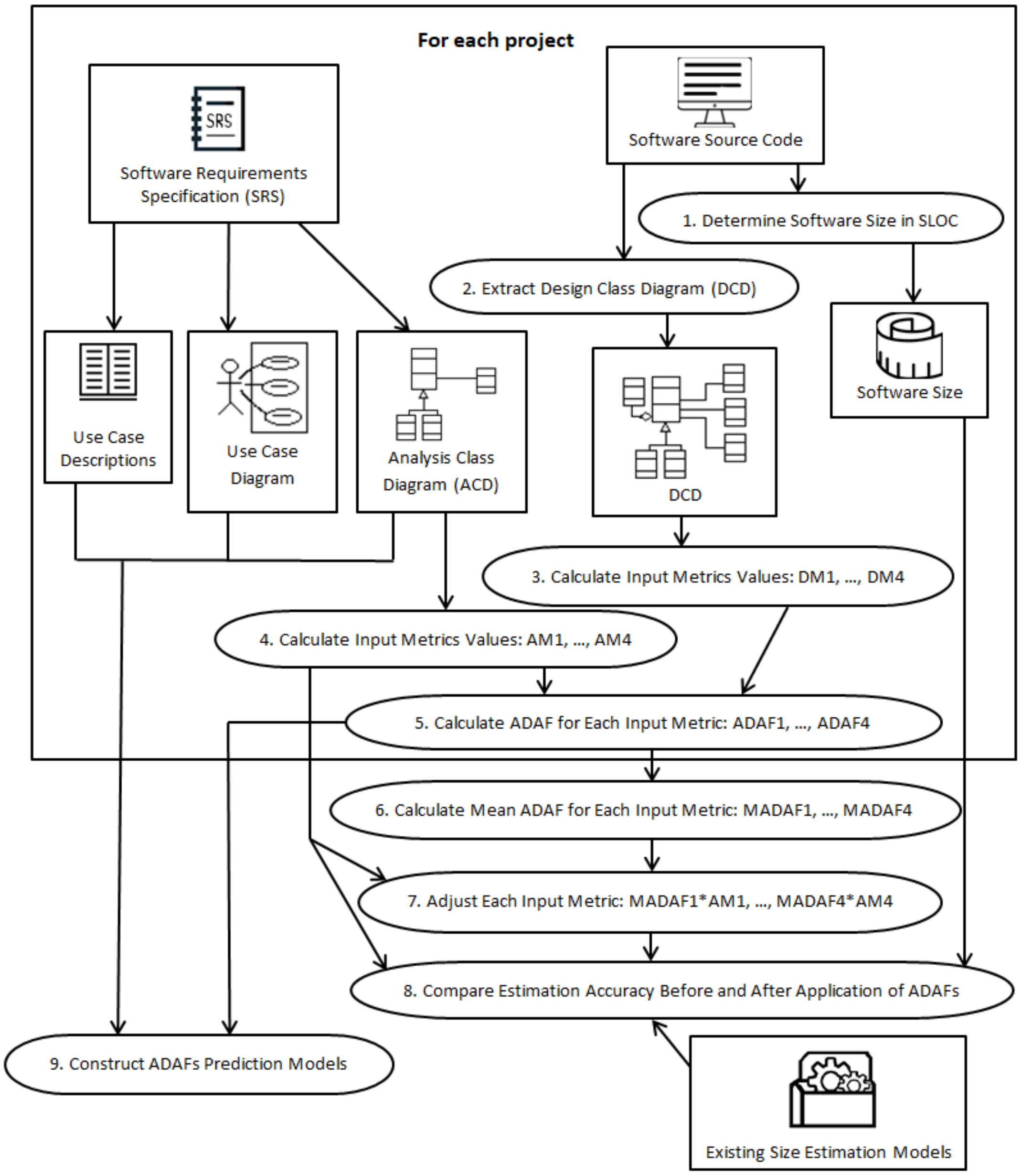

For each project
Software Requirements Specification (SRS)
SRS
Software Source Code
1. Determine Software Size in SLOC
2. Extract Design Class Diagram (DCD)
Use Case Descriptions
Use Case Diagram
Analysis Class Diagram (ACD)
DCD
Software Size
3. Calculate Input Metrics Values: DM1, ..., DM4
4. Calculate Input Metrics Values: AM1, ..., AM4
5. Calculate ADAF for Each Input Metric: ADAF1, ..., ADAF4
6. Calculate Mean ADAF for Each Input Metric: MADAF1, ..., MADAF4
7. Adjust Each Input Metric: MADAF1*AM1, ..., MADAF4*AM4
8. Compare Estimation Accuracy Before and After Application of ADAFs
9. Construct ADAFs Prediction Models
Existing Size Estimation Models

FIGURE 2. Research methodology.

H. STEP 8: COMPARE ESTIMATION ACCURACY BEFORE AND AFTER APPLICATION OF ADAFS

Existing class diagram-based software size estimation models are used to estimate software projects' sizes before and after application of ADAFs. In order to ascertain the value addition of our approach, accuracy of these models is compared before and after application of ADAFs.

I. STEP 9: CONSTRUCT ADAFS PREDICTION MODELS

Once the utility of ADAFs has been established, prediction models for all ADAFs (listed in Step 5) are constructed. Values of remaining problem domain metrics (see Table 1) are obtained manually from the ACD, use case diagram, and use case descriptions available in the SRS of a project. These problem domain metrics are then used as potential predictors of ADAFs.

V. EVALUATION AND EMPIRICAL VALIDATION OF ADAFS

The framework proposed by Misra *et al.* [23] was used for evaluation and validation of ADAFs because it provides a more comprehensive criteria as compared to other frameworks proposed in the literature. This framework consists of the following steps:

1) Practical evaluation
2) Theoretical evaluation and establishment of measurement scale

3) Empirical validation
   3.1) Preliminary empirical validation
   3.2) Advanced empirical validation
4) Self-evaluation of proposed metric: discussion of pros, cons, and future work

A. PRACTICAL EVALUATION

In this section, 5 criteria or features suggested by Misra *et al.* [23], are adopted to evaluate the practical utility of the ADAFs as mentioned below:

Objective or goal of measure The objective of ADAFs is to improve the accuracy of early software size estimation.

Users and scope of the measure:

ADAFs can be used by project managers to estimate software size during the analysis phase of SDLC by supplementing the information available in ACD.

Identify the entities and attributes to be measured:

ADAFs can be used to estimate the size (i.e. internal attribute) of software source code (i.e. entity).

