Mehran University Research Journal of Engineering and Technology

https://doi.org/10.22581/muet1982.3289

2025, 44(2) 164-173

Regression-based predictive modelling of software size of fintech projects using technical specifications

Iqra Kanwal [a, *], Ali Afzal Malik [a]

[a] *FAST School of Computing, National University of Computer and Emerging Sciences (NUCES), Lahore*

[*] Corresponding author: Iqra Kanwal, Email: iqrakanwal545@gmail.com

Received: 10 June 2024, Accepted: 27 March 2025, Published: 01 April 2025

K E Y W O R D S

K-fold cross validation

Lines of code

Multiple linear regression

Size prediction model

Software size prediction

Technical specifications

A B S T R A C T

This research aims to develop a predictive model to estimate the lines of code (LOC) of software projects using technical requirements specifications. It addresses the recurring issue of inaccurate effort and cost estimation in software development that often results in budget overruns and delays. This study includes a detailed analysis of a dataset comprising past real-life software projects. It focuses on extracting relevant predictors from projects' requirements written in technical and easily comprehensible natural language. To assess feasibility, a pilot study is conducted at the beginning. Then, Simple Linear Regression (SLR) is employed to determine the relative predictive strength of eight potential predictors identified earlier. The number of API calls is found to be the strongest independent predictor (R2 = 0.670) of LOC. The subsequent phase entails constructing a software size prediction model using Forward Stepwise Multiple Linear Regression (FSMLR). The adjusted R2 value of the final model indicates that two factors – the number of API calls and the number of GUI fields – account for more than 80% of the variation in code size (measured using LOC). Model validation is performed using k-fold cross-validation. Validation results are also promising. The average MMRE of all folds is 0.203 indicating that, on average, the model's predictions are off by approximately 20% relative to the actual values. The average PRED (25) is 0.708 implying that nearly 71% of predicted size values are within 25% of the actual size values. This model can help project managers in making better decisions regarding project management, budgeting, and scheduling.

1. Introduction

In the dynamic realm of software development, innovation races hand in hand with complexity. Conceiving, developing, and delivering successful software projects in such a dynamic and complex environment, requires effective project management. Effective project management, in turn, requires thorough planning, continuous monitoring, and intelligent resource allocation.

The backbone of all software projects is an artifact called the project plan. The core of the project plan is the project schedule. This schedule is built around estimates of software size, effort, time, and cost. Therefore, the foundation of project management is estimation.

Since most real-life software projects are complicated, estimating the amount of effort, time, and resources needed to complete a project requires a methodical approach [1]. Furthermore, all such methodological approaches need a starting point which is provided by size estimation since software size is considered one of the most important determinants of project effort, duration, and staff/resources. Thus, regardless of whether the classical waterfall process or a modern agile process is

used, reliable software size estimation is crucial for developing a realistic project plan.

Software size, to a great extent, depends on the number and complexity of software features. Technical requirements specifications act as the basis for providing the details of the features required in the software that needs to be developed and maintained [2]. These technical requirements are available early in the software development life cycle (SDLC). Thus, the information contained in these technical specifications can be used for early software size estimation.

Over the years, several software size and effort estimation techniques and models have been proposed. These include function point analysis (FPA), parametric modeling, and expert judgment [3]. Parametric software cost estimation models like COCOMO II [4] and functional sizing approaches like FPA [5] have been widely used for decades to estimate project effort and size [6]. Despite their widespread usage, these approaches and models exhibit some limitations in adapting to modern development scenarios and diverse project environments. The shift towards machine learning-based models has introduced a more adaptable and data-driven approach. These models, however, rely heavily on historical data, which potentially limits their applicability in cases of novel projects lacking substantial historical references [7].

Software size is often measured in lines of code (LOC). As expected, there is no universal method, approach, or technique for LOC estimation that works for all projects. This is precisely because projects differ greatly in complexity, scale, and scope. Size estimation of complex real-life software projects often requires a more personalized approach. A size prediction model that incorporates project-specific information contained in the technical requirements specifications of a project may provide more accurate results. This study aims to build such a model.

Data collection is one of the most important parts of this research. Technical requirements specifications of past real-life projects are used as a source. Data obtained from these specifications is analyzed to identify the factors that can help in predicting the size (LOC) of a project. First, the individual predictive strength of these factors is determined using simple linear regression (SLR). Then, forward stepwise multiple linear regression (FSMLR) is used to build and validate a size prediction model using the factors identified earlier.

The following three research questions (RQs) are addressed in this study:

RQ1: Which factors (identified from the technical requirements specifications) affect the variation in LOC for different software projects?

RQ2: What is the relative predictive strength (in terms of R2 value) of each factor in predicting LOC?

