Predicting Software Size and Effort from Code Using Natural Language Processing

Samet Tenekeci[1,*], Hüseyin Ünlü[1,*], Emre Dikenelli[1], Uğurcan Selçuk[1], Görkem Kılınç Soylu[2] and Onur Demirörs[1]

[1]*İzmir Institute of Technology, Gülbahçe, İzmir, 35430, Türkiye*
[2]*İzmir University of Economics, Balçova, İzmir, 35330, Türkiye*

Abstract

Software Size Measurement (SSM) holds a crucial role in software project management by facilitating the acquisition of software size, which serves as the primary input for development effort and schedule estimation. However, many small and medium-sized companies encounter challenges in conducting objective SSM and Software Effort Estimation (SEE) due to resource constraints and a lack of expert workforce. This often leads to inaccurate estimates and projects exceeding planned time and budget. Hence, organizations need to perform objective SSM and SEE with minimal resources and without relying on an expert workforce. In this research, we introduce two exploratory case studies aimed at predicting the functional size (COSMIC and Event-based size) and effort of software projects from the code using a deep-learning-based NLP model: CodeBERT. For this purpose, we collected and annotated two datasets consisting of 4800 Python and 1100 C# functions. Then, we trained a classification model to predict COSMIC data movements (entry, exit, read, write) and four regression models to predict Event-based size (interaction, communication, process) and effort. Despite utilizing a relatively small dataset for model training, we achieved promising results with an 84.5% accuracy for the COSMIC size, 0.13 normalized mean absolute error (NMAE) for the Event-based size, and 0.18 NMAE for the effort. These findings are particularly insightful as they demonstrate the practical utility of language models in SSM and SEE.

Keywords

Software size measurement, Effort estimation, Artificial intelligence, Natural language processing

1. Introduction

A software project achieves success when it meets customer expectations within the agreed-upon timeframe and budget [1]. Therefore, accurate estimation of effort is crucial in project management for predicting schedules and costs. In this context, Software Size Measurement (SSM) plays a pivotal role in software project management by providing a fundamental input for effort and schedule estimation, offering a significant business advantage [2, 3]. Therefore, reliable SSM is of great importance.

SSM methodologies can be broadly categorized into formal methods and expert opinions [4]. One such standardized formal method is Functional Size Measurement (FSM) [5], which presents an objective approach to measurement, enhancing model estimation, internal processes, and benchmarking [6, 7, 8]. Additionally, FSM facilitates tracking project scope changes, negotiating agreements between suppliers and acquirers, and improving organizational processes by standardizing performance and quality metrics [9].

COSMIC, recognized as a second-generation FSM method and an ISO standard, stands out as one of the most widely adopted FSM approaches [10, 11]. The measurement process revolves around calculating data movements, including Entry (E), Exit (X), Read (R), and Write (W). While COSMIC FSM has demonstrated success in traditional monolithic architectures, the transition from monolith architectures to more distributed and service-oriented architectures has prompted the exploration of new size measurement methods compatible with these innovative software structures. One such method

IWSM-MENSURA, September 30 – October 4, 2024, Montréal, Canada
*Corresponding author.
samettenekeci@iyte.edu.tr (S. Tenekeci); huseyinunlu@iyte.edu.tr (H. Ünlü); emredikenelli@iyte.edu.tr (E. Dikenelli); ugurcanselcuk@iyte.edu.tr (U. Selçuk); gorkem.soylu@iue.edu.tr (G. Kılınç Soylu); onurdemirors@iyte.edu.tr (O. Demirörs)
0000-0001-8875-4111 (S. Tenekeci); 0000-0001-9906-6066 (H. Ünlü); 0000-0002-7047-0556 (G. Kılınç Soylu); 0000-0001-6601-3937 (O. Demirörs)

is event-based size measurement, which proposes counting events such as process, communication, and interaction events [12, 13, 14].

While literature showcases successful applications of FSM-based prediction methods in agile organizations [7, 15, 16], challenges persist [17, 18]. Notably, despite evidence of effectiveness, agile organizations generally underutilize FSM-based prediction methods [19, 20, 21]. This issue is particularly prominent in many small and medium-sized software companies where limited resources and a shortage of expert workforce hinder the proper execution of objective SSM and Software Effort Estimation (SEE). Even as organizations struggle to implement established size measurement methods like COSMIC, it is evident that integrating innovative methods such as event-based size measurement into their processes will pose even greater challenges. Consequently, project estimates are often rushed, superficial, and inadequate, leading to projects exceeding planned timelines and budgets. Hence, there is a need to conduct objective SSM and SEE with minimal resources and without relying solely on expert manual workforce [21].

In recent years, advancements in deep learning and natural language processing (NLP) have led to many new developments in software engineering (SE), similar to trends in other fields. In SE, traditional models based on n-grams, Support Vector Machine (SVM), Gaussian Naive Bayes, Part-of-Speech (PoS) tagging, or Named Entity Recognition (NER) are being replaced by neural network-based Language Model Models (LLMs) [22, 23]. These LLMs have three main strengths: state-of-the-art Transformer architectures with masking and attention mechanisms, context awareness, and transfer learning capabilities. This allows LLMs to be easily run on different downstream tasks, providing accurate results even with limited labeled data. Through minor modifications and simple fine-tuning, LLMs can achieve human-level performance in various SE tasks like SSM [24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34], SEE [24, 35, 36, 37, 38, 39], requirement extraction and classification [40, 41, 42, 43, 44, 45, 46, 47]. However, there is not much research on using these models for tasks like size measurement and effort estimation from code.

