Detailed Use Case Points (DUCPs) : A Size Metric Automatically Countable from Sequence and Class Diagrams

Kan Qi
University of Southern California
United States
kqi@usc.edu

Barry W. Boehm
University of Southern California
United States
boehm@usc.edu

ABSTRACT

Sequence and class diagrams are widely used to model the behavioral and structural aspects of a software system. A size metric that is defined automatically countable from sequence and class diagrams boosts both the efficiency and the accuracy of size estimation by producing reproducible software size measurements. To fulfill the purposes, a size metric called Detailed Use Case Points (DUCPs) is proposed based on the information automatically derived from sequence and class diagrams. The automation is largely supported by our proposed user-system interaction model (USIM) that fills the gap between the system abstraction by the sizing model and the metamodels of the UML diagrams. The effectiveness of our proposed size metric in project effort estimation is validated by our empirical study of 22 historical projects.

CCS CONCEPTS

• **Software and its engineering → Software development process management**; *Unified Modeling Language (UML)*;

KEYWORDS

Unified modeling language (UML), automated model transformation, model-based analysis, use case analysis, object-oriented modeling, automated analysis, software size metric, effort estimation, model calibration, use case points, function point analysis

ACM Reference Format:
Kan Qi and Barry W. Boehm. 2018. Detailed Use Case Points (DUCPs) : A Size Metric Automatically Countable from Sequence and Class Diagrams. In *MiSE'18: MiSE'18:IEEE/ACM 10th International Workshop on Modelling in Software Engineering , May 27, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3193954.3193955

1 INTRODUCTION

Effort estimation at the early stages of a project has been essential for a variety of software management decisions. For example, with project effort estimated, one can further estimate the schedule and the cost of a project to avoid the risks of being over schedule or budget. Most effort estimation models rely on a size metric as the major effort predictor [1] [2] [3]. Therefore, to ensure the utility of effort estimation in decision making, it is necessary for a software sizing model to estimate software size accurately at the early stages of a project.

Use Case Points (UCPs) has been widely used to provide size estimation for object oriented projects at the early stages. However, it also has been criticized to be inaccurate for not being particularly sensitive to the complexity of use cases [4] [5]. Besides, its manual process of counting transactions has been the barrier for its further use for estimating effort efficiently [6][7].

To deal with the difficulties, We define a size metric called DUCPs based on the original UCPs, which is automatically countable from sequence and class diagrams. The enabler of the automated counting process is the proposed user-system interaction model (USIM) that fills the gap between the sizing model's abstraction of a system and metamodels of sequence and class diagrams. The rules, data structures, and algorithms to construct this abstract model are clearly defined and elaborated. The effectiveness of the proposed metric in effort estimation is validated by an empirical study of 22 projects.

This paper is structured as follows. In section 2, we discuss the related work. Section 3 introduces the basic concepts. Section 4 formally defines the sizing model, the counting process of DUCPs, and USIM. In section 5, the rules, data structures, and algorithms adopted to support the automated procedures for model construction and size metric counting are presented. Section 6 presents the results from the empirical study of 22 projects, and demonstrates the effectiveness of DUCPs. Section 7 discusses the threads to validity. Section 8 concludes the research.

2 RELATED WORK

UML diagrams have been adopted in various approaches to sizing software systems, for example, Predictive Object Points (POP)[8], Class Points[9], UML Points[7], the Fast&Serious method [10], the metrics for eServices[11]. However, compared with the size metrics that are defined using UML primitives, our proposed metric, as type of functional point analysis (FPA), measures a system's functional size in terms of transactions, which is more theoretically and empirically validated in terms of its ability in project effort estimation[12][3][13] [14][15][16]. The automated methods for FPA also have been widely explored. For example, Fetcke proposed an approach to map use case model, domain object model, analysis model to FPA's abstraction of a software system [17]. Umeura proposed the rules to count Function Points from sequence and class diagrams[18]. However, the theoretical sizing models and the rules that map the UML's abstractions of a system to the sizing models adopted in their researches are fundamentally different from our

MiSE'18, May 27, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5735-7/18/05…$15.00
https://doi.org/10.1145/3193954.3193955

method. Previous researches on UCPs have been focused on classifying use cases with better granularity and adjusting the weighting schema via empirical studies to improve effort estimation accuracy [19][20][21][4][22][23], which rarely dealt with the inaccuracy brought by the insufficient information available in the abstract use case descriptions to estimating software size accurately, and the inefficiency brought by involving human effort in the counting process. Our approach integrates more detailed information from sequence and class diagrams to improve the accuracy and applies automated procedures to improve the efficiency.

3 BACKGROUND

3.1 Use Case Modeling With UML

In the context of Object-oriented Software Engineering (OOSE), functional requirements are captured by use cases, which are modeled by a UML use case model to provide high-level description about the interactions between a system and its actors [24]. Use cases can further be elaborated by sequence and class diagrams if object-oriented method is applied for detailed descriptions of a system's structural and behavioral aspects[25] [26]. The lifecycle of this model-based use case driven approach is presented in Figure 1. Examples of sequence diagram and class diagram are provided in Figure 2.

To support the exchange of the modeling information, XMI is invented and used as the standard interchangeable format for UML models. The XMI files exported from CASE Tools can generally be separated into two parts: $< uml : Model >$, which contains tagged elements $< packagedElement >$ to store information about the UML elements, and $< xmi : Extention >$, which is the extension made by a CASE tool to further characterize the UML elements. Each element is assigned with a UUID and can be referenced by other elements through its UUID. In this paper, we exploit the structure of the XMI files to extract relevant information that defines the proposed user-system interaction model (USIM) to support the automated software sizing procedures.

