International Journal of Computer Science Trends and Technology (IJCST) – Volume 10 Issue 6, Nov-Dec 2022

RESEARCH ARTICLE OPEN ACCESS

Estimation Of Uml Use Case Point Tool Against Software Architecture

Arpana Bharani [1], Sweta Singh Patel [2]

[1] Dept. Of Computer Science, Dr. A.P.J Abdul Kalam University Indore, Madhya Pradesh
[2] Researchscholar At Dr. A.P.J Abdul Kalam University Indore - Madhya Pradesh 452016

ABSTRACT
Use case point (UCP) is calculated from use case model [G. Schneider and J. P 2001]. The use case model mainly consists of two documents, system or subsystem documents and use case documents that include the following items: system name, risk factors, system-level use case diagram, architecture diagram, subsystem descriptions, use case name, brief description, context diagram, preconditions, flow of events, post conditions, subordinate use case diagrams, subordinate use cases, activity diagram, view of participating classes, sequence diagrams, user interface, business rules, special requirements and other artifacts. Here, we explain the main items used to calculate UCP.
Keywords: - UCP,

I. INTRODUCTION

Use case point (UCP) is calculated from use case model [G. Schneider and J. P 2001]. The use case model mainly consists of two documents, system or subsystem documents and use case documents that include the following items: system name, risk factors, system-level use case diagram, architecture diagram, subsystem descriptions, use case name, brief description, context diagram, preconditions, flow of events, post conditions, subordinate use case diagrams, subordinate use cases, activity diagram, view of participating classes, sequence diagrams, user interface, business rules, special requirements and other artifacts. Here, we explain the main items used to calculate UCP.
They are system-level use case diagram and flow of events. System-level use case diagram includes one or more use Case diagrams showing all the use cases and actors in the system. Flow of events includes a section for the normal path and each alternative path in each use case.

It is this idea that gave birth to the creation of Use Case Point (UCP) metrics, originally developed by Gustav Karner[4]. In this method, Gustav Karner attempted to estimate the project size by assigning points to use cases, like in the same way, FPA assigns points to functions [G. Karner, 1993]. Use Case Point is simple, and it has more accuracy than lines of code or DELPHI (expert experience method), it avoids quite a difference results in the same project caused by different estimating personnel [Q. Yu et al 2011]

The use case is a notional description of a system, frequently used at the earliest stages in a project. The use case is one type of graphically oriented notation in the Unified Modeling Language (UML), a family of notational methods used to describe various aspects of software and its underlying structures. Use cases have three descriptive characteristics, which can be exploited to provide sizing information [J. Kammelar 2000].

Cost models like COCOMO and sizing methods like Function Point Analysis (FPA) are well known and are widely used parametric estimation models in software industry and academia. The limitation of Function Point Analysis method is that experiences people are required for counting the function points. In order to minimize the limitations of FPA method, Gustav [Karner, G 1993] developed a method for sizing and estimating projects developed with the object-oriented method and named it as “use case point” method for estimation software efforts in the early stages of software project development. An early estimate of effort based on use cases can be made when there is some understanding of the problem domain, system size and architecture at the stage at which the estimate is made. The use case points method is a software sizing and estimation method based on use case counts called use case points.

II. PROCESS OF USE CASE POINT METHOD

This section briefly explains the measurement steps in the use case point method described in.

2.1. Use case model

Use case point (UCP) is calculated from use case model. The use case model mainly consists of two documents, system or subsystem documents and use case documents that include the following items: system name, risk factors, system-level use case diagram, architecture diagram, subsystem descriptions, use case name, brief description, context diagram, preconditions, flow of events, post conditions, subordinate use case diagrams, subordinate use cases, activity diagram, view of participating classes, sequence diagrams, user interface, business rules, special requirements and other artifacts. Here, we explain the main items used to calculate UCP [G. Schneider and J. P.2001].

They are system-level use case diagram and flow of events. System-level use case diagram includes one or more use Case diagrams showing all the use cases and actors in the system. Flow of events includes a section for the normal path and each alternative path in each use case. Figure 1 shows an example of system-level use case diagram of "Order Processing System". Figure 2 shows a part of flow of events of the use case "Place order" in Figure 1.

