Received: 15 September 2019 | Revised: 19 January 2020 | Accepted: 26 January 2020

DOI: 10.1002/smr.2253 10.1002/smr.2253

RESEARCH ARTICLE

Effort Estimation of Open Source Android Projects via Transaction Analysis

Kan Qi* | **Barry Boehm**

[1]Center for Systems and Software Engineering, University of Southern California, California, United States

Correspondence
*Kan Qi, 941 Bloom Walk (SAL), Room 327. Los Angeles, CA 9007, USA. Email: kqi@usc.edu

Present Address
941 Bloom Walk (SAL), Room 327. Los Angeles, CA 9007, USA

Summary

Transactions have been used in many software sizing methods to measure software functional size and provide a basis for effort estimation. To further investigate the effects that transactions have on software functional size and more accurately estimate project effort using the transactional information, in this paper, we propose a method to identify transactions, as well as their complexity attributes that are influential to software functional size, directly from the source code of Android projects using static and dynamic analyses.

The identified transactions are classified into different complexity levels and applied with different weights based on their evaluated complexity levels to distinguish their effects on software functional size. A size metric called SWT is proposed to measure software functional size using the sum of weighted transactions. We calibrate an effort estimation model using SWT based on a data set of 34 open source Android projects, and show the effectiveness of using SWT as a software functional size measure to estimate project effort.

KEYWORDS:
Android application analysis, software functional size analysis, transaction analysis, use case analysis, software size metrics, effort estimation, model calibration, Bayesian analysis

1 | INTRODUCTION

The rising popularity of mobile devices has drawn a great amount of attention from project effort estimation research community to mobile app development. Many effort estimation methods have been proposed or adopted to support the project management decisions of developing Android applications. For instance, Kaur proposed a modified version of Use Case Points, called M-UCP, to estimate project effort of Android applications[1]; Preuss discussed her experience in using IFPUG function points to develop a mobile app, called Terrific Tuner[2]; Francese[3] proposed a multiple linear regression model using requirement and program features to estimate mobile project effort. To more effectively measure software functional size and accurately estimate development effort of Android projects, we extended our previous research[4,5] of transaction analysis to Android source code and empirically studied 34 open source Android applications using the proposed transaction analysis method.

The proposed transaction analysis method first identifies transactions from Android source code and the compiled Android Application Packages (APKs), and then classifies the transactions into different complexity levels based on their complexity attributes measured to represent transactional complexity in different aspects. These two transaction analysis steps - transaction identification and transaction classification - are used to support software functional size analysis. Different weights are assigned to the transactions of different complexity levels to distinguish their effects on software functional size. The weights are calibrated using Bayesian analysis that combines both the expert-based judgment and effects identified from the empirical data set. The sum of weighted transactions (SWT) is used to measure software functional size.

An effort estimation model is calibrated using SWT to estimate project effort of Android projects. To calibrate the effort estimation model and evaluate its performance, we analyzed 34 open source Android projects. The transactional information is derived using the proposed transaction

analysis method, while effort data is derived by extending the effort simulation method proposed by Robles et al.[6 7], which classifies the developers into full-time and part-time developers based on Git commits frequency and applies average weekly working hours to calculate the project effort in person hours. In our empirical study, we validate the calculated effort with the open source project developers to ensure the project effort data used to calibrate and evaluate the effort estimation model achieve its best accuracy at the current stage of research. Based on these two pieces of information, we calibrate the proposed effort estimation model and evaluate its performance in terms of goodness of fit and out-of-sample accuracy using *k*-fold cross-validation.

We regard Robles at al.'s method as a counting method of project effort, which can be used to measure productivity, instead of an estimation method that can be applied at the early stage of a project, since the complete committing records are not available until a project finishes. Therefore, their method works better as a proxy to simulate project effort, which can be used to calibrate other effort estimation models.

The proposed transaction analysis method targets the source code, which is the type of information only available at the post-development phase of a project. To bridge the gap between post-development information and early effort estimation, in Section 2, we provide in-depth discussion about how the transaction analysis method proposed in this study and the transaction analysis methods proposed in our previous studies[5 4] together can provide an effective solution. In summary, the outcomes of this paper are as follows:

1. An automated method of identifying and classifying transactions, which supports the software functional size analysis of Android projects.
2. An effort estimation model that is calibrated using the transactional information derived from 34 open source Android projects.
3. The evaluation results of the model in terms of its goodness of fit and out-of-sample accuracy in comparison with 4 baseline effort estimation models.

This paper is structured as follows: in Section 2, we discuss how our previous work[5 4] and this proposed work together provide early-phase effort estimation via post-development model calibration; Section 3 introduces the transaction-based effort estimation model that we extend from our previous research[5]; Section 4 elaborates the proposed transaction analysis method for Android projects; in Section 5, we report our transaction analysis results of the 34 open source Android projects and evaluate the proposed effort estimation model; Section 6 discusses the threats to validity and Section 7 discusses the related work. Section 8 concludes the research and introduces the future directions.

2 | POST-DEVELOPMENT CALIBRATION FOR EARLY EFFORT ESTIMATION

The proposed transaction analysis method is directly applied to the source code, which is the type of information only available after the development of a software product. To bridge the gap between the post-development information and early effort estimation, we provide transaction analysis methods that target different phases of a project to provide transactional information for model calibration and effort estimation. For instance, in our previous study[5], we propose an incremental effort estimation approach that provides three transaction-based effort estimation models targeting the three early phases of a project, including requirements, analysis, and design phases, to support early-phase project management decisions; in this research, we propose a transaction analysis method of Android source code to provide transactional information for model

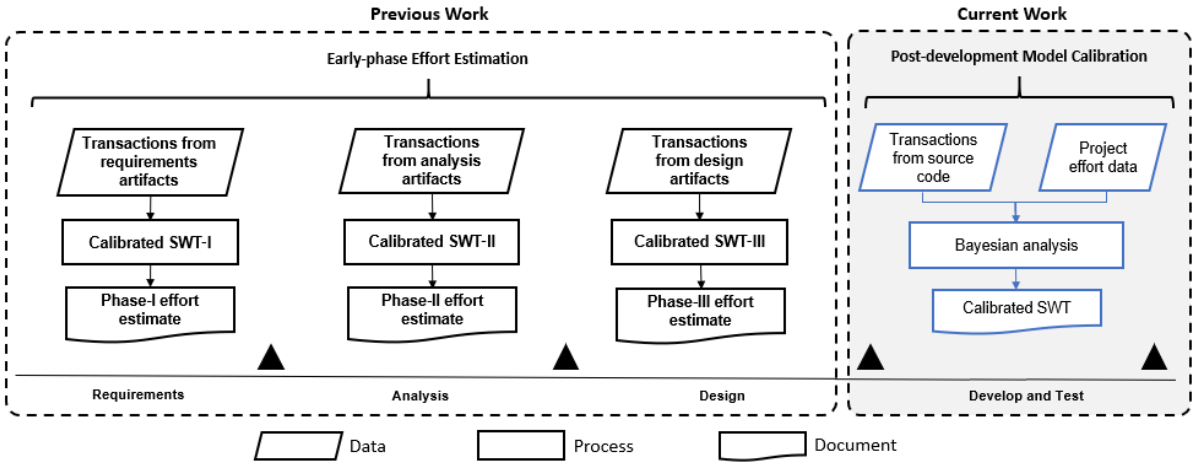

FIGURE 1 Post-development Model Calibration for Early Effort Estimation

calibration. Figure 1 provides an overview about the transaction analyses applied to different phases of a project. Since the three early-phase effort estimation models proposed in the previous study [5] share the same transaction model (Section 3.1) used by the transaction analysis method proposed in this study, the proposed transaction analysis method provides aid in early-phase transaction-based effort estimation in the following two ways:

1. Providing transaction information for early-phase transaction-based effort estimation model calibration. To provide early-phase effort estimates, the early-phase estimation models need to be calibrated first. The challenge for early-phase model calibration is the limited early-phase analysis and design artifacts that are available for transaction analysis. Beyond the research environments that provide specific types of artifacts for transaction analysis, the transactional information available at the most of the commercial or open-source environments is from the source code. Therefore, the techniques of analysing transactions directly from source code have become a critical part that produces the sufficient data for model calibration and evaluation. For instance, the three early-phase transaction-based effort estimation models proposed in our previous research [5] can be calibrated using the transactions identified from source code with tailored numbers of complexity attributes that are used to define early-phase estimation models.

2. Providing early-effort estimates using early-phase representations of transactions. The transactional information derived using the transaction analysis method proposed in this study is mapped into an abstract transaction model, which has many early phase representations [4 8 9 10]. For example, a transaction can be modeled as user-system interactions identified from use case narratives, as sequences of operations identified from object analysis diagrams, or as sequences of messages identified from sequence diagrams. Early effort estimates can be made using those early-phase representations as input of the transaction-based effort estimation models calibrated using source code information. For instance, in our previous research [4], we proposed a model called user-system interaction model (USIM) that identifies the key elements from sequence and class diagrams to define transactions. USIM is an extension of the transaction model (Section 3.1), which models the complete set of transactions of a software system in order to determine the counting scope. The sequence and class diagrams are the design artifacts available at the early stage of a project, which can be used as input for the transaction-based effort estimation models calibrated based on post-development information for early effort estimation.

Therefore, the transaction analysis method proposed in this study adds the missing link between the post development information and early phase effort estimation, and has become an important part of the incremental effort estimation approach proposed in our previous study [5].

3 | TRANSACTION-BASED EFFORT ESTIMATION MODEL

In this section, we define a transaction-based software sizing model that measures software functional size in terms of sum of weighted transactions, and the effort estimation model that estimates project effort using the proposed transaction-based size measure. We also introduce the Bayesian model that models the posterior probability of the parameters of the effort estimation model, which is used to statistically calibrate the model parameters.