3.2 Use Case Points

To estimate the effort of use case driven projects, Karner developed Use Case Points (UCPs) based on his working experience at Objectory Systems [3]. Eq. (1) summarizes the four types of information that need to be evaluated from a project to count UCPs, which include unadjusted use case weight (UUCW), unadjusted actor weight (UAW), environmental factor (EF), and technical complexity factor (TCF). The major effort for calculating UCPs is to count the number of transactions of the use cases and use the number to classify the use cases into three complexity levels in order to calculate UUCW. After UCPs is calculated, it is multiplied by a productivity factor (20 is suggested by the author) to estimate the project effort (in person hours). Local calibration is encouraged by the author and other researchers to improve the accuracy of effort estimation.

$$UCP = (UUCW + UAW) * TCF * EF \tag{1}$$

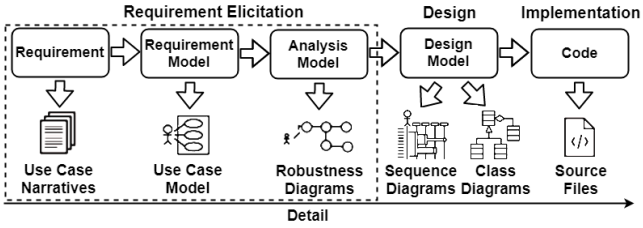

Figure 1: The use case driven process and its deliverables.

4 MODEL DEFINITION

4.1 The Sizing Model

The original UCPs weights each use case by the number of its internal transactions and uses the sum of the weighted use cases as the size measurement of system functionality. This sizing model can be summarized by Eq. (2).

$$Size = \sum_{c \in C} W_c \tag{2}$$

However, as system design being further detailed by the design and analysis activities, for example, using sequence and class diagrams to model a system's behavior and structure, we are able to understand the internal structure of an identified transaction, for example, the activities a transaction will be implemented upon and the data elements a transaction references. Those properties help classify transactions into different levels of complexity, which can be weighted differently in size estimation to represent their different levels of difficulty in implementation, testing, and maintenance. For this reason, we propose using the sum of weighted transactions to achieve better granularity in software sizing, which is formalized by Eq. (3).

$$Size = \sum_{c \in C} \sum_{t \in c} w_t \tag{3}$$

This proposed sizing model models a system as a set of transactions and the measurement of a system's size requires two basic procedures - transaction identification and classification. Conceptually, a transaction can be interpreted as a sequence of system operations to realize a basic unit of a system's functionality. A transaction may be identified differently based on different artifacts. In terms of sequence diagrams, a transaction (defined in Definition 1) is defined as a sequence of messages that represents a type of user-system interaction. Such a sequence of messages can further be modeled as the methods of the classes of the class diagrams that describe the structure of a system. The transactions that can be identified from the previous sequence diagram example are shown in (c) of Figure 5.

Definition 1. *(Transaction) A transaction is a sequence of messages identified from sequence diagrams, such that this sequence fulfills a type of interaction between an actor and a system.*

4.2 User-system Interaction Model (USIM)

To support the automated procedures for transaction identification and classification, we further define a user-system interaction

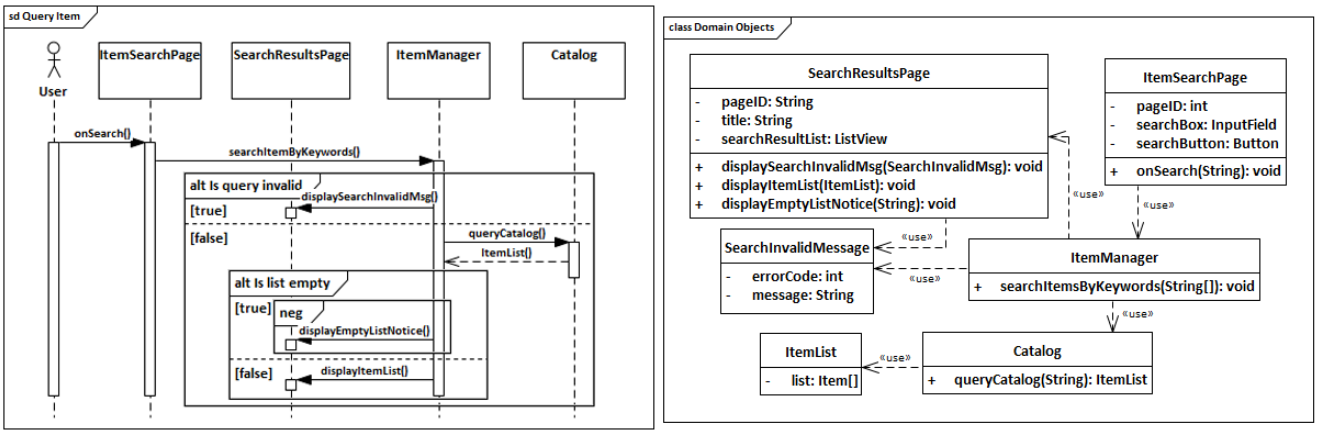

Figure 2: Examples of sequence and class diagrams.