2.2. Counting use case point

Intuitively, UCP is measured by counting the number of actors and transactions included in the flow of events with Some weight. A transaction is an event that occurs between an actor and the target system, the event being performed entirely or not at all. Effort estimation based on UCP method is conducted through the following Steps 1 - 6:

Step1- (Counting actors' weight): The actors in the use case model are categorized as simple, average or complex. A simple actor represents another system with a defined API. An average actor is either another system that interacts through a protocol such as TCP/IP or it is a person interacting through a text-based interface (such as an old ASCII terminal). A complex actor is a person interacting through a GUI interface. Then, the number of each actor type that the target software includes is calculated and then each number is multiplied by a weighting factor shown in Table

1. Finally, actor's weight is calculated by adding these values together.

Step2 - (Counting use cases weight): Each use case is categorized as simple, average or complex. The basis of this decision is the number of transaction in a use case, including alternative paths. For this purpose, a transaction is defined to be an atomic set of activities, which is either performed entirely or not at all. But, included or extending use cases are not considered. A simple Use case has 3 or fewer transactions, an average use case has 4 to 7 transactions, and a complex use case more than 7 transactions then, the number of each use case type is counted in the target software and then each number is multiplied by a weighting factor shown in Table 2. Finally use case weight is calculated by adding these values together.

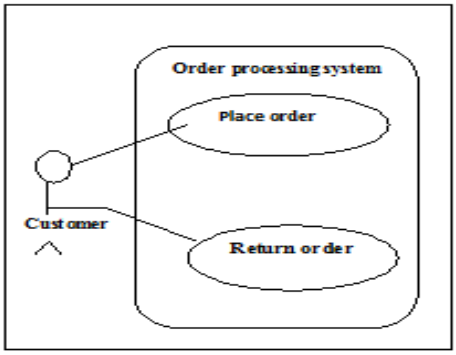

Figure 1. Use case diagram

Step 3 (Calculating UUCP): Unadjusted use case points (UUCP) is calculated by adding the total weight for actors to the total for use cases.

Step 4 (Weighting Technical factors and Environmental factors): The UUCP are adjusted based on the values assigned to a number of technical and environmental factors shown in Tables 3 and 4. Each factor is assigned a value between 0 and 5 depending on its assumed influence on the project. A rating of 0 means the factor is irrelevant for this project and 5 means it is essential. The technical factor (TCF) is calculated by multiplying the value of each factor (T1-T13) in Table 3 by its weight and then adding all these numbers to get the sum called the TFactor. Finally, the following formula is applied: TCF=0.6+(0.01*TFactor). The environmental factor (EF) is calculated accordingly by multiplying the value of each factor (F1-F8) in Table 4 by its weight and adding all the products to get sum called the Efactor. Finally, the following formula is applied: EF=1.4+(-0.03*EFactor)

1. The customer presses a button to select "Place Order".
2. The system supplies an input screen.
3. The customer enters product codes for product to be ordered.
4. The system supplies the product description and price.

Step5 (Calculating UCP): The adjusted use case points (UCP) are calculated as follows.UCP=UUCP*TCF*EF.

Table1. Actor Weighting Factors

Type	Description	Factor
Simple	Program interface	1
Average	Interactive ,or protocol, driven interface, interface	2
Complex	Graphical interface	3

Step6 (Estimating Effort): By multiplying the specific value (man-hours) by the UCP, estimated effort can be obtained. In, a factor of 20 man-hours per UCP for a project is suggested and Pedross reported that the use of the use case point method is accepted to estimate the size [M. Arnold, 1998]. They also described that since the language concepts for documentation are not well understood, it would be important to define the language concepts more precisely in advance. Anda et al. applied use case point method to three kinds of software project [B. Anda et al 2001]. The results showed that the estimated effort for each project was quite similar between use case point method and the specialist. They suggested that use case point method should use with other estimation method (e.g. function point, COCOMO). Also, for the novice manager, use case point method is easy to use in the estimation.