Definition of metric and its measuring methods/instruments:

ADAF of a metric M is defined as the ratio of the DCD value of M and the corresponding ACD value of M. The formula to calculate ADAFs is formally defined in section IV, step 5. The inputs to calculate ADAFs are collected from ACD and DCD as shown in section IV, steps 2-4. We computed ACD metrics manually and DCD metrics using existing automated tools. In the future, we plan to develop a tool to automatically count the required components (i.e. ACD and DCD metrics).

Relationship between attribute and metric:

There is an indirect relation between ADAFs and software size. ADAFs provide extra information that is helpful in estimating software size early in the software development life cycle.

B. THEORETICAL EVALUATION

To perform a theoretical evaluation of ADAFs, we used the evaluation method proposed by Misra [24] because it was developed specifically for object-oriented metrics. The important features (from the perspective of measurement theory) suggested by this model [24] are as follows:

1) *Entity:* The ADAFs are measured at the class diagram level. Therefore, the entities for ADAFs are ACD and DCD.
2) *Property:* The property for ADAFs is complexity.
3) *Metric definition:* The metric definition and formula is presented in section IV, step 5.
4) *Attributes:*

4.1) *Internal attributes:* The internal attributes for the ACD are total number of classes, total number of attributes, total number of methods, and total number of relationships. Similarly, the internal attributes for the DCD are total number of classes, total number of attributes, total number of methods, and total number of relationships.

4.2) *External attributes:* The external attribute that ADAFs address is maintainability.

It is necessary to determine the scale type – nominal, ordinal, interval, ratio and absolute [2] – for a metric because it describes the empirical properties of a metric. Consequently, scale type tells us what kind of operations (e.g. mean, median, etc.) can be performed on the metric. In case of ADAFs, the scale type is ratio.

C. EMPIRICAL VALIDATION

In this section, preliminary empirical validation (employing 73 student projects) and advanced empirical validation (employing 11 real industry projects) of ADAFs is conducted using the model proposed by Misra [25]. However, the second step of advanced empirical validation (i.e. application of ADAFs in real projects from industry, developed in different languages and environments) is left as a task to be performed in the future.

1) PROJECT DATASETS

For preliminary empirical validation, we collected 73 student projects from different offerings of two undergraduate-level courses – Object-Oriented Analysis and Design (OOAD) and Software Engineering (SE) – taught in a renowned private university in Lahore over a period of six years. To minimize the impact of variations in exogenous variables, we focused on only those projects which implemented a management information system (MIS) of some type and were programmed using either C++ or Java. Furthermore, only those projects were selected for which documentation and final source code was available. Based on their interface and execution environment, these projects were assigned to three different categories i.e. desktop applications–GUI, desktop applications–command line, and web applications.

We encountered a lot of difficulty in acquiring relevant data for industrial projects since a number of software houses were reluctant to share their systems' documentation and artifacts due to reasons related to confidentiality. To ensure quality, we have collected only those industry projects for which we can verify data. Consequently, we collected metrics from 11 projects at a renowned software house in Lahore. This software house is a member of Pakistan Software Houses Association for IT and ITeS (P@SHA). All of these 11 projects were implemented in VB.NET and detailed documentation and user manuals were available for these projects. ACDs were created using the requirements documents of these projects.

In this study, we investigated six different datasets. Five (Dataset #1 to Dataset #5) contained student projects, while one (Dataset #6) contained real industry projects. Table 2 summarizes the details of these six datasets. Dataset #1, for instance, consists of 31 C++ GUI-based desktop applications, while Dataset #3 consists of 11 Java command-line-based desktop applications.

According to Ungan *et al.* [38] improving the quality of SRS decreases the gap between the estimated size and the

TABLE 2. Dataset characteristics.

Sr #	Dataset Number	Data Points	PL	Project Category	Minimum Size	Maximum Size	Mean Size	Std. Deviation of Size
1	Dataset #1	31	C++	Desktop applications – GUI	493	76,479	7,365.29	13,188.008
2	Dataset #2	19	Java	Desktop applications – GUI	1,220	27,373	7,534.00	7,026.180
3	Dataset #3	11	Java	Desktop applications – command-line	975	2,038	1,355.73	329.504
4	Dataset #4	12	Java	Web applications	350	5,821	2,373.08	1,595.746
5	Dataset #5	42	Java	All three categories	350	27,373	4,441.33	5,534.703
6	Dataset #6	11	VB.NET	Desktop applications – GUI	31,138	1,04,611	66,009.45	21,184.671

PL: programming language, Size: project size in SLOC

TABLE 3. Pre-processing activities performed on all datasets.

Preprocessing Activities	Dataset #1	Dataset #2	Dataset #3	Dataset #4	Dataset #5	Dataset #6
Activity #1	4	3	0	0	3	11
Activity #2	16	9	4	10	23	0
Activity #3	19	14	9	9	32	0

TABLE 4. Mean and median ADAFs values for all datasets.

	Dataset #1		Dataset #2		Dataset #3		Dataset #4		Dataset #5		Dataset # 6	
	Mean	Median	Mean	Median	Mean	Median	Mean	Median	Mean	Median	Mean	Median
ADAF(NOC)	2.71	2.63	2.30	2.22	1.81	1.43	3.56	3.30	2.54	2.38	4.26	4.16
ADAF(NOA)	2.03	1.90	9.26	7.79	1.87	1.5	1.86	1.97	5.21	2.4	5.76	5.64
ADAF(NOM)	8.88	8.42	10.76	9.90	5.5	4.95	8.13	7.24	8.63	7.49	10.19	10.14
ADAF(NOR)	3.31	2.67	4.70	4.06	4.18	2.57	4.2	3.2	4.42	3.82	4.65	4.79

final true size of software. Therefore, some pre-processing activities are conducted to make the information in datasets complete, consistent, and correct. In particular, the following pre-processing activities were performed manually:

1) If problem domain artifacts (e.g. ACD, use case diagram, etc.) were missing, they were created using the information contained in the requirements and other artifacts available in the project's documentation.
2) If the project's documentation did not match its implementation (e.g. some documented use case was not implemented), the problem domain artifacts were corrected to make them consistent with the source code.
3) If there was a discrepancy between two or more problem domain artifacts, it was removed to make these artifacts consistent with each other.

Table 3 summarizes the pre-processing activities performed on all datasets. For example, in case of Dataset #1, we performed pre-processing Activity #1 with 4 projects for which 1 project had missing ACD and 3 projects had missing detailed use case descriptions, while Activity #2 was performed with 16 projects for which less or more documentation was available as compared to the source code.