RQ3: How accurately can an FSMLR-based model built using the factors identified earlier predict the size (LOC) of a software project?

The next section discusses previous work in this area. Section 3 presents a summary of our research methodology. Section 4 contains a detailed discussion of the factors identified as LOC predictors. Section 5 summarizes our pilot study, while Section 6 describes our data collection. Sections 7 and 8 present and discuss the results of SLR and MLR, respectively. The results of model validation are summarized and discussed in Section 9. Section 10 outlines the threats to the validity of these results. Finally, Section 11 highlights our main contributions and presents some directions for future work in this area.

2. Related Work

Machine learning (ML) algorithms enable computers to understand problems and produce effective and efficient solutions. Zakaria et al. [8] compared the effort estimation accuracy of four different ML algorithms i.e. Random Forest (RF), Linear Regression (LR), Regression Tree (RT), and Support Vector Machine (SVM). The effort prediction accuracy of these four ML algorithms was compared on four different datasets i.e. COCOMO NASA 1, COCOMO NASA 2, COCOMO81, and Kaushik et al., 2012. SVM was found to give good results. Another important finding from their research was that not all attributes of the dataset were relevant for effort estimation. Only five attributes of the COCOMO datasets were found to be relevant. Our current research is similar to their research in the sense that our final size prediction model (made using FSMLR) uses a subset of the eight potential predictors identified earlier. However, the main difference between our work and theirs is that we focus on size estimation while their focus is on effort estimation. Another significant distinction is that, instead of depending on a pre-existing set of predictors like COCOMO model drivers, we find our own potential predictors.

Sharma and Kushwaha [9] first proposed a new metric called Improved Requirement Based Complexity (IRBC) for quantifying the complexity of requirements documented in the software

requirements specification (SRS) of a software project. Later, they developed an approach for augmenting the original function points approach with IRBC to estimate the software development effort (SDE) of a software project. The results of this approach were also compared with other existing SDE techniques such as use case-based estimation. Our research is similar to theirs in the sense that we also use requirements-related information for early estimation. One of the main differences, however, is that our focus is on size estimation instead of effort estimation. Secondly, we use technical requirements specifications instead of SRS as input.

Ayyıldız and Koçyigit [10] studied the correlations between problem domain measures (specifically, the number of distinct nouns and verbs in requirements) and solution domain measures (specifically, the number of software classes and methods in object-oriented software) to improve early software size estimation. They analyzed data from 12 commercial software projects to develop MLR-based estimation models for solution domain metrics based on problem domain metrics. The prediction accuracy of their models was found to be acceptable. Our research also focuses on early estimation. However, instead of using parts of speech (e.g., nouns and verbs) to predict design concepts (e.g., classes and methods), our model directly predicts size (LOC) using the information contained in the technical requirements specifications.

Lind and Heldal [11] looked at the utility of using the COSMIC Functional Size Measurement (FSM) method [12] to estimate the implemented code size (in bytes) of embedded software components in the automotive industry. They found a strong correlation between the two. Like our approach, their approach can also be used for early size estimation. However, determining the COSMIC function points of a software application from a given requirements specification is a non-trivial exercise that requires prior training. On the other hand, prior training is not required when using our proposed size (LOC) estimation approach.

Table 1 presents a summary of the past related works discussed earlier in this section. Apart from summarizing the main contributions of these works, this table also highlights their limitations and main differences from our work.

Table 1

Summary of related work

Sr#	Title	Author(s)	Publication Year	Main Contributions	Limitations/ Differences
1	Software Project Estimation with Machine Learning	Zakaria et al.	2021	A comparative study of 4 different ML algorithms using 4 different datasets. Not all attributes are relevant for effort estimation.	Used pre-existing predictors (i.e. COCOMO model drivers). They are majorly focused on effort estimation instead of size estimation.
2	A Case Study on the Utilization of Problem and Solution Domain Measures for Software Size Estimation	Ayyıldız and Koçyigit	2016	MLR-based estimation models for solution domain metrics based on problem domain metrics.	Indirect estimation of software size in terms of the number of classes and methods instead of LOC.
3	Estimation of Software Development Effort from Requirements Based Complexity	Sharma and Kushwaha	2012	Prediction of software development effort using the function points approach augmented with requirement-based complexity.	Used SRS instead of technical requirements specifications. They are majorly focused on effort estimation instead of size estimation.
4	A Practical Approach to Size Estimation of Embedded Software Components	Lind and Heldal	2012	Correlation between COSMIC function points and implemented code size (in bytes) for	Determining COSMIC function points from a given requirements

	embedded components.	software specification requires prior training.