Table 2-Transaction-Based Weighting Factors

Type	Description	Factor
Simple	3 or fewer transactions	5
Average	4 to 7 transactions	10
Complex	More than 7 transactions	15

Table3. Technical Factors for System and Weight

Factors	Description	Weight
T1	Distributed system	2
T2	Response or throughout performance objective	1
T3	End user efficiency (online)	1
T4	Complex internal processing	1
T5	Code must be reusable	1
T6	Easy to install	0.5
T7	Easy to use	0.5
T8	Portable	2
T9	Easy to change	1
T10	Concurrent	1
T11	Includes special security features	1
T12	Provides direct access for third parties	1
T13	Special user training facilities are required	1

Table 4. Environmental Factors for Team and Weight

Factors	Description	Weight
F1	Familiar with the Rational unified Process	1.5
F2	Application experience	0.5
F3	Object-Oriented Experience	1
F4	Lead analyst capability	0.5
F5	Motivational	1
F6	Stable requirements	2
F7	Port time workers	-1
F8	Difficult programming language	-1

III. UML OBJECT ORIENTED

In object-oriented software production, use cases describe functional requirements. The use case model may, therefore, be used to predict the size of the future software system at an early development stage. This paper describes a simple approach for software cost estimation based on use case models: the 'Use Case Points Method'.
Use cases provided a high-level description of the intended function of the system. A small application might have only one use case, while very large applications may have hundreds. The numbers of scenarios are the potential outcomes of the software. There is no limit on the number of scenarios that a particular use case may have. Actors are the “agents” that interact with the software and so, use cases must have at least one actor. A commonly accepted standard is one actor per use case. In size estimating with use case point, various non-functional requirements are an important role. They are portability, performance, maintainability, security, easy to change and so on, those are not written as a use case. Cost and effort estimation is an important aspect of the management of software development projects. Experience shows that accurate estimation is difficult. Most methods for estimating effort require an estimate of the size of the software. The basic formula for converting all of this into a single measure, use case points, is that we will “weight” the complexity of the use cases and actors and adjust their combined weight to

reflect the influence of the nonfunctional and environmental factors. Then use case point can be used to calculate the effort of the project [Schneider 1998].

IV. SOFTWARE DEVELOPMENT USE CASE POINT MEASUREMENT TOOL

In order to effectively introduce use case point method to the actual software development, we decided to develop a use case point measurement tool. As described in Section 2.3, there exist several tools. But, it is necessary to judge the complexity of actors and use cases by manually. The judgment is the most important activity to count use case point and should be automated. To effective introduction of use case point, at first, we develop an automatic use ase point measurement tool as possible. Especially, it is necessary to develop a way of decide the weight for each actor and use case in the use case model of the target software system. To attain it, we propose several rules to classify the weight for actor and use case in Section 3.2 and 3.3 Also, it is necessary to write the use case model in Machine-readable format. So, we assume that the use case model is written in XMI (XML Metadata Interchange). Because most CASE tools for writing UML diagrams support to export the them as XMI files. Fortunately, Hitachi Systems & Services is going to use UML design tool that exports the use case model as XMI files.

3.2. Rules for weighting actors

As described in Section 2.2, weight for each actor is determined by the interface between the actor and the target System. But, the description of actor described in use case model does not include information of the interface. That is, we can get only the name of actor. So, we propose the following three steps to classify the complexity of actor.

Step1: Classification based on actor's name: Generally, actor is a person or an external system. According to Table 1, in case that the actor is a person, the weight can be average or complex. Also, in case that the actor is a external system, the weight can be simple or average. So, at first, based on the name of the actor, we judge whether the actor is a person or an external system. That is, beforehand, we prepare the list of keywords which can be included in the name of software system. For example, the keywords "system" and "server" are used in the system's name In our tool, we set the following keywords for an external system through the discussions with the engineers in Hitachi Systems & Services .Keywords for Step1 (KL a): system, server, application, tool.