In this research, we perform two exploratory case studies [48] to predict software projects' functional size and effort from the code using a transformer-based, pre-trained language model: CodeBERT [49]. In Case Study 1, we train CodeBERT with a set of functions derived from the CodeSearchNet corpus [50] and the related COSMIC size. In Case Study 2, we train CodeBERT with a set of functions from a real-life game project and its related event-based size and effort. We achieve promising prediction results in both studies. In Case Study 1, we predict the COSMIC size with 84.5% accuracy. In Case Study 2, we predict the event-based size and effort with 0.13 and 0.18 normalized mean absolute error (NMAE), respectively. These results show the applicability of using NLP models for predicting the size and effort from the code.

The remainder of this paper is structured as follows. Section 2 gives the background. Section 3 summarizes the related work. Section 4 explains the research method. Section 5 gives the results and discusses our findings. Section 6 discusses the validity threats. Lastly, Section 7 concludes the paper.

2. Background

In this section, we briefly mention important background, including the size measurement methods and large language models used in this study.

2.1. COSMIC Functional Size Measurement Method

COSMIC Functional Size Measurement Method is based on counting the Data Movements (DM) [10]. Following this purpose, the method suggests identifying first the Functional User Requirements (FUR), then decomposing FURs into the Functional Processes (FP), and after that, measuring each functional process by identifying its DMs. These DMs are defined as the flow of data groups belonging to a single Object of Interest (OOI). In other words, a flow of data group can be defined as DM if it is related to a single OOI. DMs can appear in four forms: Entry (E), Exit (X), Read (R), and Write (W). An Entry moves a data group from a functional user into the functional process. An Exit moves a data group from a

functional process to the functional user. A Write moves a data group from inside a functional process to the persistent storage area. A Read moves a data group from persistent storage into the functional process. Each data movement is counted as 1 COSMIC Function Point (CFP). The size of the software is the sum of data movements accumulated over all functional processes.

2.2. Event-based Size Measurement Method

The Event-based Size Measurement method categorizes the events into three: (1) Process Event, (2) Communication Event, and (3) Interaction Event [12, 13, 14]. Process Event covers the calculations required during the flow and the events that result from decision-making. It mainly covers back-end layer events. Communication Event covers events that occur as a result of publishing, subscribing, and messaging. It mainly covers back-end communication events between microservices. Interaction Event covers events that occur as a result of triggers made by the user to the system during the flow and any output from the system to the user. It mainly covers the front-end layer events. The method suggests modeling the project requirements using the event-driven process modeling techniques and then counting the number of events that occur. As a measurement unit, the method proposes the "Event Points" concept. As a result of measurement, the size of a specific requirement can be defined as the total number of events (Event Points) it has. Consequently, the size of a project is the total of the size of its requirements.

2.3. BERT and CodeBERT

BERT (Bidirectional Encoder Representations from Transformers) [51] is a pre-trained, masked language model based on Transformer architecture and self-attention mechanism. It is pre-trained on a large corpus of book and Wiki texts to understand the context and meaning of words and sentences in natural language. BERT achieves state-of-the-art performance in many downstream NLP tasks, such as text classification, named entity recognition, and sentiment analysis. Moreover, through transfer learning and fine-tuning, it can be adapted to various tasks in different domains including software engineering.

CodeBERT [49] extends BERT by adding programming language (PL) understanding to its natural language (NL) capabilities. It is a bimodal pre-trained model for PL and NL that learns general-purpose representations to support downstream NL-PL applications such as natural language code search and code documentation generation. CodeBERT uses the same Transformer-based neural architecture as BERT. CodeBERT utilizes a hybrid objective function that combines masked language modeling (MLM) with replaced token detection (RTD) to train the model on both bimodal NL-PL pairs and unimodal code data.

3. Related Work

Measuring the functional size of software from requirements or code has great importance in software engineering. Traditional techniques are often prone to error and require significant time and effort. Thus, the focus of the recent studies shifts towards automating this process. This section briefly reviews the existing work on different approaches to automate SSM and SEE.

In [31], the authors present the results of a structured survey that is designed to explore opportunities, challenges, and obstacles associated with deriving functional size automatically from the software project code. Among the FSM methods evaluated, COSMIC was the most frequently preferred by experts. The survey also revealed that experts have mixed opinions about the importance of automated size measurement from the source code. While 42% of participants found it important, their expectations from it included increased reliability, speed, and decision-making support, and the majority were neutral or skeptical. One of the key concerns stated by the participants was accuracy.

[52] introduces a tool named COSMIC Solver, which automates FSM of Java Business Applications (JBA) by extracting textual representations of UML sequence diagrams from execution traces of a JBA and tagging them using AspectJ. The automation accuracy of the tool is calculated to be 77%. [53] introduces

the COSMIC APP to automate the measurement process to approximate the COSMIC functional size of a project. The tool utilizes the sequence diagram of the software after reverse engineering it from the given code using a third-party tool. The accuracy rate in predicting the functional size achieved by the COSMIC APP is 87.8%. [54] automates COSMIC FSM for Java Web applications using the Spring Web MVC framework. Using the CFP4J library, Spring MVC methods are mapped to COSMIC components. The method achieves 97.6% accuracy.

To address the limitations arising from rule-based learning, researchers and practitioners have turned to Artificial Intelligence (AI) techniques, specifically Neural Networks (NN), to enhance size prediction. AI techniques have been extensively adopted to enhance various aspects of software development processes and have shown notable success in recent years. Deep learning models, in particular, have been effectively employed for FSM from early software artifacts such as analysis and design models.