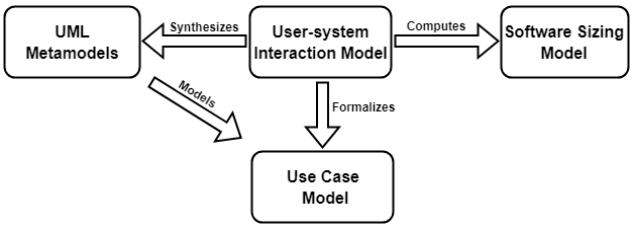

Figure 3: Relationships between Use Case Model, UML metamodels, USIM, and the software sizing model.

model, called USIM, to synthesize the information derived from the sequence and class diagrams based on their metamodels, and construct this abstract model by parsing the XMI files exported from CASE tools. The difference between USIM and the UML use case model [24] is that the UML use case model defines user-system interactions at the level of identifiable user actions and system reactions (exposed by user interfaces), while USIM defines the system reactions at the level of the operations among system components and computationally supports the sizing model. Figure 3 summarizes the relationships between UML metamodels, Use Case Model, the sizing model, and USIM. USIM can be denoted by a tuple $< A, PR, SB, SCP, STL, CMP >$, which includes 6 types of elements - activities (A), precedence relations (PR), system boundary (SB), system scope (SCP), stimuli (STL), and components (CMP). The details of constructing an instance of USIM are elaborated in section 5.1.

4.3 The Counting Process of DUCPs

Based on the proposed software sizing model and information computable from USIM, we define a size metric called DUCPs to estimate the software size. DUCPs reuses UCPs' approach to evaluating the project properties, such as Unadjusted Actor Weight (UAW), Environmental Factor (EF), and Technical Complexity Factor (TCF), while it uses the automated approach to identifying and classifying transactions. The steps of counting DUCPs are listed below:

Table 1: Transactional Complexity Levels and Weights

DET/TL	1-3	4-7	≥8
0-10	1 (Very Low)	2 (Low)	5 (Average)
11-25	2 (Low)	5 (Average)	13 (High)
≥26	5 (Average)	13 (High)	21 (Very High)

(1) Identify the transactions from sequence diagrams for each use case of the system.
(2) Count the transaction length (TL) and the data element types (DETs) for each identified transaction.
(3) Determine the Unadjusted Detailed Transaction Weight (UDTW) for each transaction according to Table 1 based on the TL and the DETs.
(4) Calculate the Unadjusted Detailed Use Case Weight (UDUCW) by Eq. (4)

$$UDUCW = \sum_{c \in C} \sum_{t \in T_c} UDTW(t) \tag{4}$$

(5) Evaluate the technical complexity factor (TCF) and the environmental factor (EF) based on the original UCPs.
(6) Calculate the DUCPs by Eq. (5).

$$DUCP = (UDUCW + UAW) * TCF * EF \tag{5}$$

5 THE AUTOMATED COUNTING FRAMEWORK

5.1 User-system Interaction Model (USIM) Construction

5.1.1 Extracting information from XMI files. The goal for this step is to parse the tagged elements of the XMI files into the UML elements that are defined in the metamodels of sequence and class diagrams and replace the associations between the tagged elements based on their UUIDs by the references between the objects that represent the UML elements. The UML elements can be identified

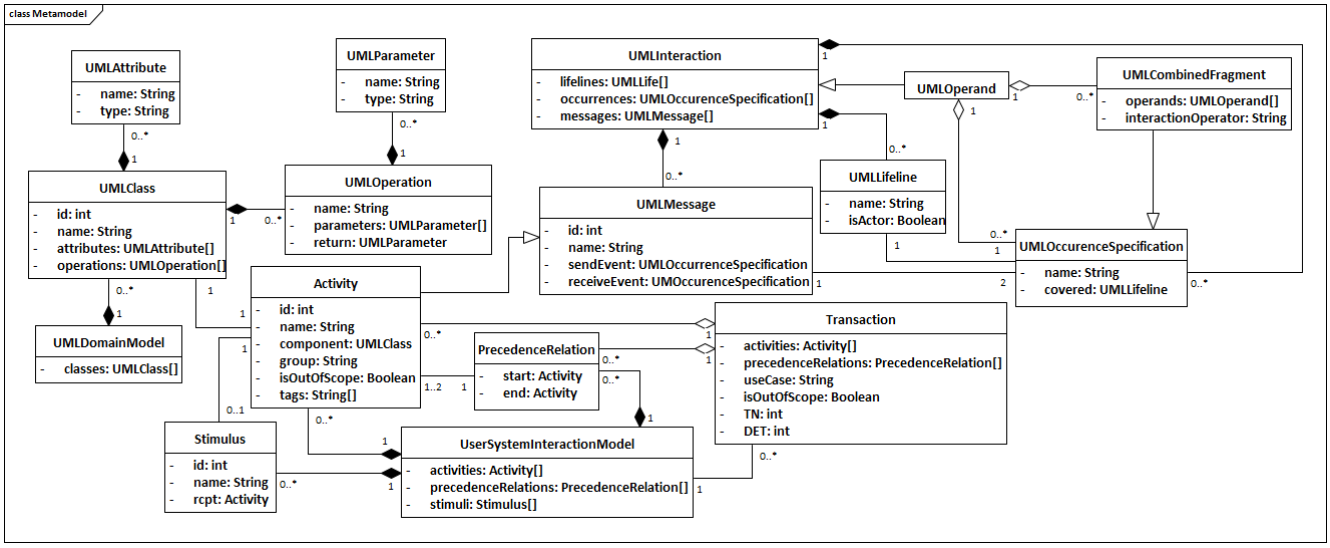

Figure 4: The metamodel for User-system Interaction Model (USIM).