2) ANALYSIS OF ADAFS

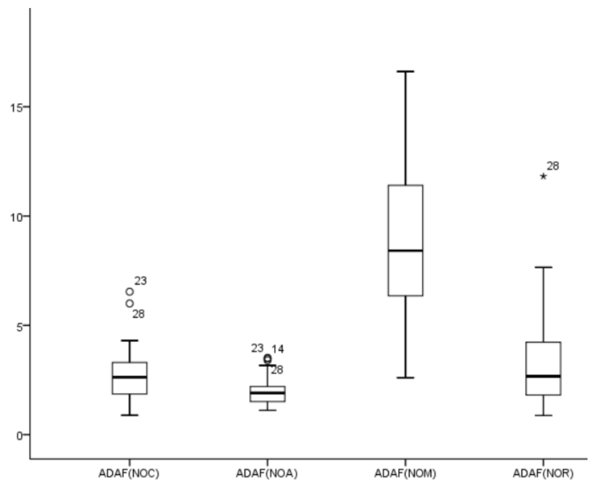
Table 4 shows the mean ADAFs values for all six datasets. These values were obtained using Steps 2–6 of our research methodology. The value of mean, however, is affected by outliers (i.e. very high or low values as compared to the rest of the dataset). Therefore, median values were also calculated for comparison. Values highlighted in this table represent situations in which the mean values are more than 30% higher as compared to median values.

A careful look at Table 4 reveals that in all cases except (Dataset #4, ADAF (NOA)) and (Dataset #6, ADAF (NOR)), ADAFs mean values are higher than their median values. This implies that the distribution of most ADAFs is skewed rightwards and a few outliers are present at the higher end. Another observation is that, as compared to Java command-line-based desktop systems (Dataset #3), Java GUI-based desktop systems (Dataset #2) have higher values of ADAFs. This is expected due to the presence of additional GUI-related classes, attributes, and methods. Last, but not the least, the value of ADAF (NOA) is much higher for Dataset #2 as compared to Dataset #1. This is because the software tool that we have used for counting (i.e. Understand 5.1) does not count GUI attributes for C++ projects by default.

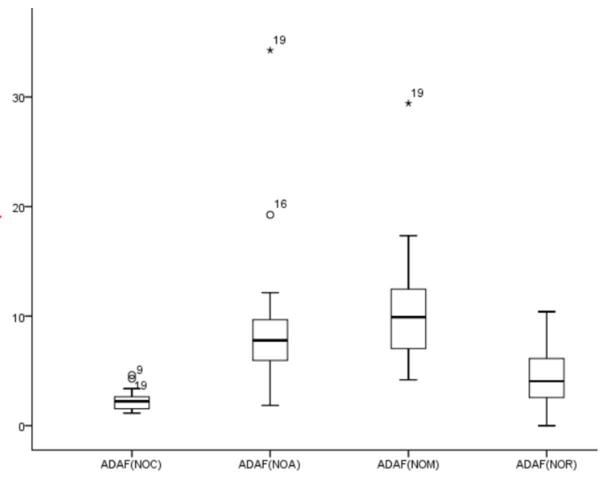
IBM SPSS statistical tool [39], [40] is used to draw boxplots of ADAFs. These boxplots shown in Figure 3 were used to identify outliers (indicated as circles and asterisks) in each dataset. Table 5 shows the mean and median ADAFs values for all datasets recomputed after removing these outliers. Value highlighted in Table 5 is the only case (Dataset #5, ADAF (NOA)) where mean value is more than 30% higher as compared to median value.

3) EVALUATION CRITERIA

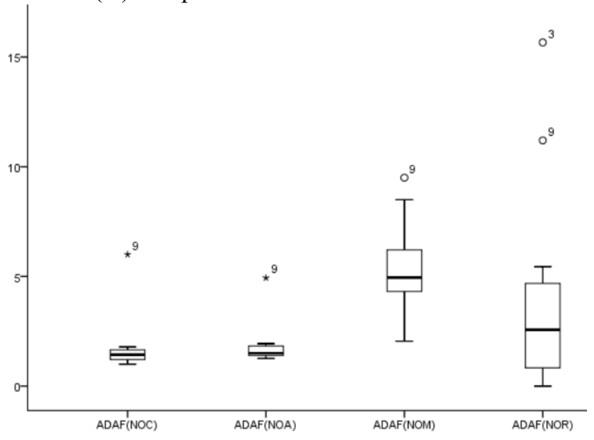
The performance metrics used in this study to measure and compare the accuracy of existing size estimation models before and after application of ADAFs are mean magnitude of relative error (MMRE) [41], percentage relative



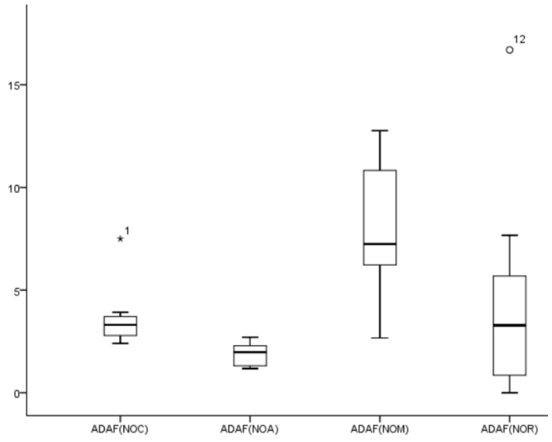
(A) Boxplot of ADAFs for Dataset #1.



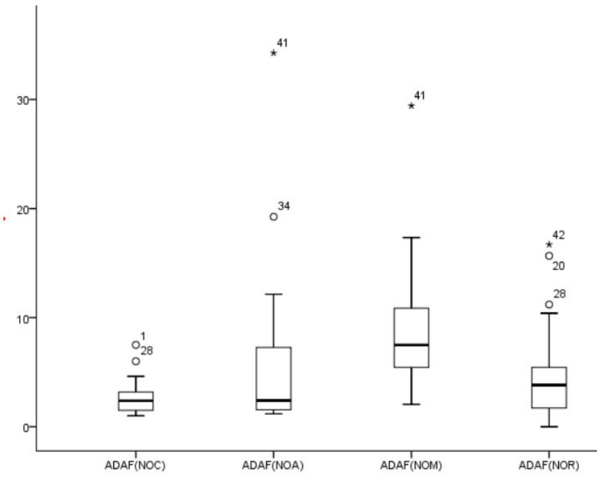
(B) Boxplot of ADAFs for Dataset #2.