In [26], the authors present a method to predict COSMIC functional size from text representing use-case names. The method is based on the DEEP-COSMIC-UC model, which is a multi-layer convolutional neural network that is pre-trained with different word embeddings. The normalized mean absolute error (NMAE) of the model for prediction of functional size is calculated as 0.39. [32] uses a model named RE-BERT for COSMIC-based functional size classification in agile software development. RE-BERT is trained with a generic BERT model over requirement engineering domain texts to be used for COSMIC functional size classifications. The highest prediction accuracy reached is 78.97%. [29] utilizes NLP and pattern matching to predict the size of a functional requirement. The accuracy of prediction ranges between 70% and 95%. [34] modifies an existing FSM automation tool for IoT by integrating NLP with it. Two learning models, Naive Bayes and neural networks, were used in the study, and the accuracy was reported to be 53% and 82.3%, respectively. In [27], the authors built three functional size models using Support Vector Regression (SVR), Random Forest (RF), and Neural Network (NN) approaches to predict software functional size by Function Point Analysis (FPA). The achieved coefficient of determination values, R2, are given for the three models as 0.982, 0.949, and 0.972. [33] presents a case study to predict functional size from the requirements by using NLP. The authors fine-tuned BERT and BERT_SE with a set of user stories and their respective functional sizes. In total size prediction, they achieved 72.8% and 74.4% accuracy with BERT and BERT_SE, while in data movement-based size prediction, they achieved 87.5% and 88.1% average accuracy, respectively.

With the rapid advancement in AI, numerous models have begun to explore their application in SE, including size prediction. Although a substantial effort has been made to use NLP techniques for SSM, to our knowledge, directly predicting functional size from code remains unexplored.

4. Research Method

We follow the case study research method proposed by [46] in both case studies. Yin suggests that a typical case study design involves the following steps.

4.1. Case Study Design

In this research, our primary goal is to explore how deep learning-based NLP methods are successful in predicting the size and effort of software projects from the code. More specifically, we aim to evaluate the success of the CodeBERT model in predicting COSMIC functional size, Event-based size, and effort from the code. For this aim, we derive the following research questions:

- **RQ1:** How successful is NLP-based COSMIC functional size prediction from the code?
- **RQ2:** How successful is NLP-based Event-based size prediction from the code?
- **RQ3:** How successful is NLP-based effort prediction from the code?

4.1.1. Case Selection Criteria

As the first step of this study, we aim to prepare datasets for model training. For this purpose, we define the following case selection criteria:

- **Criterion 1:** The case should include the code of the projects.
- **Criterion 2:** The code should be measurable by COSMIC size.
- **Criterion 3:** The code should be measurable by Event-based size.
- **Criterion 4:** The code's size and effort unit should be at the smallest level, such as function level.
- **Criterion 5:** The code should include documentation or comments at the smallest level to provide ease of measurability.

To meet all the defined criteria, we perform two exploratory case studies. In Case Study 1 (Criteria 1, 2, 5), we aim to answer RQ1. In Case Study 2 (Criteria 1, 3, 4, 5), we aim to answer RQ2 and RQ3. Hence, the conduction of these two case studies differs.

4.1.2. Data Collection Procedures

We use CodeSearchNet corpus [50] to construct the dataset for Case Study 1. The initial dataset comprises 2 million (comment, code) pairs from open-source libraries. Concretely, a comment is a top-level function or method comment (e.g., docstrings in Python), and code is an entire function or method. The dataset contains Python, Javascript, Ruby, Go, Java, and PHP code. Although we use only Python codes (457K instances), the other languages can be easily included in the experiments.

We use a game development project named "Green Balance" to construct the dataset for Case Study 2. Green Balance is a city building and management simulation developed by a professional software company in 1 year using C# programming language and Unity game engine. The dataset constructed based on Green Balance includes 1100 C# functions in 97 namespaces.

4.2. Conduction of the Case Studies

The conduction of the case study consists of three stages: (i) preparing the datasets, (ii) training the models using the datasets, and (iii) evaluating the trained models.

4.2.1. Preparing the Datasets

In Case Study 1, we use a rule-based data labeling algorithm to assign categorical class labels to Python codes based on their docstring texts. After the labeling, the docstrings and comments are removed from the code. This process is referred to as "dedoc" for the remainder of this article. The dedoc step is essential for the model to learn solely from the code (i.e., unimodal approach) during training. The class labels include the COSMIC data movements, which are 0 (W), 1 (R), 2 (X), and 3 (E). Using the automated labeling algorithm, we obtain 1100 instances for each class. Additionally, we create a disjoint dataset of 400 manually labeled code samples (i.e., 100 samples from each class) to test the performance of the trained model.

In the process of filtering the data, there are significant differences in labeling criteria. For example, while labeling for the "Write" COSMIC function is based solely on the presence of the word "write," more comprehensive sets of words have been used for the "Entry" and "Read" functions. This strategy has been adopted to maintain a balanced data distribution, which is at least 1100 data points in each class. The descriptive statistics of the dataset are given in Table 1, and a sample dataset is presented in Table 2.

In Case Study 2, we manually measure the size and effort of each C# function and annotate them with the corresponding event-based size (interaction, communication, process) and effort values. The size and effort values are continuous numbers in this case. The effort values denote the workforce needed in person-hour to implement the corresponding function. All values are measured at the function level and grouped by namespace for evaluation. The descriptive statistics of the dataset are given in Table 3, and a sample dataset is presented in Table 4.

Algorithm 1 Automated Labeling Algorithm

if the word "write" appears in the docstring **then**
    assign the label 0 (W).
end if
if the words "read" and "data", "read" and "from", "read" and "file", "read" and "database", "data" and "from" and "file", or "data" and "from" and "database" appear in the docstring **then**
    assign the label 1 (R).
end if
if the word "send" or "sends" appears in the docstring **then**
    assign the label 2 (X).
end if
if the words "get" and "from" or "gets" and "from" or "message" and "from" appear in the docstring **then**
    assign the label 3 (E).
end if

Table 1
Descriptive statistics of the dataset for Case Study 1

	Training set	Test set (Automatic)	Test set (Manual)
Number of samples	3520	880	400
Min number of words	4	5	5
Max number of words	118	90	1453
Mean number of words	27.1	26.6	50.8
Std number of words	15.7	15.4	98.4

Table 2
Sample dataset for Case Study 1

Code	Docstring	Data Movement
`def _schema_to_json_file_object(self, schema_list, file_obj):` `  json.dump(schema_list, file_obj, indent=2, sort_keys=True)`	Helper function for schema_to_json that takes a schema list and file object and writes the schema list to the file object with json.dump	0 (W)