Table 2: Rules Mapping Tagged Elements to UML Elements

UML Element	XMI Tag	XMI Type
UMLClass	<packagedElement>	uml:Class
UMLAttribute	<ownedAttribute>	uml:Property
UMLOperation	<ownedOperation>	–
UMLParameter	<ownedParameter>	–
UMLInteraction	<ownedBehavior>	uml:Interaction
UMLLifeLine	<lifeline>	uml:Lifeline
UMLMessage	<message>	uml:Message
UMLOccurrence	<fragment>	uml:Occurrence-Specification
UMLCombined-Fragment	<fragment>	uml:Combined-Fragment
UMLOperand	<operand>	uml:Interaction-Operand
UMLUseCase	<packagedElement>	uml:UseCase

by the combination of the tag and the XMI type. The rules for mapping the tagged elements (of specific "XMI" types) into the UML elements are presented in Table 2. The structure of XMI also stores the information about the hierarchy of the UML elements. For example, UMLCombinedFragment objects are recursively defined by its inner UMLCombinedFragment and UMLOccurrence objects. The result is an instance of the metamodels of sequence and class diagrams presented in Figure 4, which is used to formally define a USIM instance.

5.1.2 Control Flow Construction (A and PR). The underlying structure of USIM is the control flow among the activities. To derive the control flow graph, we apply a graph transformation algorithm to abstract messages into activities and determine the precedence relations of the activities. We call this directed graph SDCFG. To be compatible with UML2.0 standard (in considering the different types of fragments), Kundu proposed an algorithm that converts sequence diagrams into control flow graphs based on XMI files, and used extra fragment nodes ($fragment_{start}$, $fragment_{end}$) to maintain the semantics of the fragments [27]. In our algorithm, we adopt this idea and use extra fragment nodes for the same purpose. However, different from their algorithm, we use the occurrences to identify the order of the messages, and deal with the nested fragments by recursive processing of the fragments. Also, we eliminate those extra fragment nodes afterwards to simplify the CFG, while keeping the semantics of the fragments in terms of their influences on the control flow. The algorithm of constructing SDCFG is presented in Algorithm 1, which essentially defines the mapping (summarized by Eq. (6)) between the messages (M) and the activities (A), and the precedence relations (PR) among A. The SDCFG converted from the example sequence diagram is presented in (b) of Figure 5.

$$msg(\alpha) = \{m | \alpha.id = m.id \ \& \ m \in M \ \& \ \alpha \in A\} \tag{6}$$

5.1.3 Identifying Components (CMP). To identify the components that implement the activities, we identify the classes (C) from class diagrams, which model the lifelines (L) covered by the occurrences (O) that are the "receiveEvents" of the messages corresponding to the activities. For each activity, we search through the domain model by comparing the standardized names ($sName$) of such lifelines with the standardized names of the classes in the domain model, to establish the association between the class and the activity that regards it as the component. To standardize a name, it is first capitalized and then has the spaces removed. The practical consideration for using the standardized names in the comparison is to prevent the inconsistency due to using different names to

Algorithm 1: Convert Sequence Diagram to SDCFG

Data: $SD < M, L, O, F >$ - the sequence diagram needs to be transformed
Result: $SDCFG < A, PR >$ - the CFG converted from the sequence diagram

Function ***constructSDCFG*** $SD < M, L, O, F >$ // M:messages, L:lifelines, O:occurrences, F:fragments
  Initialize $SDCFG < A, PR, startA, endA >$; $preA \leftarrow$ NULL ; // A:activities, PR:precedence relations
  while *Dequeue* o_1 *from* $SD.O$ **do**
    if $o_1 \in SD.F$ **then** // if o_1 is a combined fragment
      $< A', PR', startA', endA' > \leftarrow$ procesCombinedFragment(o_1);
      Push A', PR' into A, PR; push $\{start : preA, end : startA'\}$ into PR if $preA \neq$ NULL; $preA \leftarrow endA'$;
      $startA \leftarrow startA'$ if $startA$ = NULL;
    else // if o_1 is an occurrence
      Dequeue o_2 from $SD.O$ and create an activity α for msg m such that $o_1 = m.sendEvent \& o_2 = m.receiveEvent \& m.id = \alpha.id$;
      Push α into A; push $\{start : preA, end : \alpha\}$ into PR if $preA \neq$ NULL; $preA \leftarrow \alpha$; $startA \leftarrow preA$ if $startA$ = NULL;
  return $< A, PR, startA, endA >$;
Function ***processCombinedFragment*** F
  Initialize $F < A, PR, startA, endA >$; initialize and push $CFStart$, $CFEnd$ into A ; // $CFStart$, $CFEnd$:extra fragment nodes
  for *each operand* OP *of* F **do**
    $< A', PR', startA', endA' > \leftarrow$ constructSDCFG($SD' < L, M, O', F' >$) ; // O', F':occurrences, fragments of OP;
    Push $< start : CFStart, end : startA' >$ into PR; push A', PR' into A, PR; push $< star : endA', end : CFEnd >$ into PR;
  if OPR is "loop" **then** push $\{start : CFEnd, end : CFStart\}$ into PR; $startA \leftarrow CFStart$; $endA \leftarrow CFStart$;
  if OPR is "opt" **then** push $\{start : CFStart, end : CFEnd\}$ into PR; $startA \leftarrow CFStart$; $endA \leftarrow CFEnd$;
  if OPR is "break" **then** $startA \leftarrow CFStart$; $endA \leftarrow CFStart$; // OPR:operator of F
  else $startA \leftarrow CFStart$; $endA \leftarrow CFEnd$;
  return $< A, PR, startA, endA >$;

reference the same object during the modeling process. The rule for identifying components is summarized in Eq. (7).