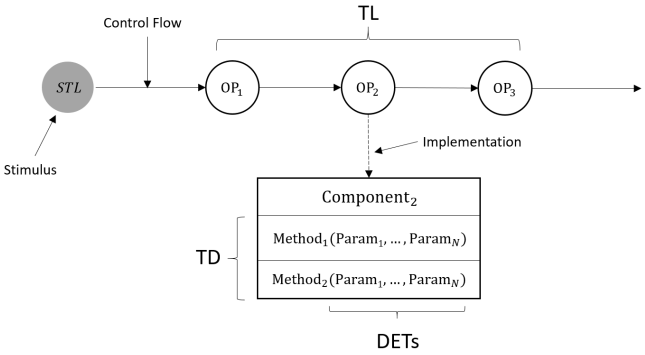

FIGURE 2 The Transaction Model for Software Sizing

3.1 | The Transaction Model

We formalize the definition of a transaction, as a tuple of 4 elements - <*S*, *O*, *C*, *D*>, including the stimulus (*S*) that triggers a transaction, the set of operations (O) that realize a transaction, the components (*C*) where the operations are implemented, and the data element types (*D*) associated with a transaction. The definition of a transaction is provided in Definition 1.

Definition 1. *Transaction*: A transaction is a sequence of operations (*O*) exercised by system components (*C*) to fulfill a type of interaction (*S*) between an actor and a system.

This definition models a transaction as a type of interaction between an actor and a software system, and provides a perspective about the internal behavior of a software system in terms of operations and components involved to provide the response to an actor's action. The data aspect (*D*) is implicitly defined as the data elements referenced by the operations (*O*). Therefore, for each identified transaction, the overall complexity of a transaction can be characterized in three aspects, including the operational, structural, and data complexities. The three types of complexity can be measured in terms of TL, TD, and DETs, which are three complexity attributes of a transaction defined as follows:

1. Transaction Length (TL), measures operational complexity of a transaction, by counting the number of the operations that realize a transaction.
2. Transaction Degree (TD), measures the structural complexity of a transaction, using the average number of service methods of the components that implement a transaction.
3. Data Element Types (DETs), measures data complexity of a transaction, by counting the number of data element types associated with a transaction.

Measurements of TL, TD, and DETS suggest the amount of effort for understanding, implementing, and testing a transaction. Figure 2 provides a graphic representation of a transaction and its complexity attributes. The identification of a transaction requires identifying all the elements specified in the definition, which provides transactional information for software functional size analysis.

3.2 | The Transaction-based Sizing Model

Due to the fact that transactions of different complexity levels require different amount of effort for development, the influence of the transactions on software functional size is weighted differently. Therefore, we use sum of weighted transactions as the measure for software functional size. The functional form of the software sizing model can be generally defined as Eq. (1).

$$\textit{Software Size} = \sum_{t \in T} w(t) \tag{1}$$

Where, T is the set of identified transactions of a software system. $w(t)$ is the function that weights the influence of a transaction t on software functional size. Our approach of implementing $w(t)$ is to classify the identified transactions into different complexity levels, which generates subsets of T: $T = \{T_1, T_2, ..., T_n\}$, and assign different weights W: $W = \{w_1, w_2, ..., w_n\}$, to the subsets to differentiate effects of the transactions on software functional size. Therefore, Eq. (1) is extended into Eq. (2) in our software functional size analysis.

$$\textit{Software Size} = w_1 * \sum_{t \in T_1} t + w_2 * \sum_{t \in T_2} t + ... + w_n * \sum_{t \in T_n} t \tag{2}$$

The number of subsets (n) defines the granularity of classifying transactions, which critically affects the accuracy of estimating software functional size and project effort. In the proposed model calibration process, we classify transactions into different numbers of subsets using different classification criteria to optimize the estimation accuracy. Each way of classifying the transactions is defined as a classification function $f_{cmplx}(t)$, which evaluates the complexity level l of a transaction t. Therefore, the functional form of the transaction-based software sizing model can be rewritten as Eq. (3).

$$\textit{Software Size} = \sum_{t \in T} \sum_{i \in \{1...|W|\}} w_i * I_i(f_{cmplx}(t)) \tag{3}$$

Where, T is the set of transactions;

$f_{cmplx}(t)$ classifies a transaction into a level of complexity $l \in \{1...n\}$;

$W = \{w_1, w_2, ..., w_n\}$ is a set of weights assigned to the n complexity levels; $|W|$ is the size of W, which equals n;

$I_i(x)$ is the indicator function, which returns 1 if l equals i, or 0 if otherwise; i iterates through $1...|W|$ to identify the weight w_i assigned to a transaction t.

3.3 | The Effort Estimation Model

The effort estimation model models the effect that one unit of the size measurement has on project effort using an effort adjustment factor (α), which can be generally defined as Eq. (4).

$$\textit{Project Effort} = \alpha * \textit{Software Size} \tag{4}$$

Plugging the functional size metric defined in Eq. (3) into Eq.(4), we derive the functional form of the transaction-based effort estimation model as Eq. (5).

$$\textit{Project Effort} = \alpha * \sum_{t \in T} \sum_{i \in \{1 \ldots |W|\}} w_i * I_i(f_{cmplx}(t)) \tag{5}$$

This transaction-based effort estimation model is parameterized by an effort adjustment factor α, a classification function $f_{cmplx}(t)$ that classifies the transactions into different complexity levels, and a set of the weights W assigned to the complexity levels. To calibrate the model, we introduce the Bayesian model that models the posterior probability of α and W in the following section, and the Manhattan distance based classification function $f_{cmplx}(t)$ in Section 4.2.

3.4 | The Bayesian Model for Parameter Calibration

The weights W assigned to the complexity levels and the effort adjustment factor α are statistically modeled by a Bayesian model as shown in Eq. (6).

$$p(\alpha, W|x, y) \propto p(x, y|\alpha, W) * p(\alpha) * p(W) \tag{6}$$

We model the joint posterior probability $p(a, W|x, y)$ of the weights W and the effort adjustment factor α as proportional to the product of the likelihood $p(x, y|a, W)$ and the prior probabilities of the parameters: $p(a)$ and $p(W)$. The prior probabilities represent our prior beliefs about how the parameters affect project effort, while the likelihood represents how likely the observed effort data can be explained by the parametric effort estimation model defined in Eq. (5).

In our previous study [5], we elaborated the Metropolis-Hastings MCMC algorithm to simulate the posterior probabilities of the parameters using this Bayesian model. We used the mean values of the posterior probability distributions as the final estimates of the parameters. In this study, we follow the same algorithm to simulate the posterior probability distributions of the parameters. Therefore, we skip the algorithm to save space.

4 | TRANSACTION ANALYSIS OF ANDROID SOURCE CODE

To calibrate and evaluate the proposed effort estimation model, we apply two stages of transaction analysis - transaction identification and transaction classification to derive the transactional information from Android source code.

4.1 | Transaction Identification

To identify a transaction, three types of elements need to be determined, including the system components (*C*), the stimulus (*S*) of the user-system interaction that initiates the transaction, and the operations (*O*) that realize the transaction. We identify system components (*C*) using the agglomerative clustering approach proposed by Cui et al. [11], stimuli (*S*) using use case analysis, and transaction operations (*O*) through run-time method calls analysis.

4.1.1 | System Components Analysis

Due to lacking project-specific knowledge about the subject systems, it is impossible to accurately identify system components using manual effort. Therefore, we adopt the automated method of identifying system components from source code and validate the identified system components with the developers.

The basic idea of automated system components identification is to classify the class units (e.g. Java classes) into different clusters and use the clusters to represent the system components [11 12 13]. In this research, two popular methods were tested as candidate methods to identify system components from Android source code, which are the comprehension-driven clustering algorithm (ACDC) proposed by Tzerpos et al. [12] and the agglomerative hierarchical clustering algorithm proposed by Cui et al. [11] For ACDC, a tool is provided to analyse the byte code of Java projects. In

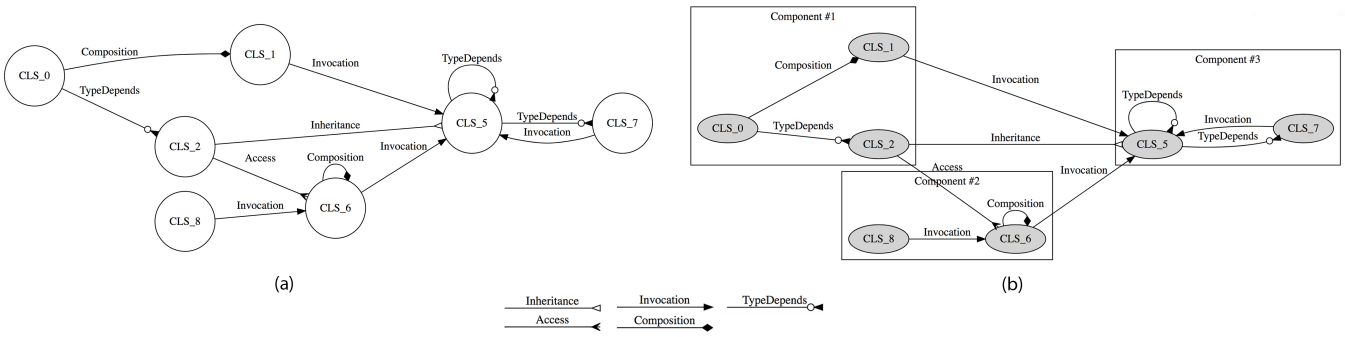

FIGURE 3 Identifying Dependency Graphs and System Components

our empirical evaluation of the 34 Android projects, we found the ACDC tool is only able to generate the results for 78% of the subject projects. Therefore, for a complete set of system components identification results, we use the agglomerative hierarchical clustering algorithm provided by Cui et al.[11] and follow their empirical study to optimize the system components analysis results. We apply the following three steps to identify system components:

Step 1: Identify Class Units. To identify class units, we use Soot framework[14] to parse the Java files of the subject Android projects. This process identifies the application classes, which are separated from the library classes, to establish the counting scope for software size measurement. The classes under the package declared in the "manifest" file of an Android project are defined as application classes. Only application classes are considered influential to project effort.