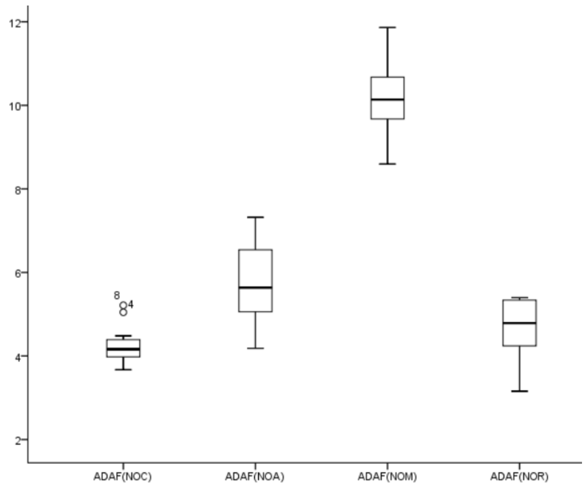
(C) Boxplot of ADAFs for Dataset #3.



(D) Boxplot of ADAFs for Dataset #4.



(E) Boxplot of ADAFs for Dataset #5.



(F) Boxplot of ADAFs for Dataset #6.

FIGURE 3. Identifying outliers using boxplot of ADAFs for all datasets.

error deviation Pred(x) [42] and mean absolute residual (MAR) [43]. Both MMRE and Pred(x) are calculated using a metric called magnitude of relative error (MRE). Equation (4) shows the formula used to calculate MRE of a data point i whereas y_i denotes the actual size and $\hat{y}_i$ denotes the predicted size. MMRE of a dataset consisting of n data points

TABLE 5. Mean and median ADAFs values for all datasets after removing the outliers.

	Dataset #1		Dataset #2		Dataset #3		Dataset #4		Dataset #5		Dataset # 6	
	Mean	**Median**	**Mean**	**Median**	**Mean**	**Median**	**Mean**	**Median**	**Mean**	**Median**	**Mean**	**Median**
ADAF(NOC)	2.47	2.45	2.06	1.88	1.39	1.40	3.20	3.18	2.33	2.28	4.1	4.13
ADAF(NOA)	1.87	1.9	7.20	7.27	1.56	1.47	1.86	1.97	4.13	2.29	5.76	5.64
ADAF(NOM)	8.88	8.42	9.71	9.55	5.10	4.88	8.12	7.24	8.12	7.33	10.19	10.14
ADAF(NOR)	3.03	2.67	4.70	4.06	2.13	1.73	3.05	2.5	3.64	3.63	4.65	4.79

TABLE 6. Comparison of prediction accuracy of size estimation models before and after application of ADAFs.

Sr #	Author(s) (Year) Description	Model	Regression Model	Dataset Used	DPs	Accuracy Before Applying ADAFs				Accuracy After Applying ADAFs			
						MMRE	Pred(25)	Pred(30)	MAR	MMRE	Pred(25)	Pred(30)	MAR
1	Bianco and Lavazza (2006) Java 12 student projects and 5 open source projects	M1	LOC = 5.7 * NOM + 3.3 * NOA	Dataset #1	31	0.93	0	0	7374.6	0.62	0.06	0.1	6147.7
				Dataset #2	19	0.93	0	0	7221.6	0.56	0.05	0.05	5013.5
				Dataset #3	11	0.83	0	0	1133.1	0.4	0.09	0.27	567.87
				Dataset #4	12	0.89	0	0	2173	0.45	0.17	0.25	1311.5
				Dataset #5	42	0.89	0	0	4184.6	0.46	0.24	0.29	2941.5
				Dataset #6	11	0.95	0	0	62692.6	0.59	0	0	39835.6
2	Lazic *et al.* (2012) 8 C++ industry projects and 17 Java student and open source mixed projects	M2	LOC = 241.41 + 10.2 * NOA + 9.547 * NOM - 24.84 * NOC	Dataset #1	31	0.88	0	0	7219.8	0.53	0.23	0.26	5690.9
				Dataset #2	19	0.87	0	0	6997.8	0.54	0.21	0.37	3524.5
				Dataset #3	11	0.65	0	0	905.68	0.19	0.55	0.82	263.4
				Dataset #4	12	0.68	0.08	0.08	1896.1	0.39	0.33	0.5	989.6
				Dataset #5	42	0.75	0.02	0.02	3944.7	0.64	0.29	0.33	2574.3
				Dataset #6	11	0.9	0	0	59427.9	0.2	0.73	0.82	14522.6

DPs: Data Points

is obtained by taking the mean of all n MREs as shown in (5). Pred(x) captures the proportion of predicted values that are within x% of the actual values [44] and is calculated using (6).

$$\mathrm{MRE_i} = \frac{|y_i - \hat{y}_i|}{y_i} \tag{4}$$

$$\mathrm{MMRE} = \frac{1}{n}\sum\nolimits_{i=1}^{n} \mathrm{MRE_i} \tag{5}$$

$$\mathrm{Pred\,(x)} = \frac{1}{n}\sum\nolimits_{i=1}^{n} \begin{cases} 1 & \text{if } \mathrm{MRE_i} \le \frac{x}{100} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

$$\mathrm{MAR} = \frac{1}{n}\sum\nolimits_{i=1}^{n} |y_i - \hat{y}_i| \tag{7}$$

For Pred(x), the commonly used values of x are 25 and 30 [17], [28], [31]. We use both 25 and 30 as the value of x in this study. Equation (7) shows the formula used to calculate MAR of a dataset consisting of n data points by taking the mean of differences in the absolute value between the actual size y_i and predicted size $\hat{y}_i$.

In literature, [41], [43] MMRE was indicated to be biased towards prediction systems that underestimate. Port *et al.* [42] showed that MMRE is an unreliable metric based on its sensitivity to the presence of outliers. However, they observed that Pred(25) is a reliable estimate of model accuracy. Despite these limitations, MMRE is a most widely used accuracy metric in related literature [10]–[12], [14]–[17], [26], [28]–[31]. However, to overcome these limitations we

TABLE 7. ADAFs prediction models built using dataset #1.

ADAF Prediction Model	DPs	R^2	VIF
ADAF(NOC) = 0.198 + 1.688 * AvgNOM	28	0.46	AvgNOM = 1.000

ADAF(NOA) = 2.454 + 0.043 * AT - 0.089 * NOGen - 0.383 * AvgNOA	27	0.63	AT = 1.195
			NOGen = 1.426
			AvgNOA = 1.266

ADAF(NOM) = 1.479 + 0.236 * AT + 0.540 * NOAgg	25	0.62	AT = 1.001
			NOAgg = 1.001

ADAF(NOR) = - 0.160 + 1.784 * AvgNOM	22	0.42	AvgNOM = 1.000

TABLE 8. ADAFs prediction models built using dataset #2.