`def read_json(filename, mode='r'):`
`  with open(filename, mode) as filey:`
`  data = json.load(filey)`
`  return data`
	Read_json reads in a json file and returns the data structure as dict.	1 (R)

`def _send_coroutine():`
`  with PoolExecutor() as executor:`
`  while True:`
`  msg = yield`
`  future = executor.submit(msg.send)`
`  future.add_done_callback(_exception_handler)`
	Creates a running coroutine to receive message instances and send them in a futures executor.	2 (X)

`def mouse_move(self, event):`
`  if (self.ui.tabWidget.currentIndex() == TabWidget.NORMAL_MODE):`
`  self.posX = event.xdata`
`  self.posY = event.ydata`
`  self.graphic_target(self.posX, self.posY)`
	The following gets back coordinates of the mouse on the canvas.	3 (E)

Table 3
Descriptive statistics of the dataset for Case Study 2

	Interaction	Communication	Process	Total	Effort
Minimum	0	0	4	5	0.19
Maximum	86	17	286	372	30.55
Mean	5.36	0.21	54.71	60.28	4.21
Median	0	0	34	37	2.40
Std	14.49	1.74	58.84	68.21	4.75
Total	520	20	5307	5847	408.17

Table 4
Sample dataset for Case Study 2

Code	i	c	p	Total	Effort

`public void AddAllResDebug() {`
`  rawResource.increaseYCoin(10000);`
`  rawResource.increaseYp(10000);`
`  rawResource.increaseCement(10000);`
`  rawResource.increaseWire(10000);`
`  rawResource.increaseChemical(10000);`
`  rawResource.increaseMechanicalPart(10000);`
`  rawResource.increaseFood(500);`
`  seedResource.increaseRegularSeedAmount(10);`
`  seedResource.increasePremiumSeedAmount(10);`
`}`
	2	0	9	11	0.10

`public IEnumerator CheckToken() {`
`  var uwr = new UnityWebRequest(APIURI +`
`  APICONTROLLERAUTHMANAGEMENT + "CheckToken", "GET");`
`  uwr.SetRequestHeader("Content-Type", "application/json");`
`  uwr.SetRequestHeader("Authorization", "Bearer" + bearerToken);`
`  uwr.downloadHandler = new DownloadHandlerBuffer();`
`  yield return uwr.SendWebRequest();`
`  if (uwr.responseCode != 200) {  // not valid`
`  isBearerTokenValid = false;`
`  yield return false;`
`  }`
`  else { // valid`
`  isBearerTokenValid = true;`
`  yield return true;`
`  }`
`}`
	0	1	9	10	1.00

4.2.2. Training the Models

The original CodeBERT model has 12 hidden RoBERTa layers with 768 hidden states, 12 attention heads, and one classification head at the end. When training our models, we keep the reference architecture as is, except for the last layer, which is modified depending on the type of our tasks (see Figure 1). Case Study 1 has a multiclass classification task, while Case Study 2 has a set of regression tasks. Accordingly, we modify the objective function in the last layer. In both cases, the input is an array of codes (i.e., methods). The only target in Case Study 1 is the class labels, while there are four different targets (interaction, communication, process, and effort) in Case Study 2. Thus, we train four different regression models for Case Study 2. In both cases, we set the maximum input length as 512 tokens and truncate the longer codes. We keep the default optimizer as AdamW. However, we use different hyperparameters for each case, taking into account the task type and dataset features. We use $lr = 2e-5$, $eps = 1e-8$, $epochs = 5$, $batch\_size = 64$, and $num\_labels = 4$ (cross-entropy loss) in Case Study 1. On the other hand, we use $lr = [1e-4, 2e-5]$, $eps = 1e-8$, $epochs = 4$, $batch\_size = 16$, and $num\_labels = 1$ (mean-square error loss) in Case Study 2.

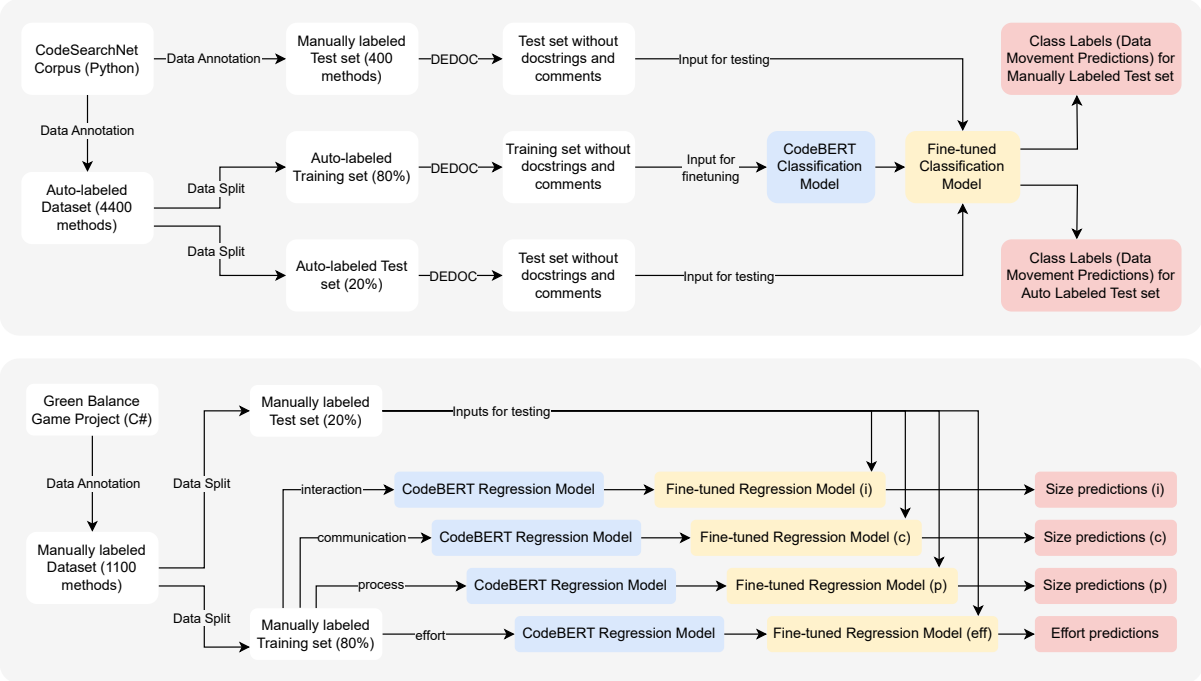

Figure 1: Model training workflows (Top: Case Study 1, Bottom: Case Study 2)