3. Research Methodology

Fig. 1 provides a pictorial summary of our research methodology. The first step was to identify the factors (hereinafter called predictors) that could play an important role in predicting software size (LOC). This was followed by a pilot study aimed at gauging the magnitude of effort required in gathering data and refining the research design before proceeding further. The next main step was data collection, which involved extracting quantitative data from technical requirements specifications as well as the final code of real-life software projects. Once the data was collected, it was pre-processed to detect any outliers. Then, the data was analyzed using both SLR and FSMLR to determine the relative strength of each predictor individually and to build a software size (LOC) prediction model. Last, but not the least, the prediction accuracy of this size prediction model was assessed using commonly used accuracy metrics, and the model was validated using k-fold cross-validation. The following sections describe these main steps of our research in more detail.

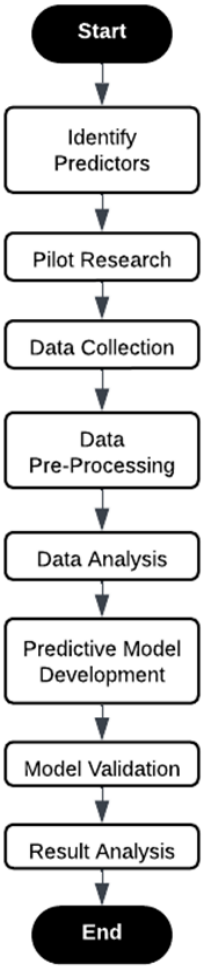

Fig. 1. Main Research Steps And Their Sequence

4. Predictor Identification

Table 2 lists all nine quantitative software size (LOC) predictors.

Table 2

Predictors

Sr#	Name	Abbreviation
1	Number of API Calls	nAPI
2	Number of Third-Party Libraries	nTPL
3	Number of GUI Fields	nGUI
4	Number of Input Validation Checks	nIVC
5	Number of Database Queries	nDBQ
6	Number of Total Execution Steps	nTES
7	Number of Scenarios	nSCN
8	Number of Security Checks	nSEC
9	Number of Exception Handling Checks	nEHC

These predictors were identified by analyzing technical requirements specifications of real-life projects and by holding discussions with experts and software development team members. This answers RQ1.

Each predictor has been given a four-letter abbreviation for easy reference. nAPI refers to the application programming interfaces (APIs) that need to be developed while implementing a feature along with the wrapper services to call these APIs. nTPL is the count of the third-party libraries and helper classes (made available by the relevant language and/or framework) that are used to implement a feature. nGUI refers to the graphical user interface (GUI) fields that need to be made available on the screen for receiving inputs. nIVC corresponds to the number of input validation checks. Each GUI field used for input can be associated with several input validation checks. Table 3 lists the various input validation checks that may be required on any particular input GUI field.

Table 3

Input validation checks

Sr#	Name
1	Min length
2	Max length
3	Min value

4	Max value
5	Regex
6	Mandatory
7	Valid selected item
8	File type (pdf, CSV, etc.)

nDBQ captures all direct interactions with the database to perform certain operations without the help of any API. Every execution step required to implement a feature is counted in nTES, while all possible scenarios (success as well as failure) are counted in nSCN. nSEC refers to the security checks required to be implemented in a feature. Table 4 provides a list of the different types of security checks that may be required. Finally, nEHC is the count of the exceptions that are handled in the code explicitly. All such exceptions result in displaying a user-friendly error message.

Table 4

Security checks

Sr#	Name
1	SQL injection
2	XSS
3	CSRF
4	File upload
5	Access redirections
6	Struts token validations
7	Public flow authentications

5. Pilot Study

A pilot study was conducted to determine the effort required to collect the relevant data from the technical requirements specifications and the source code. 10 completed features (data points) were used for this purpose. The values of the predictors (independent variables) were obtained from the technical requirements specifications, while the values of the LOC (dependent variable) were obtained from the source code. SLR analysis of this data was also performed using IBM's SPSS tool [13] to determine the tentative relationship between each independent predictor and LOC.

Another goal of this pilot study was to refine and finalize the research design before proceeding with full-fledged data collection (which included these 10 pilot study data points as well). Another potential predictor of software size (LOC) which was considered at this stage was the number of exception-handling checks (nEHC). After scrutiny, nEHC was ruled out for being redundant. It was found that nEHC contained information already captured by two other predictors (nTES and nSCN) identified earlier.

6. Data Collection

The technical requirements specifications and source code of a total of 80 features belonging to three different recently-completed industrial projects were thoroughly analyzed. All three projects were classified under the fintech category and were carried out by an ISO-certified and PCI-compliant organization with a presence in Pakistan. In all of these projects, JavaScript was used to implement the front end, and Java was used to implement the back end. Technical requirements specifications were written by solution architects and team leads using the information contained in the customer requirements. Programmers were responsible for writing the source code that implemented these technical requirements specifications. Table 5 provides a quick summary of the dataset used in this research.