ADAF Prediction Model	DPs	R^2	VIF

ADAF(NOC) = 1.075 + 0.774 * AvgNOM - 0.003 * NassifSize	15	0.68	AvgNOM = 1.000
			NassifSize = 1.000

ADAF(NOA) = 4.339 + 0.105 * AT	15	0.45	AT=1.000
ADAF(NOM) = 4.300 + 0.150 * NOA	16	0.49	NOA= 1.000
ADAF(NOR) = - 0.892 + 0.205 * NOUseC	18	0.45	NOUseC = 1.000

TABLE 9. ADAFs prediction models built using dataset #3.

ADAF Prediction Model	DPs	R^2	VIF
ADAF(NOC) = 0.650 + 0.017 * ST	8	0.55	ST = 1.000
ADAF(NOA) = 1.045 + 0.004 * UUCW	8	0.62	UUCW = 1.000

ADAF(NOM) = 13.812 – 0.191 * NassifSize + 0.082 * UUCW	9	0.87	NassifSize = 6.350
			UUCW = 6.350

ADAF(NOR) = 11.940 – 0.079 * NassifSize	8	0.52	NassifSize = 1.000

TABLE 10. ADAFs prediction models built using dataset #4.

ADAF Prediction Model	DPs	R^2	VIF
ADAF(NOC) = 2.324 + 0.015 * Trans	9	0.52	Trans = 1.000

ADAF(NOA) = 1.235 + 0.011 * NassifSize – 0.194 * NOAct	11	0.88	NassifSize= 1.564
			NOAct = 1.564

ADAF(NOM) = 2.429 + 0.141 * Trans - 0.878 * NOGen	11	0.86	Trans = 1.416
			NOGen = 1.416

ADAF(NOR) = - 3.776 + 0.181 * Trans - 0.579 * NOC	9	0.81	Trans = 2. 273
			NOC = 2.273

have also used MAR that is symmetric and unbiased towards under or overestimation as mentioned in study [43].

4) COMPARISON OF SOFTWARE SIZE PREDICTION ACCURACY BEFORE AND AFTER APPLYING ADAFS

Table 6 shows the comparison of prediction accuracy (before and after applying ADAFs) of two existing size estimation models i.e. M1 [15] and M2 [26]. This comparison of prediction accuracy (using MMRE, Pred(25), Pred(30), and MAR metrics) shows that the prediction accuracy of both models

TABLE 11. ADAFs prediction models built using dataset #5.

ADAF Prediction Model	DPs	R^2	VIF

ADAF(NOC) = 0.450 + 1.177 * AvgNOM – 0.075 * NOGen	35	0.66	AvgNOM= 1.138
			NOGen = 1.138

ADAF(NOA) = 0.137 + 0.057 * Trans - 0.380 * NOComp	35	0.60	Trans = 1.002
			NOComp = 1.002

ADAF(NOM) = 3.427 + 0.163 * NOA - 0.453 * NOComp	37	0.46	NOA = 1.006
			NOComp = 1.006

ADAF(NOR) = -1.010 + 0.203 * AT	34	0.42	AT = 1.000

(i.e. M1 and M2) improves considerably after the application of ADAFs.

A careful look at Pred(25) and Pred(30) columns in Table 6 reveals that greater values are obtained after the application of ADAFs. For example, model M1 (Dataset #4) has 17% improvement in Pred(25) and 25% improvement in Pred(30) while model M2 (Dataset #3) has 55% improvement in Pred(25) and 82% improvement in Pred(30). We also observe a considerable decrease in MAR values ranging from 17% to 76% after the application of ADAFs. For example, model M1 (Dataset #4) has 40% decrease in MAR while M2 (Dataset #3) has 71% decrease in MAR. Model M2 (Dataset #6) has achieved the highest improvement in prediction accuracy (using all performance metrics). Furthermore, accuracy improvement occurs for all datasets implying that our proposed approach of using ADAFs for improving the accuracy of early size estimation is indeed promising.

D. SELF-EVALUATION OF ADAFS: DISCUSSION OF PROS, CONS, AND FUTURE WORK

The pros of using ADAFs are as follows:

1) ADAFs quantitatively capture the extra information introduced when transitioning from analysis phase to design phase.
2) ADAFs can be used by project managers to estimate software size during the upstream analysis phase of SDLC by supplementing the information available in ACD.
3) ADAFs are language independent, as demonstrated in this study, through empirically validating these ADAFs using C++, JAVA, and VB.NET projects.

The con of using ADAFs is as follows:

1) It is difficult to assign threshold values (i.e. upper and lower boundaries) for ADAFs values.

The proposed future work for ADAFs includes the following:

1) Replication of this study using more real projects (i.e. data from software industry) developed in different languages and in different environments, should be performed for the purpose of advanced empirical evaluation.

TABLE 12. Comparison of predictors contributing in ADAFs prediction.

	Dataset #1	Dataset #2	Dataset #3	Dataset #4	Dataset #5
	Predictors				
ADAF(NOC)	AvgNOM	AvgNOM, NassifSize	ST	Trans	AvgNOM, NOGen
ADAF(NOA)	AT, NOGen, AvgNOA	AT	UUCW	NassifSize, NOAct	Trans, NOComp
ADAF(NOM)	AT, NOAgg	NOA	NassifSize, UUCW	Trans, NOGen	NOA, NOComp
ADAF(NOR)	AvgNOM	NOUseC	NassifSize	Trans, NOC	AT

TABLE 13. ADAFs prediction models built using dataset #2.

ADAF	Prediction Model
ADAF(NOC)	1.075 + (0.774 * AvgNOM) – (0.003 * NassifSize)
ADAF(NOA)	4.339 + (0.105 * AT)
ADAF(NOM)	4.300 + (0.150 * NOA)
ADAF(NOR)	-0.892 + (0.205 * NOUseC)

2) Threshold values for these metrics should be investigated.
3) Accuracy of additional early software size estimation models should be compared with and without using ADAFs.
4) Tool development for calculating ADAFs automatically should be considered.
5) ADAFs prediction models using industry data should be built.