4.2.3. Evaluation

We perform an 80/20 train/test split with 5-fold cross-validation in both case studies. We use various metrics for performance evaluation (see Table 5). In Case Study 1, we evaluate the performance in terms of predicting the data movement type on both automatically labeled and manually labeled test sets. We take accuracy (ACC) as the main performance comparison metric. We also investigate the precision, recall, F1-score, and the confusion matrix.

Table 5
Definitions of evaluation metrics

Regression Metrics	Classification Metrics
$MAE = \frac{1}{n}\sum_{i=1}^{n}\lvert y_i - \hat{y}_i\rvert$	$Precision = \frac{TP}{TP+FP}$
$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$	$Recall = \frac{TP}{TP+FN}$
$MMRE = \frac{1}{n}\sum_{i=1}^{n}\frac{\lvert y_i - \hat{y}_i\rvert}{\hat{y}_i}$	$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$
$NMAE = \frac{\sum_{i=1}^{n}\lvert y_i - \hat{y}_i\rvert}{\frac{1}{n}\sum_{i=1}^{n}\hat{y}_i}$	$ACC = \frac{TP+TN}{TP+TN+FP+FN}$

In Case Study 2, we evaluate the model performance in terms of predicting the size and effort at the namespace level. We take mean-absolute error (MAE) and normalized mean-absolute error (NMAE) as the main performance comparison metrics. We also investigate the mean magnitude of relative error (MMRE) and PRED(30) for effort estimations. Moreover, we conduct a linear regression analysis based on total event size and a multiple linear regression analysis based on interaction, communication, and process events. Then, we evaluate CodeBERT's performance against these baselines.

5. Results and Discussions

This section summarizes and discusses the results of two exploratory case studies answering our three research questions.

5.1. Answering RQ1

In Case Study 1, we predict the COSMIC size of automatically labeled and manually labeled codes in terms of data movements (W, R, X, E). Since the task is a multiclass classification problem, we evaluate the performance of the model with precision, recall, F1-score, and accuracy metrics (see Table 6). We also plot the confusion matrices to investigate the misclassified data movements.

Table 6
Performance evaluation for Case Study 1

	Automatically Labeled Test Set			Manually Labeled Test Set		
	precision	recall	F1-score	precision	recall	F1-score
W	0.7725	0.7409	0.7564	0.8800	0.8800	0.8800
R	0.7162	0.7227	0.7195	0.8065	0.7500	0.7772
X	0.7824	0.7682	0.7752	0.8796	0.9500	0.9135
E	0.6753	0.7091	0.6918	0.8081	0.8000	0.8040
Accuracy	—	—	**0.7352**	—	—	**0.8450**
Macro Avg	0.7366	0.7352	0.7357	0.8435	0.8450	0.8437
Weighted Avg	0.7366	0.7352	0.7357	0.8435	0.8450	0.8437

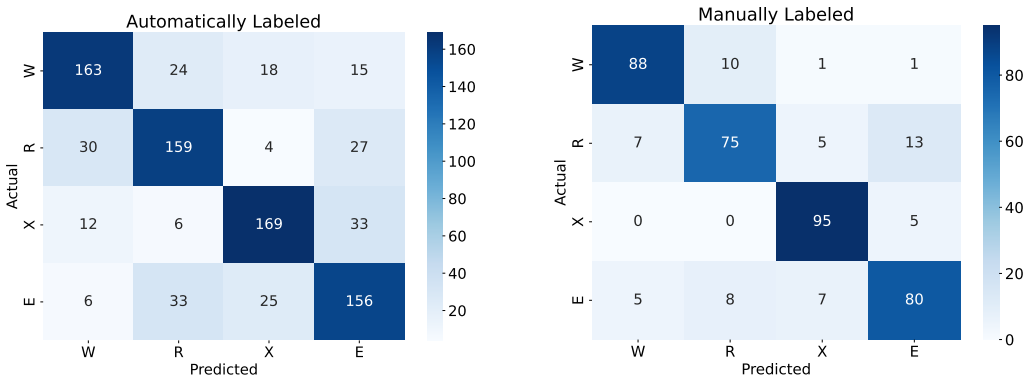

Figure 2: Confusion matrices for Case Study 1

Our classification model achieves 73.5% and 84.5% accuracies on automatically and manually labeled test sets, respectively. Accordingly, we can say that the model predictions improve significantly when the labeling is more reliable. On the other hand, no significant difference is observed between precision and recall. Confusion matrices in Figure 2 show that the model tends to misclassify W as R, R as W or E, X as E, and E as R. This may be related to that we are disregarding the possibility of functions involving multiple data movements (i.e., multiple labels) in this case study. While this assumption may be debated, it becomes reasonable when considering the Single Responsibility Principle (SRP) [55].

5.2. Answering RQ2

In the size prediction phase of Case Study 2, we predict interaction, communication, and process events. As the dataset includes zero values, we evaluate the success of the predictions using MAE and NMAE metrics (see Table 7).