Table 5

Dataset overview

Total Projects	3
Total Features	80
Domain	Fintech
Backend Code Language	Java
Frontend Code Language	JavaScript

Values of each of the eight predictors were manually extracted from the technical requirements specifications of the features. LOC values of both frontend and backhand, on the other hand, were extracted automatically using the "Statistic" plugin of IntelliJ IDE [14]. Table 6 shows the descriptive statistics of our complete dataset of 80 features.

Table 6

Descriptive statistics of the dataset

Variable	Min	Max	Mean
nAPI	0	7	0.667
nTPL	0	2	0.416
nGUI	4	38	13.472
nIVC	3	84	36.028
nDBQ	0	14	6.556
nTES	2	9	5.444
nSCN	2	8	4.514
nSEC	2	4	3.000
LOC	994	6160	1806.819

7. Simple Linear Regression Analysis

The primary aim of employing SLR [15] is to uncover the nature and strength of the relationship between each independent predictor and software size (LOC). This would allow us to determine how changes in one may impact the other thereby facilitating better understanding, prediction, and decision-making.

The first step in SLR analysis involved setting up the regression model with each predictor (one by one) as the independent variable and total LOC as the response or dependent variable. IBM's SPSS tool was utilized to derive the estimated coefficients (namely, the intercept and slope) which offered insights into the relationship's direction and strength. These coefficients are pivotal in understanding how changes in the predictor variable influence variations in the response variable. Additionally, the coefficient of determination (R2) – a measure of goodness of fit – was used to assess the model's overall effectiveness in explaining the variability in the response variable based on changes in the predictor.

Figures 2 – 9 depict the scatterplots (generated by SPSS) showing the relationship between each predictor and LOC. These scatterplots also contain the regression lines as well as the model equations. Each linear equation is of the form:

$$y = mx + c \qquad \text{Eq. (1)}$$

where 'y' represents the response variable, 'x' the predictor variable, 'm' the slope, and 'c' the intercept.