$$cmp(\alpha) = \{c | m.receiveEvent = o \ \& \ o.covered = l \ \& \ l.sName = c.sName \ \& \ msg(\alpha) = m \ \& \ o \in O \ \& \ l \in L \ \& \ c \in C\} \quad (7)$$

5.1.4 Determining System Boundary (SB). In sequence diagrams, actor, a special type of lifeline, is used to model the users and external systems. Since the activities in USIM model the receivers' realizations of the requests sent by the senders (a message represents both the request and the realization), if the receiver of a message is an actor, the corresponding activity is determined to be outside of the system boundary (*SB*). Also, the outside activities can be categorized by the actors they are associated with. To capture the information of system boundary, an attribute *group* is attached to the activities to indicate the groups the activities belong to. The activities with *group* being "system" are the activities within the system boundary. The rules to determine the group of an activity and the system boundary are summarized in Eq. (8) and Eq. (9).

$$grp(\alpha) = \{l.name | m.receiveEvent = o \ \& \ o.covered = l \ \& \ l.isActor = true \ \& \ msg(\alpha) = m \ \& \ o \in O \ \& \ l \in L\} \quad (8)$$

$$A_{system} = A - \cup_{\{} g \in G\{\alpha | grp(\alpha) = g\} \quad (9)$$

5.1.5 Defining System Scope (SCP). We identify system scope (*SCP*) by setting the attribute $isOutOfScope$ of the activities as $true$ if they are beyond the scope. The activities that are beyond the scope are the activities that exist in the design but need not to be implemented, and the transactions are defined by the activities out of system scope are ineffective transactions. The number of effective transactions is considered as the functional size of a system according to our proposed sizing model. Here we especially consider the fragments that have the effects on the number of effective transactions of a system. To identify the effective transactions, each activity maintains an attribute called $isOutOfScope$, the value of which is determined by its $fragments$ attribute that maintains the information about the fragments the activity belongs to. An activity may be nested within multiple fragments. For example, the message corresponding to the activity is located in a "loop" fragment that is nested within an "option" fragment. In this case, the activity is tagged with both "loop" and "option" by setting $a.frags = [loop, option]$. This tagging process can be easily implemented during the control flow construction process. There are 12 types of fragments an activity can be tagged with according to UML2. Only "ignore" and "negative" fragments have the semantics that the activities need not be implemented. Therefore, the activities that are tagged by "ignore", "negative", or both in the $frags$ attribute are determined as "out of the scope" activities and their $isOutOfScope$ attribute is set true. The rule is summarized in Eq. (10).

$$isOutOfScope(\alpha) = (\text{"}option\text{"} \in \alpha.tags \ || \ \text{"}negative\text{"} \in \alpha.tags) \quad (10)$$

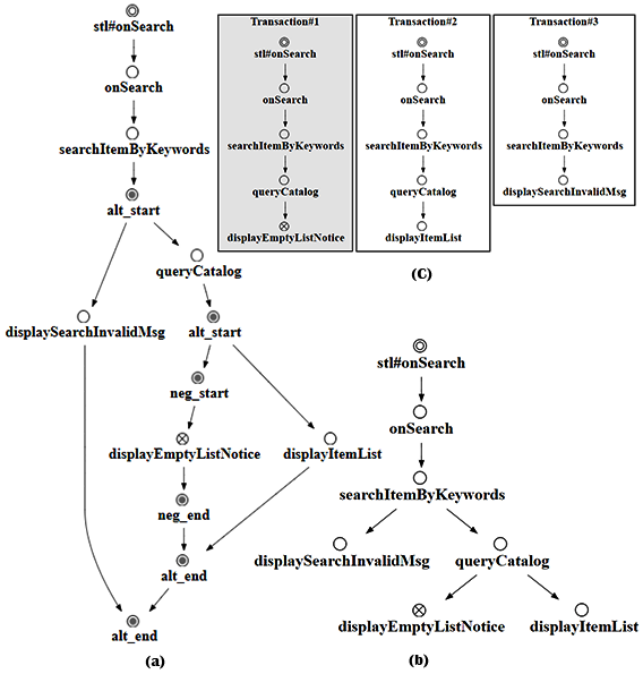

Figure 5: An example of control flow graph construction and eliminating extra fragment nodes: (a). SDCFG is constructed from a sequence diagram. (b). Fragment nodes are eliminated from SDCFG. (c). Transactions are identified by the graph traversing algorithm

5.1.6 Determining Stimuli (STL). According the definition of transaction, a transaction represents a system's reaction to an action of an actor. Actors' actions are abstracted into the stimuli (*STL*) of the system, which are used as the entry nodes in the path searching algorithm for identifying transactions. Still, since the activities model the realizations of the requests, we first identify the responses (*RSP*) of the stimuli from the identified activities, and then create stimulus nodes for the responses. The rules to determine responses and create stimuli are summarized in Eq. (11) and Eq. (12).

$$RSP = \{\alpha | m.sendEvent = o \ \& \ o.covered = l \ \& \ l.isActor = true \ \& \ msg(a) = m \ \& \ o \in O \ \& \ l \in L\} \quad (11)$$

$$STL = \cup_{\alpha \in RSP}\{\alpha' | \alpha'.rcpt = \alpha \ \& \ \alpha' id = m.id \ \& \ m \in M\} \quad (12)$$

5.1.7 Visualizing the system interaction model. To visualize USIM, we define a set of notations to denote the key elements. Figure 6 provides an example of visualizing the USIM derived from the example sequence and class diagrams given in Figure 2.