A set of class units that are defined in the same source code file, including all the top-level classes and the inner classes nested in the top-level classes are regarded as one composite class unit. These composite class units are used as the atomic units in the following agglomerative clustering process, such that critical information regarding file hierarchy, such as directory and file based implementation of logic system components[12] and the direct mapping between Java class files and object-oriented concepts in Java programming language[15], is preserved during the agglomerative clustering step.

Step 2: Identify Dependency Graphs. In addition to identifying class units, dependency relationships between the class units are also identified. The dependency relationships including composition, extension, access, invocation, type dependencies are used to calculate the similarity between the class units in the agglomerative clustering step. 5 types of graphs are established using classes as nodes and the five types of dependency relationships as edges. Section (a) of Figure 3 provides an instance of the 5 graphs that uses different arrows to distinguish the 5 dependency graphs. The 5 graphs are defined as follows[11]:

1. Type dependency graph, an operation op_i of class cls_i has a parameter or the return variable typed as class cls_j;
2. Call graph, an operation op_i of class cls_i, which invokes operation op_j of class cls_j;
3. Access graph, an operation op_i of class cls_i, which accesses an attribute $attr_j$ of class $class_j$;
4. Extends graph, a class cls_i extends class cls_j or implements interface int_k;
5. Composition graph, an attribute $attr_i$ of class cls_i is typed with class cls_j;

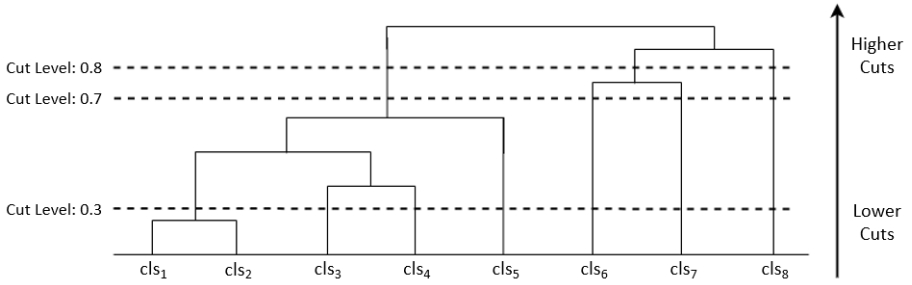

FIGURE 4 Clusters Generated by Cutting the Dendrograms

TABLE 1 The Three Clustering Optimization Schemes

Clustering Scheme	Weighting Scheme	Similarity Measure	Linkage Method	Optimization Criteria	Cut Level
S1	*Binary*	*Euclidean*	*Single*	*Cohesion*	*0.8*
S2	*Relative*	*Euclidean*	*Single*	*Coupling*	*0.3*
S3	*Absolute*	*Unbiased Ellenberg*	*Complete*	*Size*	*0.7*

Step 3: Identify Components using Agglomerative Clustering. Using the 5 dependency graphs as the input, the agglomerative clustering algorithm applies the following three steps to cluster class units into system components:

1. A weight is assigned for each pair of class units using a weighting scheme to represent the degree of their association. Therefore, each class unit is abstracted into a feature vector, and a matrix is created to record all the vectorized class units.
2. The similarity between each pair of the class units is calculated using a feature-vector based similarity measure.
3. The class units are agglomeratively combined the into clusters using a linkage function, which constructs a hierarchy from lower-level class units into high-level clusters. The hierarchy can be represented as a dendrogram shown in Figure 4.

The agglomerative clustering algorithm is parameterized using different weighting schemes, similarity measures, and linkage functions to provide different ways of clustering the class units. For instance, in Cui et al.'s study, 3 weighting schemes, including Binary, Absolute, and Relative weighting schemes, 3 similarity measures, including Euclidean, Jaccard, and Unbiased Ellenberg similarity measures, and 3 linkage functions, including Single, Complete, and Unweighted Average linkage functions are provided and evaluated for their effectiveness of recovering system components. The detailed definitions of the weighting schemes, similarity measures, and linkage functions can be found from Cui et al.'s study [11], therefore, we skip the definitions.

Each combination of weighting scheme, similarity measure, and linkage function is defined as a clustering scheme. In Cui et al.'s empirical study of 11 subject systems, 18 clustering schemes were evaluated using three architecture recovery optimization criteria, including Size, Cohesion, and Coupling criteria. Three clustering schemes were found to optimize the three clustering criteria, which are provided in Table 1. We use these three clustering schemes in our empirical study of the subject open source Android projects, which is, for each subject system, three dendrograms are generated as the results of the three clustering schemes. The dendrograms are cut at the levels provided in Table 1 to generate three lists of sub-trees. Class units in the same sub-tree is defined as a system component, and each list of subtrees is the set of system components that are identified using a clustering scheme.

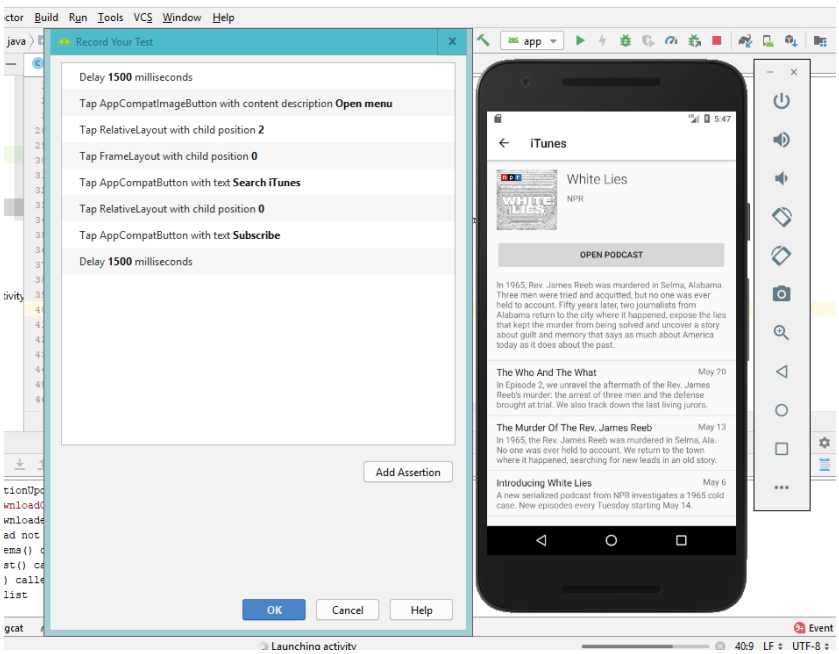

FIGURE 5 An Example of Espresso Test Case that Encodes User Interface Operations

To validate the clustering results, for each subject project in our empirical study, we conduct a survey among the developers to understand which clustering result can best describe the actual system components, the results of which are provided in Section 5.4.2. An example of the identified components is provided in section (b) of Figure 3.

4.1.2 | Use Case Analysis

Since transactions model a software system's internal processes that realize the system's functionality, identifying a complete set of transactions relies on identifying the scope of system functionality. We identify the use cases of the subject Android apps to provide such a coverage of the functionality of the Android apps [15].

Each use case includes a sunny day scenario and a set of possible rainy day scenarios. Sunny day scenarios define the user-system interactions to achieve the goal under ideal conditions, while rainy day scenarios define the user-system interactions under the unexpected conditions [15]. Since transactions are defined as the responses triggered by actors' actions during user-system interactions, therefore, the user-system interactions identified in the sunny day and runny day scenarios define the scope of transaction identification, and actors' actions of the user-system interactions are abstracted as a set of stimuli that initiate the transactions.

We operate the apps based on the identified use cases and record the actors' actions as Android Espresso test cases. Android Espresso is a testing framework to automate the testing of user graph interface (GUI), through which actions on Android apps can be recorded as testing Activities which can be automatically executed in the GUI testing [16]. An example of the set of identified stimuli in terms of an Espresso test case is provided in Figure 5. Each of the identified use cases is recorded as an Espresso test case and encoded as an Android testing Activity.

4.1.3 | Profile Transaction

We follow the following three steps to profile the transactions and their complexity attributes:

Step 1: Output Method Calls. We instrument the application classes of Android source code using RunDroid [17], which inserts instructions that output the time stamps of entering and exiting of each method during execution, so that the methods that are called within the counting scope can be recorded in a time series as an Android app is triggered by the set of identified stimuli. We then replay the Espresso test cases that include all the identified stimuli of the Android application to output logs of method calls.