Table 7
Performance evaluation for Case Study 2 - Size prediction results

	MAE	NMAE
Interaction	5.91	1.10
Communication	0.37	1.80
Process	5.12	0.09
Total Event	7.98	0.13

According to Hastings and Sajeev [56], predictions with an MMRE of less than 0.20 can be considered predictive, while an MMRE between 0.20 and 0.50 is acceptable, and an MMRE greater than 0.50 is unacceptable. Applying these thresholds to NMAE values, we find that error rates for interaction and communication events fall into the unacceptable category, whereas predictions for process events remain predictive. This is likely because interaction and communication events occur infrequently, considering the nature of the game projects. Essentially, there are many instances of zero values, leading to an imbalance in data distribution and a high standard deviation for these events in the dataset. However, the overall impact of interaction and communication events on the total number of events seems minimal, given the closeness of the total event NMAE to the process event NMAE.

5.3. Answering RQ3

In the effort prediction phase of Case Study 2, we first perform linear regression based on the total event size and multiple linear regression based on interaction, communication, and process events. Then, we evaluate the success of CodeBERT compared to regression-based results (see Table 8).

Table 8
Performance evaluation for Case Study 2 - Effort prediction results

	Linear Regression	Multiple Linear Regression	CodeBERT
MAE	2.66	2.31	**0.76**
NMAE	0.63	0.55	**0.18**
MMRE	1.72	1.59	**0.21**
PRED(30)	0.19	0.24	**0.69**

The MMRE $= 0.21$ obtained for effort estimation is just above the predictive threshold and is acceptable. This result is very promising as it shows that CodeBERT can estimate effort without using any size measures. On the other hand, the regression-based approaches fall behind CodeBERT, yielding unacceptable predictions. It is worth noting here that an event-based approach may not be suitable for measuring the size of game projects. The event-based size measurement method [12, 13, 14] is primarily intended for use in innovative software architectures such as microservices. However, since measuring a game project using the COSMIC sizing method is not feasible, events in the game project have been counted within the scope of Case Study 2. Consequently, there continues to be a need for methods to measure the size of game projects, considering the building blocks of game projects.

5.4. Comparing Our Results with Related Work

We compare our results with three studies [52, 53, 54] with 77%, 87.8%, and 97.6% accuracy. They all attempt to predict the functional size from code. The key point here is that all three of these studies require an intermediate mapping step to generate UML sequence diagrams or mapping rules and perform rule-based measurements based on the pre-defined rules. For example, [52] uses Javaagent and AspectJ pointcuts, which are only available in Java Business Applications, to generate textual representations of sequence diagrams. Similarly, [53] uses a reverse engineering tool called SequenceDiagram, which is only available in IntelliJ IDEA, to generate sequence diagrams from Java code. [54] uses the controller

annotations in the Spring MVC framework to create mapping rules. Therefore, these methods rely heavily on specific programming languages, tools, or frameworks, greatly restricting their ability to generalize.

On the other hand, our method performs measurement directly from the source code using NLP. Although we use a rule-based labeling algorithm in Case Study 1, our predictive model is based on deep learning. Such an approach is known to be more flexible and generalizable. To the best of our knowledge, this is the first study in the literature that predicts the size and effort from code using NLP. Moreover, our model demonstrates high performance in the prediction of data movements, achieving accuracy rates of 73.5% and 84.5%. In terms of error rates, only [52] reported an MMRE $= 0.21$ (as an average of 8 applications). We achieved a better NMAE $= 0.13$ and the same MMRE $= 0.21$ in Case Study 2.

6. Threats to Validity

In this section, we discuss two potential validity threats: (1) reliability of the ground truth and (2) dataset size. The reliability of the measurements plays an essential role in the success of the trained models, as the ground-truth labels provided to the model should be accurate. In Case Study 1, given the fact that measurements assigned by a rule-based labeling algorithm may not be as reliable as measurements made by an expert, it is arguable to use an automatic labeler to annotate the dataset. However, considering the need for large amounts of labeled data in model training, such automated labeling solutions can be very useful, especially when it is empowered with quality control workflows [57]. Two expert measurers with at least three years of COSMIC experience manually measured 100 samples in each class to validate the quality of automatically generated labels. Then, the performance of the model trained with the automatically labeled dataset is tested on the manually labeled dataset. The test results revealed that our model predicted manual measurements with 84.5% accuracy. This result indicates promising outcomes from the exploratory Case Study we conducted. In Case Study 2, this threat did not arise since all measurements were conducted by an expert who developed the game and verified by an expert who developed the event-based measurement method. As the second validity threat, our datasets' size can be argued to be a small input for training the models. However, in this exploratory case study, we aimed to explore the applicability of predicting the size and effort from the code using NLP models. We obtained promising results by training the models with such a limited dataset. Therefore, this study showed us the applicability of NLP models in predicting the size and effort from the code. We believe that the precision of the prediction will be improved by increasing the size of the training dataset.

7. Conclusion

SSM plays a vital role in software project management, serving as a fundamental input for estimating effort and schedules. However, many small and medium-sized software companies face challenges in prioritizing objective SSM and SEE due to resource constraints and a shortage of expert personnel. As a result, they often end up with inadequate estimates, leading to projects exceeding planned timelines and budgets. Therefore, it is imperative for organizations to conduct objective SSM and SEE with minimal resources and without relying solely on expert workforce involvement.

In this study, we conduct two exploratory case studies to investigate the applicability of using CodeBERT to predict the size and effort of software projects using the code as input. Despite utilizing limited datasets, our main objective is to assess the applicability of these models. Achieving an 84.5% accuracy for the COSMIC size, 0.13 NMAE for the Event-based size, and 0.18 NMAE for the effort estimation, we proved that NLP models can be useful in SSM and SEE. Moreover, we revealed some possible open research areas:

- In this study, we use relatively small datasets to train the models. Therefore, training the models with larger datasets can achieve more accurate size predictions.