5.2 Transaction Identification

After an instance of USIM is constructed, we apply an algorithm based on DFS to search for the paths from the instance with customized rules to determine entry and exit conditions to ensure a path is a transaction. The entry nodes are the stimulus nodes (*STL*),

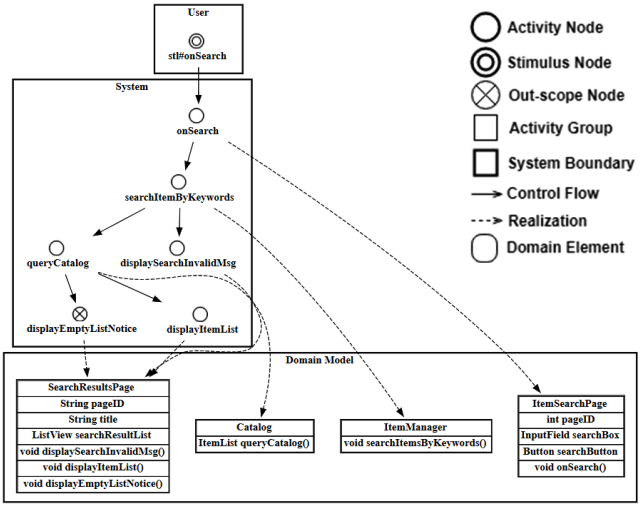

Figure 6: An example of visualizing USIM.

while the exit nodes are the nodes that are not within the system boundary (*SB*) or have no succeeding nodes. The path between a stimulus node and an exit node is output as a transaction. The algorithm is pretty straightforward, we skip it to save the space.

5.3 Transaction Classification

The calculation of DUCPs involves the three kinds of classifications as follows.

5.3.1 classification by use cases. Since usually a use case is modeled by one or multiple sequence diagrams and this hierarchy is preserved in the structure of the XMI files, if a transaction is identified from a sequence diagram that models a use case, it is counted as a transaction of that use case. This information is captured by the *useCase* attribute of a transaction to indicate which use case the transaction belongs to. The transactions of a use case is defined by Eq. (13)

$$T_{uc} = \{t | t.useCase = uc\} \quad (13)$$

5.3.2 classification by the isOutOfScope attribute. We employ the *isOutOfScope* attribute of the activities to identify effective transactions, which is, if the *isOutOfScope* attribute of any of the activities within a transaction is true, the *isOutOfScope* attribute of the transaction is set true and determined as not effective. The rule is summarized in Eq. (14).

$$isOutOfScope(t) = OR_{\alpha \in t}(\alpha.isOutOfScope = true) \quad (14)$$

A transaction determined as ineffective means the transaction is designed not to be implemented. Therefore, it will be excluded from size estimation. The effective transactions (T^*_{uc}) of a use case is defined by Eq. (15).

$$T^*_{uc} = T_{uc} - \cup_{t \in T_{uc}}\{t | isOutOfScope(t) = true\} \quad (15)$$

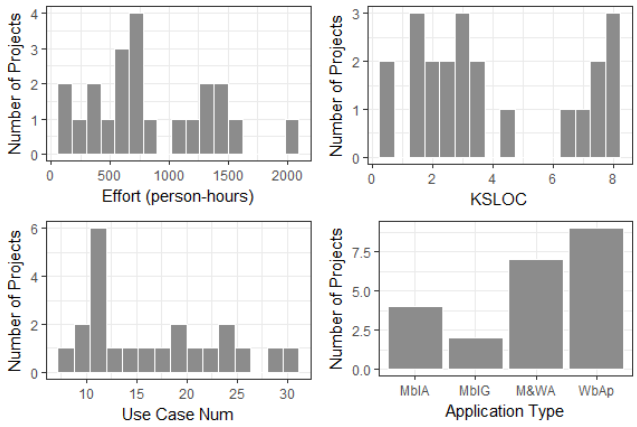

Figure 7: Project Descriptive Statistics

5.3.3 classification by complexity. Based on the structure of the transactions identified from sequence and class diagrams, we identify two factors that affect the effort in implementing, testing, and maintaining a transaction - transaction length (TL) and data element types (DETs), which are formally defined in Definition 2 and Definition 3.

Definition 2. *(TL) Transaction length is defined as the number of activities that realize a transaction, which models the procedural complexity of a transaction.*

Definition 3. *(DET) A data element type is defined as an identifiable elementary type of data that is referenced by a transaction, which models an input or an output during the processing of a transaction.*

Counting TL is straightforward based on the transactions identified by our method. It is the length of the path identified by the graph traversing algorithm. For the two effective transactions given in (c) of Figure 5, the TLs are 4 and 3 respectively.

DETs of a transaction is counted as the total number of the parameters and return variables of the methods of the classes that represent the components of the transaction's activities. For example, if a method corresponding to an activity α_1 has $P(\alpha_1)$ different parameters and 1 return variable, the DETs for this message is counted as $P(\alpha_1) + 1$. If a transaction contains n such activities, the DETs of the transaction is counted as $\sum_{i \in n} (P(a_i) + 1) = \sum_{i \in n} P(a_i) + n$. For the two effective transactions given in (c) of Figure 5, the DETs are 5 and 3 respectively.