Step 2: Identify Callbacks. In Android framework, stimuli are dispatched to UI components as the events delivered to event handlers. Those event handlers are the entry points of the internal processing of an Android app for the intended functionality. Android applications implement the event handlers as UI element callbacks that handle requests from the users, such as, clicks on buttons, scrolling gestures, and key presses, and life-cycle callbacks that handle the state changes notified by Android platform, such as, start, pause, stop, destroy, etc. Therefore, the callback methods are the entry points of the implementation of the functionality of an Android application. For instance, in the $onClick$ callback method of $onClickListener$ registered for the log-in button's click event, the routine of validating user log-in information is performed; the lifecycle callback method $onCreate$ is implemented to perform the basic program setup, such as loading user's personalized data, security checking, and cache setup;

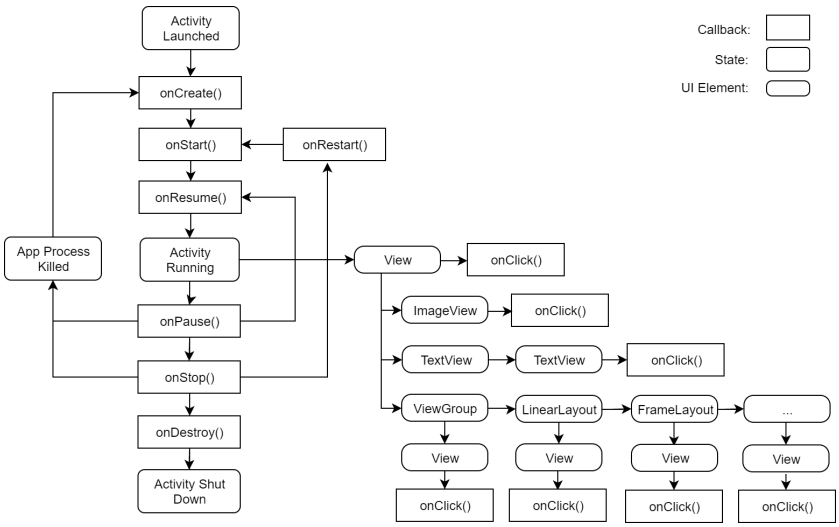

FIGURE 6 The UI Callbacks and Life-cycle Callbacks for Android Applications

in $onStop$, de-registration, cache cleaning, and state saving are performed. Figure 6 provides an overview for the UI and life-cycle callbacks we identify for the transaction analysis.

Those callbacks have been substantially studied by the previous studies of UI modeling, control flow analysis, and automated testing methods of Android applications [18] [19]. Tools are provided to identify the callbacks and their association, hierarchy, and transition relationships. We use Gator [18] and FlowDroid [19] to identify the callbacks and merge their results into a comprehensive set of callbacks. The callbacks are used to segment the logs of method calls to identify transactions. In addition to identifying callbacks from Android activities, we also identify the callbacks from the content providers, services, and broadcast receivers, since they are the three other types of Android application components that provide users with functionality.

Step 3: Output Transactions. The logs of method calls output at Step 1 are separated by the callbacks identified at Step 2, the method calls are then mapped to the components identified using method introduced in Section 4.1.1. A method call is not necessarily a component-level operation. For example, a few consecutive private method calls are not component-level operations, since they are invoked within a class unit and thus internal to a component. A method call is determined as a component-level operation if satisfying the following two criteria:

1. The called method belongs to the class units within the counting scope.
2. The system component that localizes the called method is different from the system component that localizes the last called method.

After this step, the method calls are mapped into component-level operations that realize a transaction, and the methods localized to components define the interface methods of the components. The method calls that are not mapped into component-level operations are omitted in the definition of a transaction. When profiling the transactions, information, such as the implementation components, the component-level operations, and the data element types of each transaction, is also assessed and profiled, which is used to evaluate the complexity attributes of a transaction described in Section 3.1. An example of profiling a transaction from the method calls is provided in Figure 7.

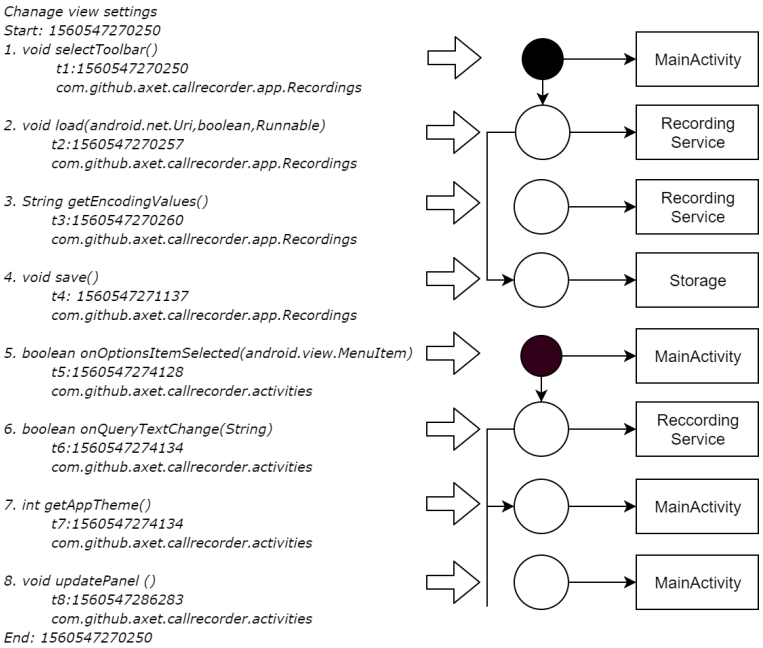

FIGURE 7 An Example of Identifying Transactions from Logs of Method Calls

4.2 | Transaction Classification

Transactions are classified into different complexity levels based on their complexity attributes. The classification function maps the multi-dimensional representation of a transaction into the uni-dimensional representation in terms of complexity levels, which are used to differentiate the effects of the transactions on project effort.

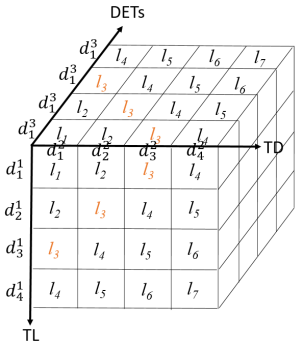

FIGURE 8 Classification by Discretizing the Three Transactional Dimensions using 4-quantiles

To classify the transactions over the three dimensions - TL, TD, and DETs, we first discretize the distributions of the transactions over the individual dimensions using the quantile-based discretization method. This provides a set of cut points over the dimensions. Figure 8 provides an example of discretizing the three dimensions using 4 bins. In our empirical study, we discretize the dimensions with 1 to 6 quantiles to search the best binning for classification. The subscript (l) of each bin (d_l^i) represents the complexity level of a transaction with respect to the dimension d^i. The higher number the subscript is, the more complex the transaction is rated in that aspect. Therefore, a transaction t is defined as a vector over the discretized dimensions with coordinates $< d_l^1, d_l^2, d_l^3 >$.

Using the coordinates of the transactions, the Manhattan distances between the transactions and the origin ($o =< 0, 0, 0 >$) can be calculated as $\sum_{i\in\{1\dots3\}} |d_l^i|$, which is the sum of absolute differences between the coordinates of t and o for the three dimensions. The classification function $f_{complx}(t)$ we adopt in our empirical study is defined based on Manhattan distance, which is provided in Eq. (7).

$$\begin{aligned} f_{cmplx}(t) &= \textit{manhattan-distance}_D(t) - |D| + 1 \\ &= \sum_{i\in\{1\dots|D|\}} d^i(t) - |D| + 1 \end{aligned} \tag{7}$$

Where, $D = \{TL, TD, DETs\}$, represents the individual discretized dimensions. $|D|$ represents the number of dimensions that characterizes a transaction, which in our case is 3. $d^i(t)$ returns the index of the bin into which a transaction t is categorized for dimension d^i. The Manhattan distance is shifted with a constant $|D| - 1$ to make the complexity levels starting from 1. This function maps measurements of complexity for the individual dimensions into a complexity level $l \in \{l_1, ..., l_n\}$. The larger Manhattan distance that a transaction has from the origin o, the higher complex level the transaction is rated at. This classification function also provides the max number of the complexity levels, which can be calculated by Eq. (8).

$$max(f_{cmplx}) = \sum_{i\in\{1\dots|D|\}} |d^i| - |D| + 1 \tag{8}$$

Where, $|d^i|$ is the number of bins for dimension d^i. For example, under discretization of 4 bins for the three dimensions, the max number of complexity levels is 10, and a transaction t that has coordinates of $\{3, 4, 1\}$ is classified into the complexity level of 6.

5 | EMPIRICAL STUDY

To understand the specific effects that transactions of different complexity levels have on project effort, and how accurately we can estimate project effort using the identified transactional information, we pose the following research questions:

RQ1: What effects do the transactions of different complexity levels have on project effort?

RQ2: Can the proposed transaction-based size metric be used as an effective size measure?

RQ3: Can the transaction-based effort estimation model estimate project effort with satisfactory estimation accuracy?

RQ3.1: Can the proposed effort estimation model effectively model the development effort of Android projects?

RQ3.2: Can the proposed effort estimation model improve estimation accuracy in comparison with other typical estimation models?

RQ3.3: Are the improvements in estimation accuracy statistically significant?

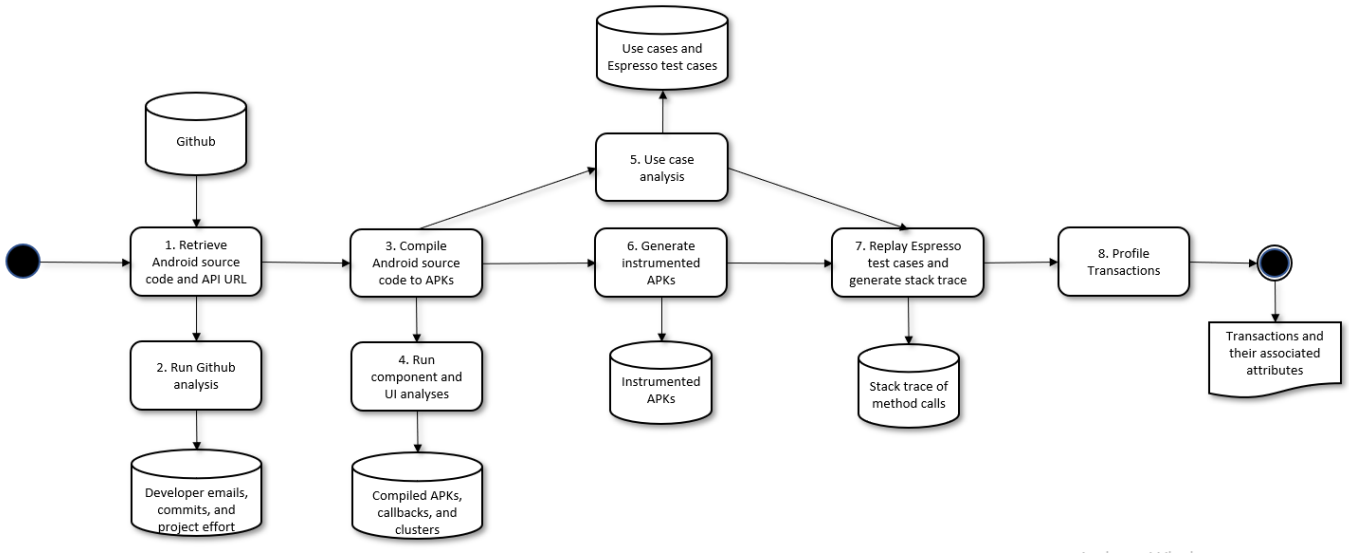

FIGURE 9 The Overview of Identifying Transactions from Open Source Android Projects