- In Case Study 1, we use a rule-based labeling algorithm to measure the COSMIC size. We believe that the accuracy of the model could be improved by using a dataset measured by experts to provide a more reliable input for training the model.
- In Case Study 1, we only use the dataset from the Python language. The study can be expanded to include datasets from different programming languages to broaden the scope of the research dataset.
- In Case Study 2, we trained the model by labeling the code within a specific scope of the project. The remaining code can also be included, allowing for training the model with a larger dataset.
- There continues to be a need for software size measurement methods specific to game software.

Supplementary Material

The source code and datasets are available at https://github.com/smtnkc/codebert-ssm

References

[1] The Standish Group, CHAOS 2020 Beyond Infinity, Technical Report, 2020.

[2] A. Abran, Software project estimation: the fundamentals for providing high quality information to decision makers, John Wiley & Sons, 2015.

[3] M. Salmanoğlu, K. Öztürk, S. Bağrıyanık, E. Ungan, O. Demirörs, Benefits and challenges of measuring software size: early results in a large organization, in: 25th International Workshop on Software Measurement and 10th International Conference on Software Process and Product Measurement, IWSM-Mensura, 2015.

[4] M. Jørgensen, B. Boehm, S. Rifkin, Software development effort estimation: Formal models or expert judgment?, IEEE software 26 (2009) 14–19.

[5] M. Usman, E. Mendes, F. Weidt, R. Britto, Effort estimation in agile software development: a systematic literature review, in: Proceedings of the 10th international conference on predictive models in software engineering, 2014, pp. 82–91.

[6] L. Buglione, S. Trudel, Guideline for sizing agile projects with COSMIC, Proceedings of the IWSM/MetriKon/Mensura (2010).

[7] C. Commeyne, A. Abran, R. Djouab, Effort estimation with story points and COSMIC function points-an industry case study, Software Measurement News 21 (2016) 25–36.

[8] A. Abran, J.-M. Desharnais, M. Zarour, O. Demirörs, Productivity-based software estimation models and process improvement: an empirical study, Int. J. Adv. Softw 8 (2015) 103–114.

[9] B. Ozkan, O. Turetken, O. Demirors, Software functional size: For cost estimation and more, in: Software Process Improvement: 15th European Conference, EuroSPI 2008, Dublin, Ireland, September 3-5, 2008. Proceedings 15, Springer, 2008, pp. 59–69.

[10] COSMIC Measurement Manual v5.0, The Common Software Measurement International Consortium, 2021. Available at https://cosmic-sizing.org/measurement-manual/.

[11] A. Abran, Automating functional size measurement–a survey, in: UKSMA/COSMIC Conference 2011-22nd Annual conference on Metrics and Estimating: hosted in collaboration with COSMIC, 2011.

[12] T. Hacaloğlu, Event Points: A Software Size Measurement Model, Ph.D. thesis, Middle East Technical University, 2021.

[13] T. Hacaloglu, O. Demirors, An exploratory case study using events as a software size measure, Information Technology and Management 24 (2023) 293–312.

[14] H. Ünlü, T. Hacaloğlu, N. K. Ömüral, N. Çalişkanel, O. Leblebici, O. Demirörs, An exploratory case study on effort estimation in microservices, in: 2023 49th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), IEEE, 2023, pp. 215–218.

[15] E. Ungan, N. Cizmeli, O. Demirörs, Comparison of functional size based estimation and story

points, based on effort estimation effectiveness in scrum projects, in: 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications, IEEE, 2014, pp. 77–80.
[16] M. Salmanoglu, T. Hacaloglu, O. Demirors, Effort estimation for agile software development: Comparative case studies using COSMIC functional size measurement and story points, in: Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement, 2017, pp. 41–49.
[17] H. Huijgens, R. v. Solingen, A replicated study on correlating agile team velocity measured in function and story points, in: Proceedings of the 5th International Workshop on Emerging Trends in Software Metrics, 2014, pp. 30–36.
[18] T. Hacaloglu, O. Demirors, Measureability of functional size in agile software projects: Multiple case studies with COSMIC FSM, in: 2019 45th Euromicro conference on software engineering and advanced applications (SEAA), Ieee, 2019, pp. 204–211.
[19] H. Ünlü, B. Bilgin, O. Demirörs, A survey on organizational choices for microservice-based software architectures, Turkish Journal of Electrical Engineering and Computer Sciences 30 (2022) 1187–1203.
[20] H. Ünlü, D. E. Kennouche, G. K. Soylu, O. Demirörs, Microservice-based projects in agile world: A structured interview, Information and Software Technology 165 (2024) 107334.
[21] T. Hacaloğlu, H. Ünlü, A. Yıldız, O. Demirörs, Software size measurement: Bridging research and practice, IEEE Software 41 (2024) 49–58.
[22] F. Dalpiaz, A. Ferrari, X. Franch, C. Palomares, Natural language processing for requirements engineering: The best is yet to come, IEEE Software 35 (2018) 115–119.
[23] J. von der Mosel, A. Trautsch, S. Herbold, On the validity of pre-trained transformers for natural language processing in the software engineering domain, IEEE Transactions on Software Engineering 49 (2023) 1487–1507.
[24] M. Choetkiertikul, H. K. Dam, T. Tran, T. Pham, A. Ghose, T. Menzies, A deep learning model for estimating story points, IEEE Transactions on Software Engineering 45 (2019) 637–656.
[25] I. Hussain, L. Kosseim, O. Ormandjieva, Approximation of COSMIC functional size to support early effort estimation in agile, Data & Knowledge Engineering 85 (2013) 2–14.
[26] M. Ochodek, S. Kopczyńska, M. Staron, Deep learning model for end-to-end approximation of COSMIC functional size based on use-case names, Information and Software Technology 123 (2020) 106310.
[27] L. Lavazza, A. Locoro, G. Liu, R. Meli, Estimating software functional size via machine learning, ACM Transactions on Software Engineering and Methodology 32 (2023) 1–27.
[28] G. Sikka, A. Kaur, M. Uddin, Estimating function points: Using machine learning and regression models, in: 2010 2nd International Conference on Education Technology and Computer, volume 3, IEEE, 2010, pp. V3–52.
[29] E. Ungan, C. Hammond, A. Abran, Automated COSMIC measurement and requirement quality improvement through scopemaster® tool., in: IWSM-Mensura, 2018, pp. 1–13.
[30] H. Soubra, Y. Abufrikha, A. Abran, et al., Towards universal COSMIC size measurement automation., in: IWSM-Mensura, 2020.
[31] H. Huijgens, M. Bruntink, A. Van Deursen, T. Van Der Storm, F. Vogelezang, An exploratory study on functional size measurement based on code, in: Proceedings of the international conference on software and systems process, 2016, pp. 56–65.
[32] Y. S. Molla, S. T. Yimer, E. Alemneh, COSMIC-functional size classification of agile software development: Deep learning approach, in: 2023 International Conference on Information and Communication Technology for Development for Africa (ICT4DA), IEEE, 2023, pp. 155–159.
[33] H. Ünlü, S. Tenekeci, C. Çiftçi, I. B. Oral, T. Atalay, T. Hacaloğlu, O. Demirörs, Predicting software functional size using natural language processing: An exploratory case study, in: 50th Euromicro Conference Series on Software Engineering and Advanced Applications (SEAA), 2024.
[34] S. Salem, H. Soubra, Using nlp for functional size measurement of iot devices, in: 2023 Eleventh International Conference on Intelligent Computing and Information Systems (ICICIS), IEEE, 2023, pp. 321–327.