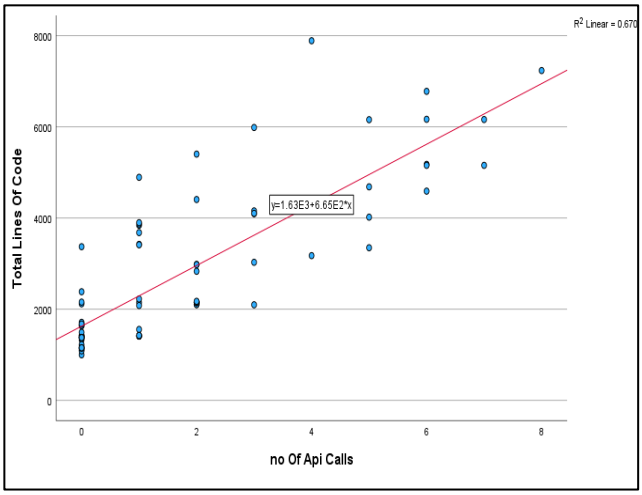

Fig. 2. Relationship with nAPI

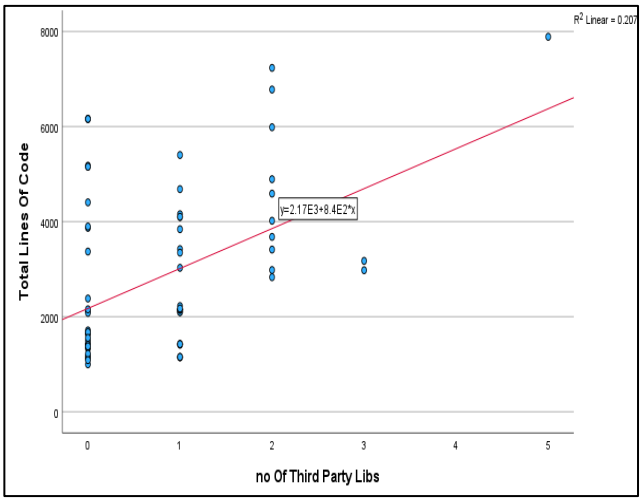

Fig. 3. Relationship with nTPL

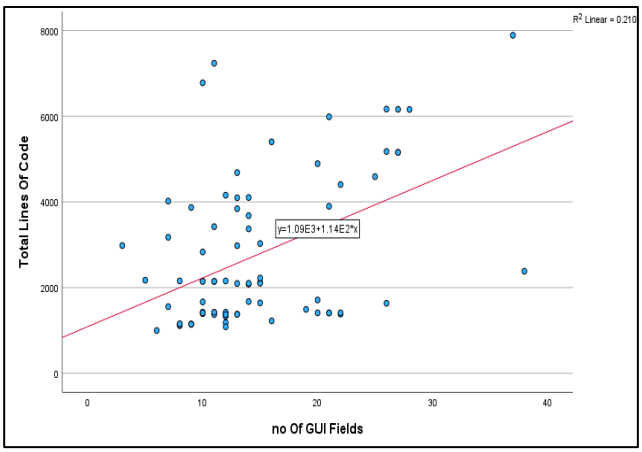

Fig. 4. Relationship with nGUI

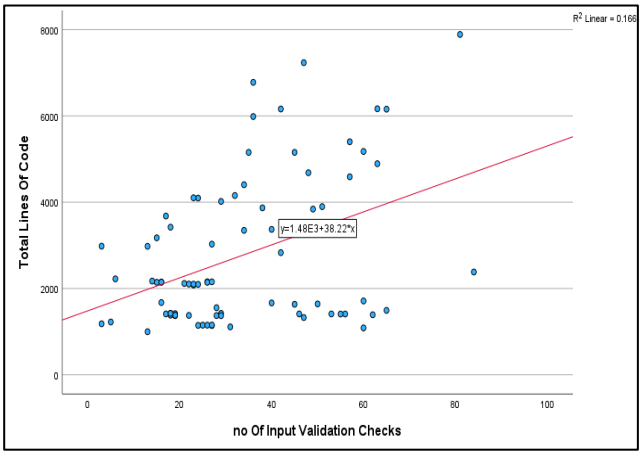

Fig. 5. Relationship with nIVC

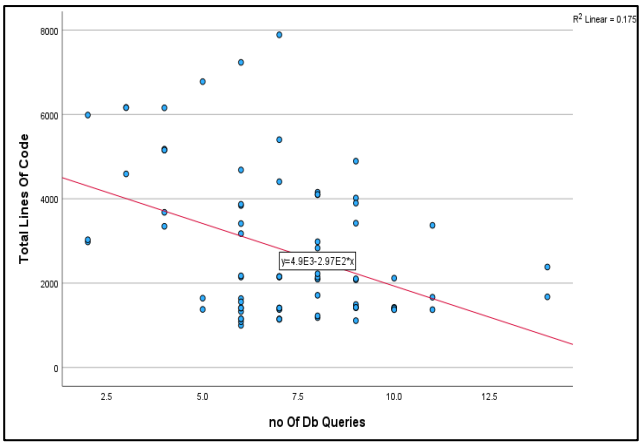

Fig. 6. Relationship with nDBQ

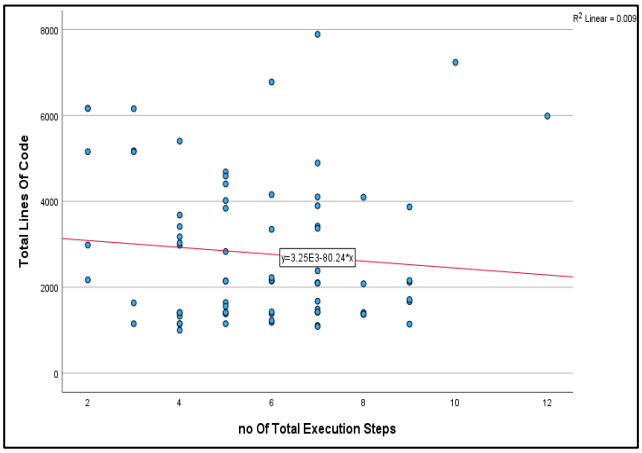

Fig. 7. Relationship with nTES

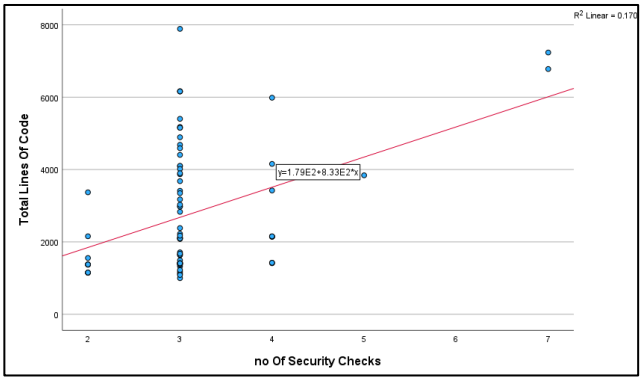

Fig. 8. Relationship with nSEC

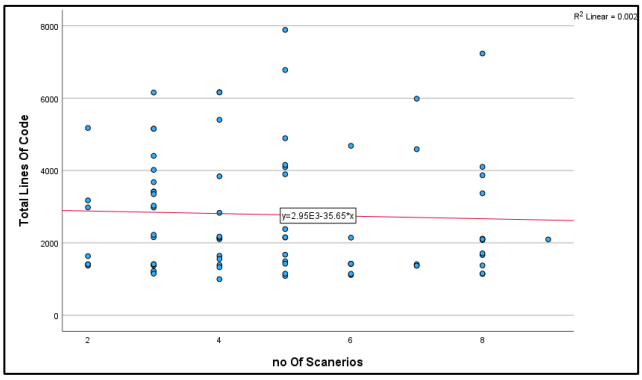

Fig. 9. Relationship with nSCN

R2 values are also shown on these scatterplots. These values indicate the proportion of variability in LOC that can be explained by each predictor. Some predictors such as nGUI and nTPL demonstrate moderate predictive power while others such as nTES and nSCN show relatively weaker relationships with LOC. The predictor nAPI has the highest R^2 value (0.670) indicating a more substantial explanatory impact as compared to the other predictors. Table 7 lists the R2 values of all eight predictors in descending order. This answers RQ2.

Table 7

The predictive strength of each predictor

Sr#	Predictor	R^2 Value
1	nAPI	0.670
2	nGUI	0.210
3	nTPL	0.207
4	nDBQ	0.175
5	nSEC	0.170
6	nIVC	0.166
7	nTES	0.009
8	nSCN	0.002

8. Multiple Linear Regression Analysis

Multiple Linear Regression (MLR) [16] is a vital tool for unravelling the complex relationships between several factors and a single outcome. It is a statistical technique used to model the relationship between a dependent variable and two or more independent variables. It extends SLR (which deals with only one independent variable) to handle situations where multiple factors influence the outcome. The general form of an MLR model is given by the following equation:

$$Y = \beta 0 + \beta 1X1 + \beta 2X2 + \ldots + \beta iXi + \varepsilon \quad \text{Eq. (2)}$$

where 'Y' represents the response variable, 'Xi' the i-th predictor variable, "βi" represents the change in Y for a one-unit change in Xi, and 'ε' is the difference between the actual value of Y and the value of Y predicted by the model.

Before building the MLR model, data pre-processing is done to clean the dataset of outliers. A commonly used statistical measure called Cook's Distance [17] is used to detect the outliers. SPSS automatically calculates Cook's Distance during the execution of the analysis. Data points with a value greater than 4/n (where n is the number of observations, in our case n = 80) are typically considered influential. High values of Cook's Distance indicate that removing the data point would significantly alter the regression coefficients or model fit, suggesting that the observation has a substantial impact on the analysis. In our case, 8 features were identified as outliers and were removed from subsequent analysis. The final model built using forward stepwise multiple linear regression (FSMLR) was, therefore, calibrated using 72 data points.