5.1 | The Subject Open Source Projects

We select 34 representative Android projects from Github. We tried to identify the Android projects that are well maintained and have at least one release at the public Android App stores, for example, Google Play Store and Fossdroid [20]. We select Android projects that vary in software size, personnel, and duration, and balance the numbers of the Android apps in 20 major categories provided by FossDroid, for example, system, multimedia, tool, navigation, etc. The Android projects have wide range of contributors from 2-300 contributors and deliver software products ranging from 4-47 KSLOC.

Figure 9 outlines the overall process of our empirical study of the Android projects. Following the transaction analysis steps outlined in Section 4, we identify the transactions for the 34 subject Android projects and classify the transactions into different complexity levels to calibrate their effects on project effort (Section 5.2). Apart from the source code of the Android projects retrieved from the Github repositories, information such as code commits and emails of the developers is also retrieved for the project effort analysis (Section 5.3) and the open source project survey (Section 5.4) to validate the intermediate results with the developers.

5.2 | Transaction Analysis Results

For the 34 Android projects, we identified 334 use cases, 895 transactions, and 247 system components. For each transaction, we identify its operations (*O*), system components (*C*), and data element types (*D*) to calculate TL, TD, and DETs, which measure the operational complexity, the structural complexity, and the data complexity respectively. Following the discretization example given in Section 4.2, we discretize the three dimensions based on the empirical distributions of the identified transactions using 1 to 6 quantiles, which generates six sets of cut points for

TABLE 2 Cut Points for TL, TD, and DETs based on 1-6 Quantile Discretization of the Empirical Distributions

	TL							TD							DETs						
Cut Points	C_1^1	C_2^1	C_3^1	C_4^1	C_5^1	C_6^1	C_7^1	C_1^2	C_2^2	C_3^2	C_4^2	C_5^2	C_6^2	C_7^2	C_1^3	C_2^3	C_3^3	C_4^3	C_5^3	C_6^3	C_7^3
Cut_1	0.0	*Inf*	-/-	-/-	-/-	-/-	-/-	0.0	*Inf*	-/-	-/-	-/-	-/-	-/-	0.0	*Inf*	-/-	-/-	-/-	-/-	-/-
Cut_2	0.0	4.2	*Inf*	-/-	-/-	-/-	-/-	0.0	11.5	*Inf*	-/-	-/-	-/-	-/-	0.0	9.2	*Inf*	-/-	-/-	-/-	-/-
Cut_3	0.0	3.8	4.8	*Inf*	-/-	-/-	-/-	0.0	10.3	14.2	*Inf*	-/-	-/-	-/-	0.0	6.8	12.3	*Inf*	-/-	-/-	-/-
Cut_4	0.0	3.1	4.2	5.3	*Inf*	-/-	-/-	0.0	9.3	11.5	15.2	*Inf*	-/-	-/-	0.0	5.2	9.2	13.2	*Inf*	-/-	-/-
Cut_5	0.0	2.8	3.8	4.9	6.2	*Inf*	-/-	0.0	8.1	10.1	13.4	18.9	*Inf*	-/-	0.0	3.4	8.9	12.3	17.2	*Inf*	-/-
Cut_6	0.0	2.3	3.1	4.2	5.1	6.8	*Inf*	0.0	7.5	8.9	11.5	15.6	21.5	*Inf*	0.0	2.1	5.6	9.2	14.5	18.5	*Inf*

each dimension. The cut points are provided in Table 2, which are used to define the classification functions as described in Section 4.2, and the classified transactions are used to calibrate the weights using the Bayesian analysis introduced in Section 3.4. The transaction analysis results are sensitive to the system components identified using the agglomerative clustering method introduced in Section 4.1.1. We did a survey among the developers of the open source projects to validate the system components identification results. The survey results are provided in Section 5.4.

5.3 | Project Effort Analysis Results

To derive project effort data, we follow the simulation study proposed by Robles et al.[6], which identifies the full-time developers based on the frequency of the commits made by a developer in a time period. A case study of the OpenStack project validates that this method provides 78% accuracy with high confidence. Qi et al.[7] also endorsed this method and applied more strict criteria in identifying active developers. The underlying assumptions of these simulation studies are:

1. With good precision, the full-time developers would be identified from the committing records.
2. Large portion of contribution is made by the full-time developers.

In our empirical study, we follow the simulation studies[7][6] to analyse the project effort of the 34 open source Android projects by mining their Github repositories. To do that, we first retrieve the committing records of the developers using Github APIs:

1. The number of commits (t) made to the repository for a period of time per developer.
2. The number of days (d) with at least one commit for the period of time per person.

Using the commits records, we classify the contributors into full-time developers and part-time developers following the criteria provided by Qi et al.[7], which are as follows:

1. Has on average one committing record per day.
2. Has total contributions exceeding the average number of contributions for all the contributors.

Based on the identified full-time developers ($fdev$) and part-time developers ($pdev$), a ratio $C_{ac} = 0.9$ is applied to differentiate their contributions to the project. The weekly working hours (h) of 40 is multiplied by the number of weeks (w) to calculate the working hours that a developer contributes to the project. The total project effort is calculated by Eq. (9).

$$\textit{Project Effort} = fdev * h * w * C_{ac} + pdev * h * w * (1 - C_{ac}) \tag{9}$$

We extend the original method by validating the effort data with developers. Specifically, after calculating the project effort based on the numbers of full-time and part-time developers, we distribute the overall calculated project effort to the developers based on their number of contributions, including commits, comments, and posted issues. We ask in the survey if the distribution of the project effort is consistent with the actual number of hours they spent on their projects. If it is not consistent, we ask how much time the developer has spent on the project in total or weekly. If the weekly effort is reported, we calculate the total project effort by multiplying weekly hours with the duration in weeks. The survey results of project effort analysis are provided in Section 5.4.1.

5.4 | The Open Source Project Survey Results

A survey was done to validate two intermediate results as the preparation of the data for effort estimation model calibration. The survey asked the developers two sets of questions which validate the project effort the developers spent on a project and the clustering result of the class units that can best represent the actual system components. The survey was conducted with 1041 developers of the 34 open source Android projects. In total, there are 107 responses to the survey, which is about 10% response rate. In the following sections, we break down the survey and discuss the responses to the survey questions.

5.4.1 | Survey of the project effort analysis results

To validate the project effort that developers spent on the projects, 2 survey questions are asked and a follow-up question is asked for each survey question for more detail or validation.

SQ1: How much time on average would you spend on the project weekly?

SQ1.1: If you know the exact number of hours or you spent more than 40 hours weekly, could you provide us with the number?

The first question is a multiple choice question that provides 6 choices: 1-5 hours, 6-10 hours, 11-20 hours, 21-30 hours, 31-40 hours, > 40 hours, for the developers to select the weekly working hours they spend on the projects. The purpose of this question is provide a quick way of submitting working hours. 96.6% of the respondents provide a choice on this question. The follow-up question is to ask an exact number of weekly working hours a developer spends on the project, 53.4% of the respondents provide exact weekly working hours. We use the responses to update the project effort calculated in Section 5.3.

SQ2: Can the following distribution of the calculated effort describe the total effort you have spent on the project?

SQ2.1: How much time have you spent on the project in total?

For this survey question, we provide the distribution of the project effort for the individual developers (calculated in Section 5.3) to ask the developers of the projects if they agree with the calculated project effort. 81% of the respondents provide answer for this question, and 78.7% of the respondents agree with the project effort calculated in Section 5.3. This result provides a quantitative evaluation of the confidence about the calculated project effort, which is 78.7% of the time the calculated project effort is reasonable. 74.1% of the respondents provide the exact numbers of total working hours they spent on their projects. We use the provided numbers to update the calculated project effort for model calibration and evaluation.

5.4.2 | Survey of the system components analysis results

One question is asked to understand which clustering of class units can best describe their actual system components.

SQ3: Which of the following clusterings best represents the actual logical system components of the app?

SQ3.1: For the best clustering of classes, what percentage of the classes is clustered in the correct components?

This survey question is used to validate the results of system components analysis introduced in Section 4.1.1. Three clusterings of class units are provided for the developers to select the one that can best represent their actual system components. 77.6% of the respondents provide a choice of the clustering that best represents the system components. We use the clustering the respondents provide to identify transactions for their projects. "Cohesion" is the mostly picked clustering, which is picked by 48.8% of the respondents. Therefore, for the projects that don't have the response for this survey question, we use "Cohesion" by default. 72.4% of the respondents provide answer for the follow-up question (SQ3.1), 57.1% of which agree that more than half of the class units are clustered into the correct system components. This follow-up question provides the evaluation about the confidence of the system components analysis method in identifying actual system components, which is 57.1% of the time the clustering algorithm can cluster more than half of the classes into actual system components.