[35] E. M. D. B. Fávero, D. Casanova, A. R. Pimentel, Se3m: A model for software effort estimation using pre-trained embedding models, Information and Software Technology 147 (2022) 106886.
[36] C. V. Dave, A. Patel, U. Keshri, An efficient framework for cost and effort estimation of scrum projects, International Journal for Research in Applied Science and Engineering Technology 9 (2021) 1478–1487.
[37] Monika, O. P. Sangwan, Software effort estimation using machine learning techniques, in: 2017 7th International Conference on Cloud Computing, Data Science & Engineering - Confluence, 2017, pp. 92–98.
[38] Y. Mahmood, N. Kama, A. Azmi, A. S. Khan, M. Ali, Software effort estimation accuracy prediction of machine learning techniques: A systematic performance evaluation, Software: Practice and experience 52 (2022) 39–65.
[39] P. Sharma, J. Singh, Systematic literature review on software effort estimation using machine learning approaches, in: 2017 International Conference on Next Generation Computing and Information Systems (ICNGCIS), IEEE, 2017, pp. 43–47.
[40] E. M. D. B. Fávero, D. Casanova, Bert_se: A pre-trained language representation model for software engineering, arXiv preprint arXiv:2112.00699 (2021).
[41] M. Qin, Lattice lstm model for function point based software cost measurement, in: 2019 IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), IEEE, 2019, pp. 731–735.
[42] M. Qin, L. Shen, D. Zhang, L. Zhao, Deep learning model for function point based software cost estimation-an industry case study, in: 2019 International Conference on Intelligent Computing, Automation and Systems (ICICAS), IEEE, 2019, pp. 768–772.
[43] A. F. de Araújo, R. M. Marcacini, Re-bert: automatic extraction of software requirements from app reviews using bert language model, in: Proceedings of the 36th annual ACM symposium on applied computing, 2021, pp. 1321–1327.
[44] K. Kaur, P. Kaur, Improving bert model for requirements classification by bidirectional lstm-cnn deep model, Computers and Electrical Engineering 108 (2023) 108699.
[45] X. Luo, Y. Xue, Z. Xing, J. Sun, Prcbert: Prompt learning for requirement classification using bert-based pretrained language models, in: Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, 2022, pp. 1–13.
[46] F. Yucalar, Developing an advanced software requirements classification model using bert: An empirical evaluation study on newly generated turkish data, Applied Sciences 13 (2023) 11127.
[47] D. Kici, G. Malik, M. Cevik, D. Parikh, A. Basar, A bert-based transfer learning approach to text classification on software requirements specifications., in: Canadian Conference on AI, volume 1, 2021, p. 04207.
[48] R. K. Yin, Case study research and applications, volume 6, Sage Thousand Oaks, CA, 2018.
[49] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, et al., Codebert: A pre-trained model for programming and natural languages, arXiv preprint arXiv:2002.08155 (2020).
[50] H. Husain, H.-H. Wu, T. Gazit, M. Allamanis, M. Brockschmidt, Codesearchnet challenge: Evaluating the state of semantic code search, arXiv preprint arXiv:1909.09436 (2019).
[51] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, Bert: Pre-training of deep bidirectional transformers for language understanding, arXiv preprint arXiv:1810.04805 (2018).
[52] A. Tarhan, M. A. SAĞ, COSMIC solver: A tool for functional sizing of java business applications, Balkan Journal of Electrical and Computer Engineering 6 (2018) 1–8.
[53] Ö. Özen, B. Özsoy, B. Aktılav, E. C. Güleç, O. Demirörs, Automated estimation of functional size from code, in: 2020 Turkish National Software Engineering Symposium (UYMS), IEEE, 2020, pp. 1–7.
[54] A. Sahab, S. Trudel, COSMIC functional size automation of java web applications using the spring mvc framework., in: IWSM-Mensura, 2020.
[55] R. C. Martin, Agile software development: principles, patterns, and practices, Prentice Hall PTR, 2003.

[56] T. Hastings, A. Sajeev, A vector-based approach to software size measurement and effort estimation, IEEE Transactions on Software Engineering 27 (2001) 337–350.
[57] F. Hvilshøj, The Full Guide to Automated Data Annotation, 2024. Available at https://encord.com/blog/automated-data-annotation-guide/.