Model		Unstandardized Coefficients B	Unstandardized Coefficients Std. Error	Standardized Coefficients Beta	t	Sig.	Correlations Zero-order	Correlations Partial	Correlations Part	Collinearity Statistics Tolerance	Collinearity Statistics VIF
1	(Constant)	1385.906	59.817		23.169	<.001					
	no Of Api Calls	631.371	38.478	.891	16.409	<.001	.891	.891	.891	1.000	1.000
2	(Constant)	1014.924	123.705		8.204	<.001					
	no Of Api Calls	579.811	39.065	.818	14.842	<.001	.891	.873	.752	.846	1.182
	no Of GUI Fields	30.088	8.952	.185	3.361	.001	.507	.375	.170	.846	1.182

a. Dependent Variable: Total Lines Of Code

b. Selecting only cases for which Cook's Distance <= .05000

Fig. 10. Final Software Size Prediction Model Details

The equation below shows the final software size (LOC) prediction model which has an adjusted R2 value of 0.818:

$$LOC = 1014.924 + (579.811 * nAPI) + (30.088 * nGUI) \quad \text{Eq. (3)}$$

As is evident from this equation, adjusted R2 value, and Fig. 10, only two – nAPI and nGUI – out of the eight predictors together play a significant role in accounting for most (80%+) of the variation in LOC. The leanness and simplicity of this model offer a big advantage to project managers who have relatively less information at their disposal at the start of the project.

Two commonly used accuracy metrics – MMRE and PRED (25) [18] – were used to assess this model's performance. The MMRE value of this model is 0.19 indicating that, on average, there is around 20% difference between the software size (LOC) predicted by this model and the actual software size. PRED (25) is 0.789 implying that almost 79% of the size values estimated by this model are within 25% of the actual size values. Thus, this model performs well with respect to both accuracy metrics.

9. Model Validation

In predictive analysis, model validation is essential for judging a model's reliability and accuracy. K-fold cross-validation is a widely used technique in machine learning and statistical modeling for evaluating the performance of predictive models [19]. Unlike traditional validation methods that split the dataset into a single training and test sets, k-fold cross-validation divides the dataset into k subsets or "folds". Each fold serves as both a training as well as a testing set ensuring that every data point is used for both training and validation. This validation approach provides a more reliable estimate of model performance by averaging the results obtained from multiple iterations. Secondly, it helps to maximize the use of available data by partitioning the dataset into multiple folds.