Step2: Classification based on keywords included in use case: Here, we focus on the flow of events included in use case to which the actor is relevant. At first, we prepare three kinds of keywords list for each complexity of actor. For example, keyword list for complex actor includes "GUI", "button", and so on. Then, we extract all words included in the flow of events and then match them with each keyword in the lists. Finally, the actor's Weight is assigned as the complexity for the keyword list that is most fitted to the words in the flow of events. In our tool, we set the following keywords for each complexity through the discussions with the engineers in Hitachi Systems & Services.

Keywords for Simple actor (KL sa): request, send, inform

Keywords for Average actor(system) (KL aas): message, mail, send

Keywords for Average actor(person) (Kl aap): command, text, input, CUI

Keywords for Complex actor (KL ca): enter, button, press, push, select, show, GUI, window

Step3: Classification based on experience data: In case that we cannot determine the actor's weight at Step2**,** we determine it based on the experience data. The experience data includes the information about the use case model and the use case point developed in the past software projects. If there exits several actors whose names are the same as the target actor, then we decide the weight whose value commands an absolute majority. By using Figures 1 and 2, we show a simple example of classification of actor. In Figure 1, there is one actor named "Customer". In Step 1, since no keywords in KL a is included in the name of the actor, the actor "Customer" is classified as a person. In Step2, events 1 and 3 are extracted because "Customer" is related to them. Then, as the result of matching the Keywords of KL aap and KL ca with the words in the events, the keywords ("press", "button", "enter") in KL ca are more included in the events. So, the complexity of the actor "Customer" is judged as "Complex".

3.3. Rules for weighting use cases

As described in Section 2.2, the complexity of use case is determined by the number of transaction. So, we focus on The flow of events in the use case model. Intuitively speaking, the simplest way to count the transaction is to count the number of event. But, since there are no standard to write the flow of events, the developer can write the description freely using natural language. It is quite possible that several transactions are described in one event.
On the other hand, several guidelines to write events in use case model have been proposed [Alistair Cockburn 2000]. There are ten guidelines to write a successful scenario (flow of events). Among them, we focus on the following two guidelines.
(G1) Use simple grammar: The sentence structure should be absurdly simple. That is, it is easily understand what is the subject, verb, direct object and prepositional phrase. (G2) Include a reasonable set of actions: Jacobson has described a step in a use case as representing a transaction. He suggests the following four pieces of a compound Interactions should be described.
(1)The primary actor sends request and data to the system,
(2)The system validates the request and the data,
(3)The system alters its internal state and
(4) The system responds to the actor with the result. So, based on the above guidelines, we propose the way to analyze the events using the morphological analysis and syntactic analysis. Through these analyses, we can get the Information of morpheme from the statement and dependency relation between words in the statement. We conduct the morphological analysis for all events (statements) and get the information of the subject word and predicate word from each event (statement). Then, we apply the following rules: • Rule U-1: We regard each set of the subject and predicate word as a candidate of a transaction. • Rule U-2: Among the candidates, we identify the one that related to actor's operation and system response as a transaction. For each use case, we conduct the above processing and then get the number of transactions. Then, based on the number of transaction, we judge the complexity of each use case. In case that there is no flow of events in a use case, we determine the complexity based on the experience data. The experience data includes the information about the use case model and the use case point developed in the past software projects. If there exits several use cases whose name are the same as the target use case, then we decide the weight whose value commands an absolute majority.

3.4. Implementation

Based on the proposed method, we have implemented a prototype tool called U-EST(Use case based Estimation Supporting Tool). The input is a XMI file. The U-EST is implemented in Java and Xerces2 Java Parser is used to analyze the model file. Since the U-EST is mainly used by Japanese engineers, it has to deal with the Japanese description. In order to conduct morphological analysis and syntactic analysis for event written in Japanese in the use case, we adopt a tool called CaboCha . CaboCha is the most famous and precise syntactic analyzer for Japanese. Figure 3 shows an architecture of U-EST. Here, we explain the processing of UCP counting based on the U-EST. At first, the user (designer) writes use case models and saves it as XMI files. Then, XMI analyzer automatically extracts actors and use cases from the input file (use case model). Then, Complexity analyzer judges the complexity of them and calculates UUCP. Here, the U-EST shows the list of actors and use cases with their complexity by the request of the user. With respect to the use case, the U-EST shows the list of events, sets of the subject and the predicate word (candidates of transaction) in the use case and the sets that are identified as transactions. If necessary, the user can modify the classification results and recalculate UUCP. Then, by setting the technical and environmental factors, UCP calculator outputs the results and the results are stored in the Experience database. The effort is calculated by multiplying the specific value (man-hours) by the UCP. Currently, the value is set as 20 man-hours per UCP shown in [10]. But, the value can be modified through the GUI.