Based on TL and DETs, transactions are classified into five levels of complexity and weighted differently by Table 1 to represent their relative influences on software size. The influences are generally agreed to be non-linear[1]. Here, we assign the weights based on Fibonacci sequence, which is similarly used in Story Points. Therefore, UDTW can be calculated by Eq. (16)

$$UDTW(t) = \sum_{t \in T^*_{uc}} w(TL(t), DETs(t)) \tag{16}$$

Table 3: SE, T-values, P-values for the Parameters

Parameter	est.	std. error	t-value	p-value
β_0	11.57	147.37	0.08	0.94
β_1	3.14	0.50	6.33	0.36e-05

6 MODEL CALIBRATION AND EVALUATION

6.1 Data Collection

22 projects were collected from master-level software engineering courses to apply the counting method of DUCPs and that of UCPs. The projects are either about developing web applications or mobile applications, which range from 1-9 KSLOC. The empirical study was done by applying the proposed automated approach to the artifacts collected from those projects. The distributions of the project characteristics are presented in Figure 7.

6.2 Model Calibration

We first calculated the correlation efficient (r) between DUCPs and project effort (in person-hours) to understand if linear relationship exists between them. The value of r being 0.67 certifies the existence of the linear relationship. Also, the result from the hypothesis test on β_1 (the $p-value$ being close to 0, given in Table 3) suggests that the linear relationship between the size metric and effort is significant. After that, least squares regression is applied to calibrate the specific effect a unit of DUCPs has on project effort. The calibrated results are presented in Table 3, which suggests that one unit of DUCPs is about 3.14 person-hours, while on average there is an overhead of 11.57 person-hours for use case driven approach of software engineering - the part of effort that is not related to developing, modifying, or testing any use case.

6.3 Model Evaluation

To evaluate the performance of DUCPs, a comparison between DUCPs and the original UCPs was conducted in terms of the out-of-sample effort estimation accuracy, for which 5-fold cross validation was performed by separating the 22 data points into 5 folds. Specifically, 5 runs of model training and testing were conducted, and the average of the testing results of the 5 runs was used as the estimate of the effort predication accuracy. In each run of cross validation, 4 subsets (17-18 data points) are used as the training data set to calibrate both UCPs and DUCPs models, and 1 subset (4-5 data points) is used as the testing dataset to calculate MMRE, PRED(.15), PRED(.25), PRED(.50), which are the commonly used metrics to evaluate prediction accuracy for effort estimation models [1] [28]. Both MMRE and PRED rely on the quantity called magnitude relative error (MRE) defined by Eq. (17). MMRE measures the sample mean of MRE, while PRED(x) measures the percentage of MRE within x. The results are presented in Table 4. In summary, DUCPs is about 21% better for MMRE, 14% better for PRED(.15), 14% better for PRED(.25), and 15% better for PRED(.50), than UCPs.

$$MRE = \frac{|y - \hat{y}|}{y} \tag{17}$$

Table 4: MMRE, PRED(.15), PRED(.25), PRED(.50) for 5-fold Cross Validation

	DUCPs				UCPs			
Run	M.	P.(.15)	P.(.25)	P.(.50)	M.	P.(.15)	P.(.25)	P.(.50)
1	0.45	0.00	0.20	0.60	0.61	0.00	0.20	0.60
2	0.19	0.25	0.75	1.00	0.56	0.00	0.50	1.50
3	0.30	0.25	0.25	1.00	0.57	0.00	0.00	0.75
4	0.37	0.25	0.50	0.50	0.59	0.25	0.50	0.50
5	0.21	0.20	0.60	1.00	0.28	0.00	0.40	1.00
Avg.	0.31	0.19	0.46	0.82	0.52	0.05	0.32	0.67

7 THREADS TO VALIDITY

Since, as shown in Figure 7, the studied projects are considered small projects for the sizes range from 1-9 KSLOC and they were done with 5-8 team members, the results of the accuracy evaluation presented in this paper may not be directly applicable to larger projects. Also, in the five-fold cross validation, only 4-5 data points were used as the testing dataset. More data points for testing are desirable to make the conclusion about the estimation accuracy and the superiority of DUCPs over UCPs.

8 CONCLUSIONS

To summarize, we first propose a software sizing model that is applicable to use case driven projects. Based on the sizing model, a size metric called DUCPs is defined, which sizes software in better granularity and efficiency by adopting the automated approaches of identifying and classifying transactions. To facilitate the automated procedures, we formally define a user-system interaction model, called USIM, to bridge the gap between the sizing model and the metamodels of sequence and class diagrams. The rules, data structures, and algorithms to construct this abstract model are clearly defined and explained. The significance of USIM is its potential of abstracting other types of UML diagrams, for example, activity diagrams, object analysis diagrams, state machines, etc, and integrate them into our evaluation paradigm. This would be a major direction for your future study. Also, the empirical study of 22 projects showed DUCPs outperforms the original UCPs 21% for MMRE, 14% for PRED(.15), 14% for PRED(.25), and 15% for PRED(.50) in the out-of-sample test. However, due to the nature of the collected data points used in the empirical study, further application to other software development environments, for example, to larger projects with more team members, requires local calibration with the data points representative for the specific environments. Therefore, to collect more data to further prove and improve the effectiveness of the size metric is another major direction of the research.

REFERENCES

[1] Barry W. Boehm. *Software cost estimation with Cocomo II.* Prentice Hall, Upper Saddle River, NJ, 2000.