5.5 | The Calibration Results

For each set of cut points for TL, TD, and DETs provided in Table 2 in Section 5.2, a classification function is defined using the method introduced in Section 4.2. Then, Bayesian analysis is used to calibrate the weights (W) that should be assigned to the complexity levels and the effort adjustment factor (α) to define a candidate model. We evaluate the performance of each candidate model using 4 typical estimation accuracy measures, including MMRE, PRED (.25), MDMRE, and MAE, which are listed below. MMRE, PRED, and MDMRE rely on the quantity called magnitude of relative error (MRE), which is defined as $MRE_i = \frac{|y_i - \hat{y_i}|}{y_i}$.

TABLE 3 The Rankings of the Candidate Models Defined based on 1-6 Quantile Discretization

Cut	MMRE	Rank1	PRED25	Rank2	MAE	Rank3	MDMRE	Rank4	Rank*
Cut_1	*0.92*	*3*	*0.14*	*2*	*2029.59*	*6*	*0.84*	*2*	*2*
Cut_2	*1.02*	*6*	*0.13*	*3*	*1641.82*	*3*	*0.87*	*3*	*4*
Cut_3	*0.81*	*1*	*0.17*	*1*	*1629.90*	*2*	*0.72*	*1*	*1*
Cut_4	*0.87*	*2*	*0.13*	*3*	*1883.02*	*5*	*0.87*	*3*	*2*
Cut_5	*0.99*	*5*	*0.07*	*5*	*1624.94*	*1*	*0.89*	*5*	*5*
Cut_6	*0.97*	*4*	*0.06*	*6*	*1845.02*	*4*	*0.96*	*6*	*6*

$$MMRE = \frac{1}{N}\sum_{i=1}^{N} MRE_i \qquad MDMRE = \frac{1}{2}(\lfloor MRE_{(n+1)\div 2}\rfloor + \lceil MRE_{(n+1)\div 2}\rceil) \qquad PRED(x) = \frac{1}{N}\sum_{i=1}^{N}\begin{cases}1, if\ MRE_i \le x\\ 0, otherwise\end{cases} \qquad MAE = \frac{\sum_{i=1}^{N}|x_1 - x_2|}{N}$$

MMRE and MDMRE measures the sample mean and median of MRE respectively. PRED (x) measures the percentage of the estimates with a threshold of MRE. In our evaluation, we usd PRED (.25), which is the mostly adopted measure based on PRED [21]. MAE, on the other hand, measures the sample mean of the absolute errors. Low values of MMRE, MDMRE, MAE, and high values of PRED (x) are desirable. These statistics give cost estimation practitioners the ability to state how often estimates can be expected to be within an acceptable margin of error.

As we can see from Table 3, the model under discretization of 3 bins provides the best performance since it provides the highest overall rank (*Rank**) in considering the individual ranks (*Rank1*, *Rank2*, *Rank3*, *and Rank4*) for the 4 types of accuracy measurements. Therefore, we use this candidate model as the final effort estimation model. The calibrated parameters of the final effort estimation model are provided in Table 4.

TABLE 4 The Calibrated Model Parameters

Parameters	Estimate	Standard Error	Confidence Interval	
			Lower Limit	Upper Limit
α	*8.35*	*3.45*	*3.97*	*15.85*
w_1	*0.91*	*0.89*	*0.27*	*2.66*
w_2	*1.53*	*0.97*	*0.41*	*3.60*
w_3	*2.73*	*0.72*	*1.05*	*3.96*
w_4	*3.92*	*1.47*	*0.52*	*6.37*
w_5	*6.47*	*2.42*	*0.90*	*10.09*
w_6	*9.96*	*3.65*	*1.53*	*16.38*
w_7	*15.41*	*6.40*	*1.78*	*26.80*
Classification Function:		$f^3_{cmplx_7}(t) = \sum_{d\in\{TL,TD,DETs\}} d(t) - 2$		
Cut Points:	Cut_1	Cut_2	Cut_3	Cut_4
TL	*0.0*	*3.8*	*4.8*	*Inf*
TD	*0.0*	*10.3*	*14.2*	*Inf*
DETs	*0.0*	*6.8*	*12.3*	*Inf*

The weights assigned to the different complexity levels represent their effects on project effort. The curve representing the effects of the 7 complexity levels on project effort is provided in Figure 10. In comparison with the prior weights that are assigned based on Fibonacci sequence, the empirical data adjust the weights towards more moderate non-linear increasing effects (**RQ1**).

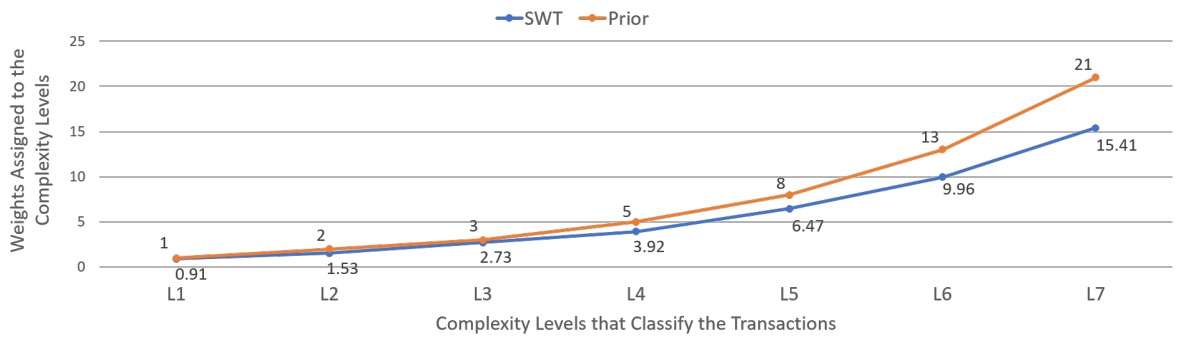

FIGURE 10 The Adjustment of the Weights using Bayesian Analysis

TABLE 5 The Correlation Analysis of the Size Metrics

Size Metric	*Pearson's r*	Effect Size	*p-value*	Rank
COSMIC	*0.55*	*Large*	*<0.01*	*1*
AFP	*0.38*	*Medium*	*0.03*	*2*
SLOC	*0.36*	*Medium*	*0.04*	*3*
UUCP	*0.32*	*Medium*	*0.07*	*4*

5.6 | Correlation Analysis of Size Metrics

To answer the second research question (**RQ2**) whether the proposed transaction-based software sizing model can be used to effectively measure software functional size, we evaluate the Pearson's correlation coefficients (r) between SWT and 4 typical size metrics, including Use Case Points (UUCP), IFPUG function points (AFP), COSMIC function points (COSMIC), and source lines of code (SLOC). The IFPUG function points, COSMIC function points, and Use Case Points measurements are derived from the transaction and use case analyses of the 34 projects using the methods introduced in studies [22] [23] [24]. SLOC is directly analyzed from the source code files using the code count tool UCC [25].

High correlation indicates the possibility of creating the backfiring tables [26], such that one metric can be mapped into another metric and used as the substitute of the other size metric in other effort estimation models. For instance, linear regression can be applied to determine how many lines of source code can be mapped into one IFPUG function point. A typical example of backfire table is the mapping between IFPUG function points and source lines of code used by COCOMO@II [27]. Using this backfiring table, COCOMO@II can be used to estimate project effort based on the information derived from functional requirements.

The correlation analysis results are presented in Table 5. The effect sizes are determined using the criteria proposed by Cohen [28] [29], by which 0.10 is rated as small, 0.30 is rated as medium, and 0.50 is rated as large. The high correlation between SWT and COSMIC indicates that SWT can be used as an effective substitute of COSMIC function points to measure software functional size. The medium effect sizes for the correlations between SWT and SLOC, AFP, and UUCP indicate the potential in creating a backfiring table by calibrating SWT specifically for those size metrics (**RQ2**).

5.7 | Goodness of Fit

To show the effectiveness of the proposed effort estimation model, we compare it with 4 baseline effort estimation models defined using the 4 size metrics introduced in Section 5.6, including IFPUG, COSMIC, UCP, and SLOC models. The definitions of the models are provided in Table 6. For IFPUG, COSMIC, UCP, and SLOC, least squares regression (LSR) are used to calibrate the effects of size metrics on project effort. For SWT, it is calibrated using the Bayesian analysis provided in Section 3.4.

We evaluate the goodness of fit of the proposed transaction-based effort estimation model and the baseline models using coefficient of determination R^2 to understand their ability in explaining the variability of project effort data. The higher the R^2 value is, the better an effort estimation model fits the dataset. The statistical significance of goodness of fit is evaluated using F-test.

The evaluation results of goodness of fit are presented in Table 6. As we can see, the SWT model provides the highest goodness of fit in comparison with the baseline models. R^2 of 0.47 means 47% of the variation of project effort can be explained by the model and the low p-value (0.02) suggests goodness of fit is statistically significant. Therefore, SWT model can be considered as effective in modeling Android project effort (**RQ3.1**).