V. IMPORTANCE OF USE CASE POINT TOOL

Software sizing is a critical activity for planning and monitoring software project development focusing on time and budget [C. Jones 2007]. Today's software applications are very heterogeneous in nature and use different technologies, tools, and several programming languages [Albrecht, 1979]. It became very difficult to understand and assess the software products. Software size is extremely important in determining and controlling costs, schedules, quality and productivity in software project management.

Software applications grow and become more complex. Therefore a common method needs to be used and established in the industry to understand measure and communicate size and productivity. Software sizing is one of the challenging, and critical activities in the software development process. Effective software sizing is necessary for successful completion accordance with the budget and time. In practice, it is tedious process for software professionals to measure the software size methodically.

VI. CONCLUSION

This paper proposed an automatic use case point tool, the U-EST. The U-EST calculates use case point from use case models written in XMI files. We have also applied the U-EST to five use case models developed in the actual software projects. As the results, the UCP calculated by the UEST are considerably adequate. We are going to introduce the effort estimation based on UCP method to the company. In order to show the usefulness of the U-EST, we will apply it to many software development projects. Also, we are going to analyze the relationship among UCP, function point and actual software development effort and evaluate the usefulness and applicability of the estimation by UCP method.

REFERENCES

[1] G. Karner, "Metrics for Objectory", Diploma thesis,University of Linköping, Sweden. No. LiTHIDA- Ex- 9344:21. December 1993.

[2] Q. Yu et al., " Application of Estimating Based on Use Cases in Software Industry ", 7th International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM), 2011.

[3] Karner, G., (1993) "Metrics for Objectory", Diploma thesis, University of Linkoping, Sweden. No. LiTHIDA- Ex- 9344:21.December 1993

[4] G. Schneider and J. P. Winters: "Applying Use Cases, Second Edition", Addison Wesley (2001).

[5] J. Kammelar, "A Sizing Approach for OO-environments". In forth International ECOOP Workshop on
Quantitative Approaches in Object-Oriented Software Engineering.

[6] B. Anda, H. Dreiem, D.I.K. Sjoberg, M. Jorgensen: "Estimating Software Development Effort based on Use Cases
- Experiences from Industry", Proc. of Fourth International Conference on the UML, pp. 487-504(2001).

[7] M. Arnold, P. Pedross: "Software Size Measurement and Productivity Rating in a Large-Scale Software Development
Department", *Proc. of the 20th ICSE*, pp. 490-493(1998).

[8] G. Schneider and J. P. Winters: "Applying Use Cases, Second Edition", Addison Wesley (2001).

[9] Schneider and Winters, "Applying use Cases". Addison- Wesley, 1998.

[10]. Albrecht, A.J. Measuring application development productivity. In: SHARE/GUIDE: Proceedings of the IBM Applications Development Symposium, (October 1979) 83-92.

[11]. C. Jones, *Software Estimating Rules of Thumb*, 2007.

[12] Alistair Cockburn: Writing Effective Use Cases (Agile Software Development Series), Addison-Wesley (2000).

[13] Object Management Group (OMG), "XML Metadata Interchange (XMI) Specifi cation Version 2.0", (2003).

[14] Apache, http://xml.apache.org/.

[15] CaboCha : Yet Another Japanese Dependency Structure Analyzer, http://cl.aist-nara.ac.jp/ taku-ku/software/cabocha/.

[16] http://www.embarcadero.com/products/describe/index.html