[2] Allan J Albrecht. Measuring application development productivity. In *Proc. of the Joint SHARE/GUIDE/IBM Applicaiton Development Symposium*, pages 83–92, 1979.

[3] Gustav Karner. Resource estimation for objectory projects. *Objective Systems SF AB*, 17, 1993.

[4] Ali Bou Nassif, Luiz Fernando Capretz, and Danny Ho. Enhancing use case points estimation method using soft computing techniques. *arXiv preprint arXiv:1612.01078*, 2016.

[5] Mohammed Wajahat Kamal and Moataz A Ahmed. A proposed framework for use case based effort estimation using fuzzy logic: building upon the outcomes of a systematic literature review. *International Journal of New Computer Architectures and their Applications (IJNCAA)*, 1(4):953–976, 2011.

[6] Meenakshi Saroha and Shashank Sahu. Tools & methods for software effort estimation using use case points model—a review. In *Computing, Communication & Automation (ICCCA), 2015 International Conference on*, pages 874–879. IEEE, 2015.

[7] SangEun Kim, William M Lively, and Dick B Simmons. An effort estimation by uml points in early stage of software development. In *Software Engineering Research and Practice*, pages 415–421, 2006.

[8] Arlene Minkiewicz. Measuring object oriented software with predictive object points. *PRICE Systems, LLC*, 1997.

[9] Gennaro Costagliola, Filomena Ferrucci, Genoveffa Tortora, and Giuliana Vitiello. Class point: an approach for the size estimation of object-oriented systems. *IEEE Transactions on Software Engineering*, 31(1):52–74, 2005.

[10] Massimo Carbone and Giuseppe Santucci. Fast&&serious: a uml based metric for effort estimation. In *Proceedings of the 6th ECOOP workshop on quantitative approaches in object-oriented software engineering (QAOOSE'02)*, pages 313–322, 2002.

[11] Yue Chen, Barry W Boehm, Ray Madachy, and Ricardo Valerdi. An empirical study of eservices product uml sizing metrics. In *Empirical Software Engineering, 2004. ISESE'04. Proceedings. 2004 International Symposium on*, pages 199–206. IEEE, 2004.

[12] Aleš Živkovič, Ivan Rozman, and Marjan Heričko. Automated software size estimation based on function points using uml models. *Information and Software Technology*, 47(13):881–890, 2005.

[13] S Oligny, A Abran, and C Symons. Cosmic-ffp some results from the field trials. In *15th International Forum on COCOMO and Software Cost Estimation*, 2000.

[14] Thomas Fetcke. A generalized structure for function point analysis. *Proc. of the 11th IWSM, Lac Superieur, Canada*, pages 143–153, 1999.

[15] Gabriela Robiolo, Cristina Badano, and Ricardo Orosco. Transactions and paths: Two use case based metrics which improve the early effort estimation. In *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pages 422–425. IEEE Computer Society, 2009.

[16] Kan Qi and Barry W Boehm. A light-weight incremental effort estimation model for use case driven projects. In *Software Technology Conference (STC), 2017 IEEE 28th Annual*, pages 1–8. IEEE, 2017.

[17] Thomas Fetcke, Alain Abran, and Tho-Hau Nguyen. Mapping the oo-jacobson approach to function point analysis. In *Software Metrics*, pages 59–73. Springer, 1997.

[18] Takuya Uemura, Shinji Kusumoto, and Katsuro Inoue. Function point measurement tool for uml design specification. In *Software Metrics Symposium, 1999. Proceedings. Sixth International*, pages 62–69. IEEE, 1999.

[19] Kasi Periyasamy and Aditi Ghode. Cost estimation using extended use case point (e-ucp) model. In *Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference on*, pages 1–5. IEEE, 2009.

[20] Fan Wang, Xiaohu Yang, Xiaochun Zhu, and Lu Chen. Extended use case points method for software cost estimation. In *Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference on*, pages 1–5. IEEE, 2009.

[21] Ali Bou Nassif. Software size and effort estimation from use case diagrams using regression and soft computing models. 2012.

[22] Mudasir Manzoor Kirmani and Abdul Wahid. Revised use case point (re-ucp) model for software effort estimation. *International Journal of Advanced Computer Science and Applications*, 6(3):65–71, 2015.

[23] Divya Kashyap, Durgesh Shukla, and AK Misra. Refining the use case classification for use case point method for software effort estimation. In *Int. Conf. on Recent Trends in Information, Telecommunication and Computing, ITC*, pages 183–191, 2014.

[24] Ivar Jacobson. *Object-oriented software engineering: a use case driven approach.* ACM Press ;Addison-Wesley Pub, [New York] :Wokingham, Eng. ;Reading, Mass, 1992.

[25] Doug Rosenberg, Barry Boehm, Bo Wang, and Kan Qi. Rapid, evolutionary, reliable, scalable system and software development: The resilient agile process. In *Proc. of the ICSSP'17*, 2017.

[26] Changjiang Wei and Xinglai Ren. Modeling user-system interaction in use cases with dynamic views. In *Computational Intelligence and Industrial Application, 2008. PACIIA'08. Pacific-Asia Workshop on*, volume 1, pages 446–450. IEEE, 2008.

[27] Debasish Kundu, Debasis Samanta, and Rajib Mall. An approach to convert xmi representation of uml 2. x interaction diagram into control flow graph. *ISRN Software Engineering*, 2012, 2012.

[28] DPUV Nguyen and Tim Menzies WVU. Studies of confidence in software cost estimation research based on the criterions mmre and pred. 2009.