VI. ADAFS PREDICTION MODELS

We built a total of twenty prediction models (4 ADAFs x 5 datasets = 20 models) after observing the different behavior of ADAFs in each of the five categories/datasets from academia. The prediction models for industry dataset were not built because use cases of projects in this dataset were not accessible. IBM SPSS Modeler statistical tool was used to perform the statistical analysis. Problem domain metrics listed in Table 1 were used as potential ADAFs predictors. We considered only metrics with more than five non-zero data points [44]. Therefore, a metric NODep (not having more than five non-zero values for all datasets) was dropped from all subsequent analyzes.

Stepwise multiple liner regression (MLR) was used for model fitting. Data points with standardized residuals values greater than 2 and less than -2 were considered outliers and removed from the datasets. We also used Cook's distance [45] to identify (and, subsequently, remove) the influential outliers from the datasets. A data point is considered an outlier if its Cook's distance is greater than (4/n), where n is the total number of data points in a dataset.

The presence of multicollinearity among the predictors was checked using the variance inflation factor (VIF) [46]. As a general rule of thumb, values of the VIF exceeding 10 indicate evidence of the existence of serious multicollinearity [17], [46]. Details of prediction models are presented in Tables 7 - 11. All of these models are statistically significant (p-values < 0.05) with R^2 values between 0.42 and 0.88.

A. COMPARISON OF PREDICTORS CONTRIBUTING IN ADAFS PREDICTION

Table 12 shows the comparison of predictors of models built using different datasets. In case of GUI-based desktop applications (Dataset #1 and Dataset #2), metrics based on use cases, methods, relationships, and attributes contribute in predicting the various ADAFs. Furthermore, for each dataset, use case-based metrics contribute in predicting at least 2 out of the 4 ADAFs.

In a DCD, getter and setter methods are usually written for each attribute. So, intuitively, NOA should correlate with ADAF (NOM). This intuition is corroborated for Java datasets, but not for C++ datasets. ADAF (NOM) predictors for Datasets 1 and 2 are very different, despite the fact that both datasets contain GUI-based desktop applications. This may be due to the difference in the size of the two (C++ and Java) datasets. The predictors of Dataset #3 are different from those of the rest of the datasets. This is expected since Dataset #3 is the only dataset that contains only command-line applications which have lower ADAFs.

VII. WORKED-OUT EXAMPLE

This section demonstrates the practical application of our proposed idea with the help of a completely worked-out example. It illustrates how our ADAFs prediction models can be used for early size estimation of software development projects. In this example, we use the ADAFs prediction models built using Dataset #2. For convenience, these models are reproduced in Table 13.

Suppose a software development company hires a new project manager to manage a recently commissioned project which involves from-scratch development of a Pharmacy Management System (PMS). The business analyst provides the complete documentation of PMS, including ACD, use

TABLE 14. Values of problem domain metrics for PMS.

Metrics	NOC	NOA	NOM	AvgNOM	NOUseC	ST	AT	NassifTrans	NassifSize
Values	14	35	25	1.786	25	62	28	76	140

TABLE 15. ADAFs predictions for PMS.

ADAF	Predicted Value
ADAF(NOC)	1.075 + (0.774 * 1.786) – (0.003 * 140) = 2.037
ADAF(NOA)	4.339 + (0.105 * 28) = 7.279
ADAF(NOM)	4.300 + (0.150 * 35) = 9.55
ADAF(NOR)	-0.892 + (0.205 * 25) = 4.233

TABLE 16. Estimated software size using ADAFs.

Size Estimation Model M2	Metric * ADAF(Metric)	Estimated Software Size in SLOC
LOC = 241.41 + 10.2 * NOA + 9.547 * NOM - 24.84 * NOC	NOC * ADAF(NOC) = 14 * 2.037 = 28.52	4,411
	NOA * ADAF(NOA) = 35 * 7.279 = 254.77	
	NOM * ADAF(NOM) = 25 * 9.55 = 238.75	

case diagram, and use case descriptions to this project manager. The project manager can now easily use this early documentation to extract the problem domain metrics for PMS. Assume that the values of these metrics for PMS are those shown in Table 14. These values can now be plugged into the prediction models (see Table 13) to predict ADAFs. Table 15 shows these ADAFs predictions for PMS.

This project manager has been able to use our models to predict ADAFs during the early phases of software development, as all the inputs to these models are available early in the SDLC. Finally, the project manager can use these ADAFs to improve the quality of inputs and, hence, the estimation accuracy of existing size estimation models. Using Model M2, the estimated size of PMS (after applying ADAFs) is 4,411 lines of code as shown in Table 16.

VIII. THREATS TO VALIDITY

This section discusses potential threats to the construct, internal and external validity of our study, and how the impact of these threats was mitigated.

A. CONSTRUCT VALIDITY

The first author of this study had manually performed some pre-processing activities to provide consistency, completeness, and correctness of problem domain artifacts. To ensure the correctness of artifacts, in case of academic datasets, these artifacts were reviewed by a collaborator (with 5 years of experience in object-oriented application development and over 3 years of experience in teaching programming and OOAD concepts). For industry dataset, however, the ACDs were constructed by a team of 2 participants. One participant was the first author of this study and other was working as a software engineer in that software house. To further ensure the correctness of these ACDs, a senior software engineer (with over 10 years of development experience) reviewed and validated these ACDs.

Values of independent variables in the ADAFs prediction models were extracted manually from the (graphical) use case diagram, (structured text-based) use case descriptions, and ACD (see Table 1). To maintain consistency in manual counting, values of these independent variables were counted only by the first author of this study. To overcome possible threats to the construct validity of dependent variables (i.e. software size and ADAFs), a tool (i.e. Understand 5.1) was used for automatically counting the values of these dependent variables. This same tool has been used by previous studies [28]–[31] to collect similar metrics, making it a safe and reliable choice.

B. INTERNAL VALIDITY

The main threat to the internal validity of this study is related to how the software size is counted. In this study, we counted software size as the total number of non-comment and non-blank physical source lines of code (SLOC) of Java, C++, and VB.NET files. This definition of software size is consistent with the one used in a previous study [28] which used 100 Java projects to build size estimation models.

We have currently used mean ADAF values to improve the quality of inputs used in software size estimation models. The use of median ADAF values may affect the results. However, this switch is not expected to have a considerable effect on our results, as most of the median ADAF values are close to the mean ADAF values (see Table 5).

C. EXTERNAL VALIDITY

In this study, we have used different datasets comprising C++, Java, and VB.NET MIS projects. These datasets have yielded similar results with respect to improving the accuracy of early software size estimation. These results, however, may not be generalizable for projects of different types, sizes, languages and application domains. Similarly, other factors such as developers' experience and use of design patterns and frameworks may also affect ADAFs and, hence, influence our ADAFs prediction models. Nevertheless, irrespective of whether results vary due to these additional factors, this study has proposed a detailed and repeatable approach for utilizing ADAFs for improving early software size estimation.