We partitioned our dataset comprising 72 features into 6 equal folds. Thus, each fold contained 12 different features. Since SPSS does not provide complete automation of the entire k-fold validation process, each of the 6 iterations was performed separately. In each iteration, data in k-1 folds was used for model training and the remaining data was used for model testing. As a result, 6 different MLR-based software size (LOC) prediction models were generated and tested separately.

Table 8

K-fold cross-validation results

Fold #	Model Equation	Adjusted R2	MMRE	PRED (25)
Fold 1	LOC=789.301 + (nAPI*506.985) + (nGUI*47.637)	0.882	0.3605	0.25
Fold 2	LOC=875.155 + (nAPI*702.844) + (nTES*78.152) + (nTPL*235.267)	0.886	0.4463	0.09
Fold 3	LOC=1074.04 + (nAPI*573.439) + (nGUI*26.381)	0.757	0.1629	0.92
Fold 4	LOC=1050.175 + (nAPI*579.344) + (nGUI*28.047)	0.809	0.1404	1.00
Fold 5	LOC=1031.969 + (nAPI*588.477) + (nGUI*28.648)	0.810	0.0676	1.00
Fold 6	LOC=1002.532 + (nAPI*582.315) + (nGUI*30.086)	0.811	0.0417	1.00

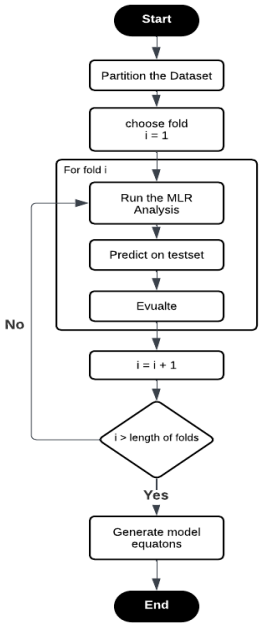

Fig. 11. K-Fold Cross-Validation Process

Fig. 11 shows a pictorial summary of the k-fold cross-validation process while Table 8 presents its results. These results help in addressing RQ3. The average MMRE value across all 6 folds is 0.203 meaning that, on average, LOC predictions are off by approximately 20% relative to the actual LOC values. Similarly, the average PRED (25) value across all 6 folds is 0.708 meaning that, on average, nearly 71% of the predicted LOC values are within 25% of the actual LOC values. Thus, both average MMRE and average PRED (25) values attest to the accuracy of our proposed approach.

10. Threats to Validity

As with any research endeavor, there are potential threats that may compromise the validity of our results. One such threat is related to the external validity of our research. It stems from the fact that we

focus on a particular set of software projects belonging to a certain domain (i.e. fintech) and implemented using certain programming languages (i.e. JavaScript and Java). The size estimation model presented here, therefore, may not be applicable as it is to other types of projects and implementation settings. Nonetheless, the repeatable process documented in this research can be used to build a software size estimation model for any type of project undertaken in any setting.

The other main threat is related to the internal validity of our results. Several measures were taken to mitigate this threat. First, we did not rely on a single source to identify our set of predictors. We consulted human experts in addition to analyzing technical documentation. Secondly, we tried to automate the measurements where possible. For instance, the LOC of all completed features was measured automatically using the "Statistic" plugin of IntelliJ IDE. Furthermore, multiple accuracy metrics (MMRE and PRED (25)) were used to judge our model's performance.

11. Conclusions and Future Work

This research addresses the common problem of inaccurate effort and cost estimation in software development projects by building a software size prediction model using information contained in projects' technical requirements specifications. The final model is built using forward stepwise multiple linear regression and is calibrated using a dataset of more than 70 completed features of actual real-life projects. Model assessment and validation results indicate that this model is fairly accurate in predicting software size (LOC) from just two predictors i.e. number of API calls and number of GUI fields. This lean and simple model is expected to assist software managers in project planning, resource allocation, and budget preparation.

This research can be extended in several different directions. First and foremost, the repeatable model-building process identified in this study can be used to prepare size estimation models for other types of projects representing a variety of domains, implemented using different programming languages, and deployed on different platforms (e.g. mobile, web, desktop, etc.). Secondly, non-linear machine learning techniques may also be explored for model building using a bigger dataset. Last, but not the least, a GUI-based user-friendly software tool can be built on top of these different size prediction models to aid project managers working on projects spanning different domains and platforms.