TABLE 6 The Evaluation Results of Goodness of Fit

Model	Functional Form	Estimator	R^2	*p-value*	Rank
SWT	$e = a * \sum w_t$	*Bayes*	*0.47*	*0.02*	*1*
IFPUG	$e = a * afp + b$	*LSR*	*0.45*	*0.10*	*2*
COSMIC	$e = a * csmc + b$	*LSR*	*0.33*	*0.27*	*3*
SLOC	$e = a * sloc + b$	*LSR*	*0.21*	*0.45*	*4*
UCP	$e = a * uucp + b$	*LSR*	*0.17*	*0.78*	*5*

Bayes: Bayes Regression; *LSR*: Least Squares Regression;

TABLE 7 The Evaluation Results of the Out-of-sample Accuracy

Model	MMRE	Rank1	PRED25	Rank2	MDMRE	Rank3	MAE	Rank4	Rank*
SWT	*0.95*	*1*	*0.23*	*3*	*0.78*	*2*	*1654.32*	*1*	*1*
COSMIC	*1.43*	*3*	*0.29*	*1*	*0.74*	*1*	*2338.53*	*3*	*2*
IFPUG	*1.38*	*2*	*0.27*	*2*	*0.92*	*3*	*2191.31*	*2*	*3*
SLOC	*1.77*	*4*	*0.18*	*4*	*1.41*	*5*	*2503.11*	*4*	*4*
UCP	*2.63*	*5*	*0.17*	*5*	*1.17*	*4*	*2891.07*	*5*	*5*

5.8 | Evaluation of Estimation Accuracy

The model that explains the variation of project effort well is not necessarily able to predict new cases with high accuracy, due to the issue of overfitting [30]. Therefore, we evaluate the out-of-sample estimation accuracy of the proposed model to understand its performance in predicting unseen cases. In the following sections, we will introduce the cross-validation process to evaluate out-of-sample accuracy and the statistical significance analysis of the accuracy improvements.

5.8.1 | Out-of-sample Accuracy

Using the accuracy measures provided in Section 5.5, we run 10-fold cross validation to evaluate out-of-sample accuracy of the models, which tests the estimation accuracy of the models in predicting new cases. The execution of 10-fold cross validation starts by separating the dataset into 10 folds, among which 9 folds are used to train the models and 1 fold (exclusive from the 9 folds) is used to test the out-of-sample estimation accuracy of the trained models using the 4 accuracy measures. This model training and testing process is iterated for 10 times, and for each time a different set of 9 folds out of the 10 folds are used for training and the left 1 fold is used for testing. Therefore, 10 measurements are generated for each of the 4 accuracy measures: MMRE, MDMRE, PRED (.25), and MAE. The average of the 10 measurements is used as the indicator of the estimation accuracy of a model in terms of an accuracy measure.

The evaluation results are presented in Table 7. We rank the models individually for the 4 accuracy measures, which are shown as *Rank1*, *Rank2*, *Rank3*, *and Rank4*, and provide an overall rank (*Rank**) based on the individual ranks. SWT model achieves the highest ranks for MMRE and MAE, the second highest rank for MDMRE, and the third highest rank for PRED (.25), and it achieves the highest overall rank. Therefore, we conclude that the proposed transaction-based effort estimation model provides satisfactory accuracy (**RQ3.2**).

5.8.2 | Significance of Estimation Accuracy Improvements

In facing the limited data points and noise in the empirical data set, the model rankings may vary. To certify the accuracy improvements presented in Section 5.8.1 are not by chance, we further assess the statistical significance of the accuracy improvements.

As suggested by Nguyen et al. [21], the non-normality and unequal variances of the accuracy measures invalidate the typical parametric hypothesis tests for statistical comparison, such as Student's t-test. We employ the non-parametric bootstrapping tests to evaluate the statistical significance of the accuracy improvements. We first draw 10000 bootstrapping re-samples to construct the sampling distributions of the accuracy measurements, and then apply bootstrap-shift tests [31] to calculate the p-values for 40 ($= 4 * C_5^2$) comparisons. We adjust p-values using Bonferroni correction process to avoid the inflation of Type-I error in multiple comparisons [32]. In total, we find 26 comparisons are significant at 5% significance level, among which 13 significant comparisons are related to the SWT model, which are presented in Table 8. As we can see, SWT model outperforms

TABLE 8 The Statistical Significance Testing Results of Accuracy Improvements

ID	$Model_1$	$Model_2$	Metric	*P-Value*	ID	$Model_1$	$Model_2$	Metric	*P-Value*	ID	$Model_1$	$Model_2$	Metric	*P-Value*
1	*SWT*	*COSMIC*	*MMRE*	*0.03*	6	*SWT*	*SLOC*	*PRED25*	*0.01*	11	*SWT*	*UCP*	*MMRE*	*0.03*
2	*SWT*	*COSMIC*	*MAE*	*0.03*	7	*SWT*	*SLOC*	*MMRE*	*0.03*	12	*SWT*	*UCP*	*MDMRE*	*0.03*
3	*SWT*	*IFPUG*	*MMRE*	*0.02*	8	*SWT*	*SLOC*	*MDMRE*	*0.03*	13	*SWT*	*UCP*	*MAE*	*0.03*
4	*SWT*	*IFPUG*	*MDMRE*	*0.02*	9	*SWT*	*SLOC*	*MAE*	*0.03*					
5	*SWT*	*IFPUG*	*MAE*	*0.02*	10	*SWT*	*UCP*	*PRED25*	*0.01*					

COSMIC, IFPUG, SLOC, and UCP for MMRE and MAE at 5% significance level, while outperforms SLOC and UCP models for MDMRE and PRED (.25) at 5% significance level (**RQ3.3**).

6 | THREATS TO VALIDITY

This section discusses the threats to validity and the possible ways in which the threats can be mitigated.

6.1 | Threats to Internal Validity

One major threat to the internal validity is related to the incomplete reporting for the survey questions. As mentioned in Section 5.4, there are 10% of the participants responded to the survey questions. Therefore, the project effort is not fully validated to be accurate as the actual effort, and the system components are not completely validated to be the actual system components. This response rate can be improved by following up with the developers who haven't provided the response yet. We will update the survey results in the future runs of follow-up with the developers to mitigate this threat.

Another threat to internal validity is the limited data points we have collected. Using the 34 data points, the 10-fold cross-validation process applied in Section 5.8.1 only has 3 data points as the validation data set, which may cause certain degree of variation in the accuracy measurements. Also the statistical significance analysis, as introduced in Section 5.8.2, uses the Bonferroni correction procedure to adjust p-values to avoid the risk of inflation of type-I error under multiple comparison scenario, which requires a large data set to identify more statistically significant accuracy improvements.

6.2 | Threats to External Validity

The 4 baseline models employed in our comparative study is a limited set. Many other types of early-phase effort estimation models are not compared, such as Object Points [33], UML points [34], and Class Points [35]. We aim to integrate more effort estimation models into our model evaluation process to comprehensively assess their relative performance and statistical significance of the accuracy improvements in the future research to mitigate this threat. The evaluation results can provide practitioners with effective guidelines to choose appropriate effort estimation methods for Android projects.

Besides, the set of accuracy measures used to evaluate the out-of-sample accuracy of the models are limited to the 4 typically used accuracy measures. For a more comprehensive evaluation, size metrics, such as the ones based on absolute errors and squared errors, can be integrated into the model evaluation process, whose results can provide the practitioners with insights about the performance of the benchmark models under different criteria.

6.3 | Threats to Construct Validity

Simulation approaches are adopted to derive transactional and effort data. For example, as shown in Section 5.3, this study employs the effort data that is analyzed from the committing records of the developers and validated using survey responses. The project effort data collected in this way is considered to deviate from the actual effort spent on the projects. Using the simulated project effort data is because of the absence of the actual project effort. Therefore, the evaluation results about the estimation accuracy of the benchmark models and the statistical significance of the accuracy improvements need to be re-evaluated as actual effort data is available.

Besides, the accuracy evaluation results are sensitive to the clustering method employed to identify the system components. Due to our limited knowledge of the actual architecture of the subject Android projects, we use the hierarchical agglomerative clustering method to automatically identify the components, the results of which are considered to deviate from the actual system components. Further study of the influence on estimation accuracy using the actual system components or other components identification methods is needed.

7 | RELATED WORK

Many effort estimation methods have been reported to be successfully applied to mobile applications. The influential factors of software size and project effort specifically related to mobile applications and Android platform are identified and modeled by the existing methods. For example, Kaur et al.[1] proposed a modified version of Use Case Points called M-UCP to estimate project effort of Android applications, which considers

14 additional mobile application specific factors proposed by Souza et al.[36]. The 14 mobile application specific factors focus on non-functional requirements, for example, performance, power, data transmission band, connectivity, context, graphic interface, and input interface aspects of a mobile app. Different from their approach focusing on the effects that non-functional requirements have on project effort, we identify transactions from Android project source code to measure software functional size of Android applications and use the transactional information to estimate project effort.

Preuss[2] discussed her experience in using IFPUG function points to estimate the cost and duration of developing a mobile app called Terrific Tuner, by analyzing data and transactions of the app and mapping the elements into the measurement elements - ILF, EIF, EI, EO, and EQ - defined in the counting process of IFPUG. She concluded that IFPUG function points is an effective size metric for mobile applications. Ferrucci[37], Heeringen[38], and Nitze[39] succeeded in applying COSMIC function points to Android apps by focusing on different aspects in applying the COSMIC method. For instance, Ferruci found COSMIC function points can be an effective size metric for mobile apps based on the empirically evaluated high correlation between COSMIC function points and other physical size metrics, such as the number of Java, XML and Bytecode files of Android projects. Heeringen discussed a few assumptions he made to better apply COSMIC function points to mobile apps, including the rules to count COSMIC function points for the error messages and the rules to identify the counting scope, for example, by separating the application layer from the data layers that include network data, time, and GPS data. Nitze proposed to consider the interface complexity in the calculation of COSMIC function points, which is believed to better reflect the fact that the development of mobile apps is user-interface-focused. Different from the above-mentioned methods that focus on better applicability of the existing methods, our proposed functional size metric is directly defined based on the transactional information derived from the source code of Android project and the compiled APKs.

Effort estimation methods that are specifically developed for Android projects also have been proposed. For example, Francese[3] proposed a multiple linear regression model that considers the effects of a list of features identified from requirement documentations, design artifacts, and source code files on project effort. The list of considered features include McCabe cyclomatic complexity, number of classes, number of files, number of methods, number of all lines of source files, number of lines containing source code, number of lines containing comments, number of statements, number of functional requirements, number of use case actors, number of use cases, number of classes, number of sequence diagrams, number of XMI files about graphical elements. Their specific effects are determined by running multiple linear regression on 23 student team projects. Different from their approach, our study focuses on the complexity attributes of transactions that are influential to software functional size and project effort, which are calibrated using the transactional and project effort data derived from 34 open source Android projects.