IX. CONCLUSION AND FUTURE WORK

The presence of limited information during the early phases of software development makes early software size estimation a challenging task. The aim of this research was to address this challenge by quantitatively capturing the impact on early software size estimation of additional information introduced when moving from the analysis phase to the design phase. A new class of metrics called analysis-to-design adjustment factors (ADAFs) was introduced for this purpose. ADAFs were calculated for four different class diagram metrics (i.e. NOC, NOA, NOM, and NOR) commonly used in different class diagram-based software size estimation models. These ADAFs were validated both theoretically and empirically using 84 projects. Moreover, practical usefulness of ADAFs is also proved by applying it to the real projects from industry. To assess the utility of these ADAFs in early software size estimation, we compared the accuracy of existing early software size estimation models before and after the application of ADAFs. The results of this comparison reveal that our proposed approach is promising, since the accuracy of these early software size estimation models is improved after the application of ADAFs. We also built regression-based models using problem domain metrics to predict ADAFs during the early phases of software development. All of these models were statistically significant (p-values < 0.05) with R^2 values between 0.42 and 0.88.

In the future, we plan to replicate this study using industrial projects developed in different languages and in different environments. These projects may also be used for building and validating domain/category-specific regression-based early software size estimation models using ADAF-adjusted inputs. We also plan to compare the accuracy of additional early software size estimation models with and without using ADAFs. Furthermore, we may also investigate the impact of automatic source code generation tools, frameworks, design patterns, and ready-made code from libraries on ADAFs.

ACKNOWLEDGMENT

The authors are thankful to the Editor and reviewers for their valuable comments. They are also thankful to their collaborators from the industry and the academia who provided them with projects and who volunteered to review and verify the datasets.

REFERENCES

[1] D. D. Galorath and M. W. Evans, *Software Sizing, Estimation, and Risk Management: When Performance is Measured Performance Improves*, 1st ed. New York, NY, USA: Auerbach, 2006.

[2] N. E. Fenton and J. Bieman, *Software Metrics: A Rigorous and Practical Approach*, 3rd ed. Boca Raton, FL, USA: CRC Press, 2015.

[3] A. J. Albrecht, "Measuring application development," in *Proc. IBM Appl. Dev. Jt. SHARE/GUIDE Symp.*, Monterey, CA, USA, 1979, pp. 83–92.

[4] R. Silhavy, P. Silhavy, and Z. Prokopova, "Using actors and use cases for software size estimation," *Electronics*, vol. 10, no. 5, pp. 1–20, 2021.

[5] E. Ungan, "A functional software measurement approach bridging the gap between problem and solution domains," Ph.D. dissertation, Dep. Inf. Syst., Middle East Tech. Uni., Ankara, Turkey, 2013.

[6] E. Ungan and O. Demirörs, "A functional software measurement approach to bridge the gap between problem and solution domains," in *Proc. IWSM/Mensura, Softw. Meas. LNBIP*, vol. 230, 2015, pp. 176–191.

[7] G. Karner, "Resource estimation for objectory projects," *Object. Syst.*, vol. 17, pp. 1–9, Sep. 1993.

[8] R. Silhavy, P. Silhavy, and Z. Prokopova, "Analysis and selection of a regression model for the use case points method using a stepwise approach," *J. Syst. Softw.*, vol. 125, pp. 1–14, Mar. 2017.

[9] A. B. Nassif, "Software size and effort estimation from use case diagrams using regression and soft computing models," Ph.D. dissertation, School Grad. Postdoct. Stud., West. Univ., London, ON, Canada, 2012.

[10] V. B. Mišić and D. N. Tešić, "Estimation of effort and complexity: An object-oriented case study," *J. Syst. Softw.*, vol. 41, no. 2, pp. 133–143, May 1998.

[11] G. Antoniol, C. Lokan, G. Caldiera, and R. Fiutem, "A function point-like measure for object oriented software," *Empirical Softw. Eng.*, vol. 4, no. 3, pp. 263–287, Sep. 1999.

[12] G. Antoniol, R. Fiutem, and C. Lokan, "Object-oriented function points: An empirical validation," *Empirical Softw. Eng.*, vol. 8, no. 3, pp. 225–254. Sep. 2003.

[13] Y. Chen, B. W. Boehm, R. Madachy, and R. Valerdi, "An empirical study of eServices product UML sizing metrics," in *Proc. Int. Symp. Empirical Softw. Eng. (ISESE)*, Redondo Beach, CA, USA, 2004, pp. 199–206.

[14] V. Del Bianco and L. Lavazza, "An empirical assessment of function point-like object-oriented metrics," in *Proc. 11th IEEE Int. Softw. Metrics Symp. (METRICS05)*, Como, Italy, 2005, p. 10.

[15] V. D. Bianco and L. Lavazza, "Object-oriented model size measurement: Experiences and a proposal for a process," unpublished. [Online]. Available: https://www.academia.edu/18304046/Object_Oriented_Model_Size_Measurement_Experiences_and_a_Proposal _for_a_Process

[16] H. B. K. Tan, "Sizing data-intensive systems from ER model," *IEICE Trans. Inf. Syst.*, vol. E89-D, no. 4, pp. 1321–1326, Apr. 2006.

[17] H. B. K. Tan, Y. Zhao, and H. Zhang, "Conceptual data model-based software size estimation for information systems," *ACM Trans. Softw. Eng. Methodol.*, vol. 19, no. 2, pp. 1–37, Oct. 2009.

[18] M. Harizi, "The role of class diagram in estimating software size," *Int. J. Comput. Appl.*, vol. 44, no. 5, pp. 31–33, Apr. 2012.

[19] M. Guo, C. Zhang, and F. Wang, "What is the further evidence about UML?—A systematic literature review," in *Proc. 24th Asia–Pacific Softw. Eng. Conf. Workshops (APSECW)*, Nanjing, China, Dec. 2017, pp. 106–113.

[20] *Object Management Group. About the UML Specification Version 2.5.1*. Accessed: Sep. 12, 2019. [Online]. Available: https://www.omg.org/spec/UML/About-UML/

[21] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, 7th ed. New York, NY, USA: McGraw-Hill, 2010.