12. References

[1] N. Nan and D. E. Harter, "Impact of Budget and Schedule Pressure on Software Development Cycle Time and Effort", IEEE Transactions on Software Engineering, vol. 35, no. 5, pp. 624-637, Sept.-Oct. 2009, doi: 10.1109/TSE.2009.18.

[2] B. Curtis, H. Krasner, and N. Iscoe, "A field study of the software design process for large systems", Communications of the ACM, vol. 31, no. 11, pp. 1268–1287, Nov. 1988, doi: https://doi.org/10.1145/50087.50089.

[3] J. T. Dhas, "Importance of Software Sizing in Software Project Management: A Study", Italian Journal of Pure and Applied Mathematics, vol. 118, pp. 269–273, Mar. 2020

[4] B. Boehm, "Cost estimation with COCOMO II", ResearchGate, Nov. 14, 2002. https://www.researchgate.net/publication/228600814_Cost_estimation_with_COCOMO_II (accessed Mar. 08, 2025).

[5] D. Garmus, D.P. Herron "Function Point Analysis: Measurement Practices for Successful Software Projects", Addison-Wesley Information Technology Series, 2001.

[6] Y. Zheng, B. Wang, Y. Zheng, and L. Shi, "Estimation of software projects effort based on function point", 2009 4th International Conference on Computer Science & Education, Jul. 2009, doi: https://doi.org/10.1109/iccse.2009.5228317

[7] E. N. Regolin, G. A. de Souza, A. R. T. Pozo, and S. R. Vergilio, "Exploring machine learning techniques for software size estimation", 23rd International Conference of the Chilean Computer Science Society, 2003. SCCC 2003. Proceedings., Chillan, Chile, 2003, pp. 130-136, doi: 10.1109/SCCC.2003.1245453.

[8] N. A. Zakaria, A. R. Ismail, A. Y. Ali, N. H. Khalid, and N. Z. Abidin, "Software Project Estimation with Machine Learning", International Journal of Advanced Computer Science and Applications, vol. 12, no. 6, 2021. doi:10.14569/ijacsa.2021.0120685

[9] Sharma and D. S. Kushwaha, "Estimation of Software Development Effort from Requirements Based Complexity", Procedia Technology, vol. 4, pp. 716–722, 2012, doi: https://doi.org/10.1016/j.protcy.2012.05.116

[10] T. E. Ayyildiz and A. Koçyigit, "A Case Study on the Utilization of Problem and Solution

Domain Measures for Software Size Estimation", 2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Limassol, Cyprus, 2016, pp. 108-111, doi: 10.1109/SEAA.2016.13.

[11] K. Lind and R. Heldal, "A practical approach to size estimation of embedded software components", IEEE Transactions on Software Engineering, vol. 38, no. 5, pp. 993–1007, Sep. 2012, doi: 10.1109/tse.2011.86.

[12] P. R. Hill, "Practical Software Project Estimation: A Toolkit for Estimating Software Development Effort & Duration", First edition. New York: McGrawHill Education, 2011. Available: https://www.accessengineeringlibrary.com/content/book/9780071717915

[13] IBM, "Downloading IBM SPSS Statistics 29", *www.ibm.com*, Nov. 17, 2022. https://www.ibm.com/support/pages/downloading-ibm-spss-statistics-29

[14] "Statistic - IntelliJ IDEs Plugin | Marketplace", JetBrains Marketplace, Dec. 27, 2023. https://plugins.jetbrains.com/plugin/4509-statistic

[15] B. C. Gupta, I. Guttman, and K. P. Jayalath, "Simple Linear Regression Analysis", Statistics and Probability with Applications for Engineers and Scientists using MINITAB, R and JMP, John Wiley & Sons, Ltd, 2020, pp. 622–692. doi: https://doi.org/10.1002/9781119516651.ch15.

[16] B. C. Gupta, I. Guttman, and K. P. Jayalath, "Multiple Linear Regression Analysis", Statistics and Probability with Applications for Engineers and Scientists using MINITAB, R and JMP, John Wiley & Sons, Ltd, 2020, pp. 693–756. doi: https://doi.org/10.1002/9781119516651.ch16.

[17] R. D. Cook, "Detection of Influential Observation in Linear Regression", Technometrics, vol. 42, no. 1, pp. 65–68, Feb. 2000, doi: https://doi.org/10.1080/00401706.2000.10485981.

[18] B. A. Kitchenham, L. M. Pickard, S. G. MacDonell, and M. J. Shepperd, "What accuracy statistics really measure", IEE Proceedings - Software, vol. 148, no. 3, p. 81, 2001, doi: https://doi.org/10.1049/ip-sen:20010506.

[19] D. Berrar, "Cross-Validation", Encyclopedia of Bioinformatics and Computational Biology, vol. 1, pp. 542–545, 2019, doi: https://doi.org/10.1016/b978-0-12-809633-8.20349-x