8 | CONCLUSION AND FUTURE WORK

8.1 | Conclusions

In this paper, we propose an effective transaction-based effort estimation model for Android projects by extending the transaction analysis proposed in our previous research[5] to Android source code. We identify transactions, as well as their complexity attributes that are influential to software functional size, directly from the source code of Android projects using static and dynamic analyses. The transactions are classified into different complexity levels and a set of weights are applied to the complexity levels to differentiate their effects on project effort. The sum of weighted transactions is used to measure software functional size and estimate project effort. The model is calibrated and evaluated based on 34 open source Android projects.

We evaluate the correlation between our proposed size metric and 4 commonly used size metrics, such as Use Case Points, IFPUG and COSMIC function points, and SLOC, and find that SWT can be an effective substitute for COSMIC functional points and potentially for other studied size metrics. The proposed transaction-based effort estimation model is further compared with 4 baseline models that are defined based on the 4 typical size metrics. We compare the models in terms of their goodness of fit of our empirical data set and out-of-sample estimation accuracy, and evaluate statistical significance of the estimation accuracy improvements. The results show that the proposed transaction-based effort estimation model can provide statically significant accuracy improvements over the existing effort estimation models.

8.2 | Future Work

In the future, we aim to collect more transactional and project effort data from the sub-domains of Android projects, and calibrate specific models for the sub-domains. Therefore, models that are particularly effective for certain domains can be provided. We also would like to add more baseline effort estimation models in the comparison study and evaluate their out-of-sample accuracy with a more complete set of accuracy measures. The evaluation results may reveal valuable information to the decision makers for them to decide what effort estimation models can better fit their specific development environments.

The implication of the transaction analysis method proposed in this study on our previously proposed incremental effort estimation approach is outlined in Section 2, which is to add the missing link between post-development calibration and early effort estimation. Our current work focuses on the transaction analysis of Android projects, which is majorly because the analysis of source code is highly dependent on the programming languages and the frameworks used for software development. Therefore, our initial goal is to calibrate the proposed SWT-based effort estimation model specifically for Android projects and benchmark its effectiveness with other typical effort estimation models. To mitigate the limitations and provide better practical values, three future directions to further our current research are proposed as follows:

1. Comparing the early-phase models calibrated using early-phase transactional information and the early-phase models calibrated using post-development transactional information to evaluate the impact of this proposed post-development transaction analysis on estimation accuracy.

2. Applying the calibrated models to the early phases of real-world projects and evaluating their early-phase effort estimation accuracy.

3. Integrating the transaction analysis methods for other types of applications, including the technologies that analyse web applications, mobile applications, and desktop applications, such that we could collect a data set to fully represent the domain of application development and provide effective models to facilitate the development of modern applications.

References

1. Kaur A, Kaur K. Effort Estimation for Mobile Applications Using Use Case Point (UCP). In: Smart Innovations in Communication and Computational Sciences. Springer Singapore; 2019; Singapore: 163–172.
2. Preuss T. Mobile Applications, Function Points and Cost Estimating. In: International Cost Estimation Analysis Association. ; 2013.
3. Francese R, Gravino C, Risi M, Scanniello G, Tortora G. On the use of requirements measures to predict software project and product measures in the context of android mobile apps: a preliminary study. In: 2015 41st Euromicro Conference on Software Engineering and Advanced Applications. IEEE. ; 2015: 357–364.
4. Qi K, Boehm BW. Detailed use case points (DUCPs): a size metric automatically countable from sequence and class diagrams. In: Proceedings of the 10th International Workshop on Modelling in Software Engineering. ACM. ; 2018: 17–24.
5. Qi K, Boehm BW. Process-Driven Incremental Effort Estimation. In: 2019 IEEE/ACM International Conference on Software and System Processes (ICSSP). IEEE. ; 2019: 165–174.
6. Robles G, González-Barahona JM, Cervigón C, Capiluppi A, Izquierdo-Cortázar D. Estimating development effort in free/open source software projects by mining software repositories: a case study of openstack. In: Proceedings of the 11th Working Conference on Mining Software Repositories. ACM. ; 2014: 222–231.
7. Qi F, Jing XY, Zhu X, Xie X, Xu B, Ying S. Software effort estimation based on open source projects: Case study of Github. *Information and Software Technology* 2017; 92: 145–157.
8. Qi K, Boehm BW. A light-weight incremental effort estimation model for use case driven projects. In: Software Technology Conference (STC), 2017 IEEE 28th Annual. IEEE. ; 2017: 1–8.
9. Qi K, Boehm BW. UMLx: a UML diagram analytic tool for software management decisions. In: Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings. ; 2018: 278–279.
10. Qi K, Hira A, Venson E, Boehm BW. Calibrating use case points using bayesian analysis. In: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. ; 2018: 1–10.
11. Cui JF, Chae HS. Applying agglomerative hierarchical clustering algorithms to component identification for legacy systems. *Information and Software technology* 2011; 53(6): 601–614.
12. Tzerpos V, Holt RC. Accd: an algorithm for comprehension-driven clustering. In: Proceedings Seventh Working Conference on Reverse Engineering. IEEE. ; 2000: 258–267.
13. Van Deursen A, Kuipers T. Identifying objects using cluster and concept analysis. In: Proceedings of the 1999 International Conference on Software Engineering (IEEE Cat. No. 99CB37002). IEEE. ; 1999: 246–255.
14. Arzt S, Rasthofer S, Bodden E. Instrumenting android and java applications as easy as abc. In: International Conference on Runtime Verification. Springer. ; 2013: 364–381.

15. Jacobson I. *Object-oriented software engineering: a use case driven approach*. ACM Press;Addison-Wesley Pub . 1992.

16. Wikipedia . Espresso (framework) — Wikipedia, La enciclopedia libre. 2018. [Internet; descargado 26-junio-2019].

17. Yuan Y, Xu L, Xiao X, Podgurski A, Zhu H. RunDroid: recovering execution call graphs for Android applications. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. ACM. ; 2017: 949–953.

18. Yang S, Wu H, Zhang H, et al. Static Window Transition Graphs for Android. *International Journal of Automated Software Engineering* 2018; 25(4): 833-873.

19. Arzt S, Rasthofer S, Fritz C, et al. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *Acm Sigplan Notices* 2014; 49(6): 259–269.

20. Fossdroid.com . https://fossdroid.com. 2019.

21. Nguyen D, WVU TM. Studies of confidence in software cost estimation research based on the criterions mmre and pred. 2009.

22. Uemura T, Kusumoto S, Inoue K. Function point measurement tool for UML design specification. In: Software Metrics Symposium, 1999. Proceedings. Sixth International. IEEE. .

23. Jenner M. COSMIC-FFP and UML: Estimation of the Size of a System Specified in UML–Problems of Granularity. In: Fourth European Conference Soft. Measurement and ICT Control. ; 2001: 173–184.

24. Karner G. Resource estimation for objectory projects. *Objective Systems SF AB* 1993; 17.

25. Wikipedia contributors . Unified Code Count (UCC) — Wikipedia, The Free Encyclopedia. 2019. [Online; accessed 28-January-2020].

26. Dekkers C, Gunter I. Using Backfiring to Accurately Size Software: More Wishful Thinking Than Science?. *IT Metrics Strategies* 2000; 6(11): 1–8.

27. Boehm BW. *Software cost estimation with Cocomo II*. Upper Saddle River, NJ: Prentice Hall . 2000.

28. Cohen J. *Statistical power analysis for the behavioral sciences*. Routledge . 2013.

29. Kampenes VB, Dybå T, Hannay JE, Sjøberg DI. A systematic review of effect size in software engineering experiments. *Information and Software Technology* 2007; 49(11-12): 1073–1086.

30. O'Hagan A, Forster JJ. *Kendall's advanced theory of statistics, volume 2B: Bayesian inference*. 2. Arnold . 2004.

31. Smucker MD, Allan J, Carterette B. A comparison of statistical significance tests for information retrieval evaluation. In: Proceedings of the sixteenth ACM conference on Conference on information and knowledge management. ACM. ; 2007: 623–632.

32. Dunnett CW. A multiple comparison procedure for comparing several treatments with a control. *Journal of the American Statistical Association* 1955; 50(272): 1096–1121.

33. Minkiewicz A. Measuring Object Oriented Software with Predictive Object Points. *PRICE Systems, L.L.C* 2015.

34. Kim S, Lively WM, Simmons DB. An Effort Estimation by UML Points in Early Stage of Software Development.. In: Software Engineering Research and Practice. ; 2006: 415–421.

35. Costagliola G, Ferrucci F, Tortora G, Vitiello G. Class point: an approach for the size estimation of object-oriented systems. *IEEE Transactions on Software Engineering* 2005; 31(1): 52–74.

36. De Souza LS, Aquino Jr dG. Estimating the Effort of Mobile Application Development. In: Proceedings of Second International Conference on Computational Science and Engineering. ; 2014: 45–63.

37. Ferrucci F, Gravino C, Salza P, Sarro F. Investigating functional and code size measures for mobile applications. In: 2015 41st Euromicro Conference on Software Engineering and Advanced Applications. IEEE. ; 2015: 365–368.

38. Heeringen vH, Van Gorp E. Measure the functional size of a mobile app: Using the cosmic functional size measurement method. In: 2014 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement. IEEE. ; 2014: 11–16.

39. Nitze A. Measuring mobile application size using cosmic fp. In: DASMA Metrik Kongress. ; 2013.

How to cite this article: Kan Qi, Barry Boehm (2019), Effort Estimation of Open Source Android Projects via Transaction Analysis