[22] Creately. *How to Draw Class Diagrams: Simple Class Diagram Rules for Clear Communication*. Accessed: Mar. 12, 2020. [Online]. Available: https://creately.com/diagram-type/article/simple-guidelines-drawing-uml-class-diagrams

[23] S. Misra, I. Akman, and R. Colomo-Palacios, "Framework for evaluation and validation of software complexity measures," *IET Softw.*, vol. 6, no. 4, pp. 323–334, Aug. 2012.

[24] S. Misra, "Evaluation criteria for object-oriented metrics," *Acta Poly. Hung.*, vol. 8, no. 2, pp. 109–136, 2011.

[25] S. Misra, "An approach for empirical validation process for software complexity measures," *Acta Poly. Hung.*, vol. 8, no. 2, pp. 141–160, 2011.

[26] L. Lazic, M. Petrovic, and P. Spalevic, "Comparative study on applicability of four software size estimation models based on lines of code," in *Proc. 6th Wseas Eur. Comput. Conf. (ECC)*, Prague, Czech Republic, 2012, pp. 71–80.

[27] M. I. Alashhb and L. Lazić, "A critical review of source code size estimation approaches for object-oriented programming languages: A comparative study," in *Proc. INFOTEH-JAHORINA*, Jahorina, Bosnia, 2016, pp. 535–540.
[28] Y. Zhou, Y. Yang, B. Xu, H. Leung, and X. Zhou, "Source code size estimation approaches for object-oriented systems from UML class diagrams: A comparative study," *Inf. Softw. Technol.*, vol. 56, no. 2, pp. 220–237, Feb. 2014.
[29] T. E. Ayyildiz and A. Koçyiğit, "Correlations between problem domain and solution domain size measures for open source software," in *Proc. 40th EUROMICRO Conf. Softw. Eng. Adv. Appl.*, Verona, Italy, Aug. 2014, pp. 81–84.
[30] T. E. Ayyildiz and A. Koçyiğit, "A case study on the utilization of problem and solution domain measures for software size estimation," in *Proc. 42nd Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA)*, Limassol, Cyprus, Aug. 2016, pp. 108–111.
[31] T. E. Ayyildiz and A. Koçyiğit, "Size and effort estimation based on problem domain measures for object-oriented software," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 28, no. 2, pp. 219–238, Feb. 2018.
[32] M. Kiewkanya and S. Surak, "Constructing C++ software size estimation model from class diagram," in *Proc. 13th Int. Joint Conf. Comput. Sci. Softw. Eng. (JCSSE)*, Khon Kaen, Thailand, Jul. 2016, pp. 1–6.
[33] S. Kumar, "Analysis of software metrics and software tools," *Int. Educ. Appl. Res. J.*, vol. 3, no. 7, pp. 86–88, Jul. 2019.
[34] E. Dias Canedo, K. Valença, and G. A. Santos, "An analysis of measurement and metrics tools: A systematic literature review," in *Proc. 52nd Hawaii Int. Conf. Syst. Sci.*, 2019, pp. 1–20.
[35] *Understand*. Accessed: Sep. 30, 2019. [Online]. Available: https://scitools.com/
[36] *Astah UML*. Accessed: Jan. 12, 2020. [Online]. Available: https://astah.net/products/astah-uml/
[37] *SDMetrics*. Accessed: Jan. 25, 2020. [Online]. Available: https://www.sdmetrics.com/
[38] E. Ungan, S. Trudel, and A. Abran, "Analysis of the gap between initial estimated size and final (true) size of implemented software," in *Proc. IWSM/Mensura*, vol. 2207, Beijing, China, 2018, pp. 123–137.
[39] *IBM SPSS Software. IBM SPSS Modeler Statistical Tool*. Accessed: Nov. 15, 2019. [Online]. Available: https://www.ibm.com/analytics/spss-statistics-software
[40] E. T. Berkman, and S. P. Reise, *A Conceptual Guide to Statistics Using SPSS*. Thousand Oaks, CA, USA: Sage, 2012.
[41] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit, "A simulation study of the model evaluation criterion MMRE," *IEEE Trans. Softw. Eng.*, vol. 29, no. 11, pp. 985–995, Nov. 2003.
[42] D. Port and M. Korte, "Comparative studies of the model evaluation criterions MMRE and pred in software cost estimation research," in *Proc. 2nd ACM-IEEE Int. Symp. Empirical Softw. Eng. Meas. (ESEM)*, Oct. 2008, pp. 51–60.
[43] M. Shepperd and S. MacDonell, "Evaluating prediction systems in software project estimation," *Inf. Softw. Technol.*, vol. 54, no. 8, pp. 820–827, Aug. 2012.
[44] L. C. Briand, J. Wüst, J. W. Daly, and D. V. Porter, "Exploring the relationships between design measures and software quality in object-oriented systems," *J. Syst. Softw.*, vol. 51, no. 3, pp. 245–273, May 2000.
[45] R. D. Cook, "Detection of influential observation in linear regression," *Technometrics*, vol. 19, no. 1, pp. 15–18, Feb. 1977.
[46] R. M. O'brien, "A caution regarding rules of thumb for variance inflation factors," *Qual. Quantity*, vol. 41, no. 5, pp. 673–690, Mar. 2007.

MARRIAM DAUD received the B.S.C.S. degree (Hons.) with double majors (software engineering and information technology) from Forman Christian College (A Chartered University), in 2010, the degree *(magna cum laude)* from FCC, in 2010, and the M.S.C.S. degree from the National University of Computer and Emerging Sciences (FAST-NUCES), in 2013, where she is currently pursuing the Ph.D. degree. She worked as a Software Engineer with WebCiters, from 2009 to 2013. From 2013 to 2015, she worked as a Lecturer with The University of Lahore (UOL). Her research interests include software engineering, software size estimation, and database systems. She received Gold Medal from Forman Christian College (A Chartered University) for her B.S.C.S. degree and the Faculty Outstanding Student Award from FCC.

ALI AFZAL MALIK (Senior Member, IEEE) received the M.S. and Ph.D. degrees in computer science from the University of Southern California (USC), Los Angeles, USA, in 2007 and 2010, respectively. He is currently working as an Assistant Professor and the Head of the Department of Computer Science, National University of Computer and Emerging Sciences (FAST-NUCES) Lahore Campus. He has undertaken research in software cost estimation at two of the world's leading research centers in software engineering, such as USC's Center for Systems and Software Engineering (CSSE) and Institute of Software, Chinese Academy of Sciences (ISCAS). His current research work focuses on areas, such as empirical software engineering and software cost estimation. He received the prestigious Fulbright Scholarship for his M.S. and Ph.D. degrees, in 2005.