UML Based Effort Estimation in Component Based Systems

Nidhi Arora1, Hitesh Singh2, Sarthak Babbar3, Sumit Bansal4 and Taruna Kalra5

1,2,3,4,5 HMR Institute of Technology and Management, Guru Gobind Singh Indraprastha University.
1 nidhiarora.gimt@gmail.com, 2 Hitesh.singh.85@gmail.com, 3 Sarthak.babbar@live.com, 4 websumit.bansal@gmail.com, 5 taruna8676@gmail.com

ABSTRACT
Software development has come a long way from traditional software development, which is characterized by the structured programming paradigm introduced in the late 60’s and early 70’s to contemporary development practices, which characterize a software application as interacting, independent components that captures the monolithic view of software development. In these models, software effort is projected at the large-grained system level. Thus these traditional measures do not take into account the amount and complexity of elaboration required, concentrating instead on the amount of data accessed or moved, whereas to accurately predict effort in Component Based Software Development (CBSD) a fine grained approach is needed hence, UML is used for modeling these systems. Its various estimation techniques expedite the estimation of resources like efforts, cost, manpower etc. This paper proposes a comparison between the different techniques used to calculate some new sub-characteristics namely, complexity, & efforts. In this context, Karner’s method, kim et al, Fast && Serious and other techniques were adapted to derive functional size measures, complexity measures such as technological and environmental factors, from UML models. These complexity factors calibrated for producing relatively accurate effort estimates efficient for CBS.

KEYWORDS
Class Diagrams, Class Points, Component Based Software Development, Environmental Factors,

1. INTRODUCTION
CBSD is principally based on the concept of object orientation techniques. Hence, the UML effort estimation techniques found perfect for the projects using the CBS approach. The accurate effort estimation were done using the various constructs of UML mainly, the Use Case Model and the Class diagrams but, later it found that other constructs such as Sequence diagram, collaboration diagram were flawless for the effort estimation.

2. COMPONENT BASED SOFTWARE DEVELOPMENT (CBSD)
“A component is a nontrivial, nearly independent, and replaceable part of a system that fulfills a clear function in the context of a well-defined architecture.” [4, 6] Also a component is a modular, deployable, and replaceable part of a system that encapsulates implementation and exposes a set of interfaces.

Hence, Component-based approaches for the development of information systems are a widely growing engineering discipline that deals with the cycle of developing components and developing with components. As, CBSD is a complex technology thus requires a more refined approach for estimation. Here, the traditional approaches don’t work out. Therefore, UML based effort estimation techniques are used [10]. Component based development can be represented as:

3. UNIFIED MODELING LANGUAGE
UML is a graphical notation for object oriented analysis and design. UML provides a framework for describing a set of models that captures the functional and structural semantics of any complex information system [17]. UML comprises of many constructs such as use case diagrams, interaction diagrams, class diagrams and others to incorporate the effort base analysis. As, UML is a universal language therefore, its measurements have been easily incorporated into Component Based Systems.

4. GOALS OF UML
The UML was invented primarily to address the challenges faced in the design and architecture of complex systems [19]. The basic objectives or goals behind UML modeling are:
- Easy to use and visual modeling language for modeling a system's structure.
- Providing extensibility.
- Language and platform independent so that it can be used for modeling a system irrespective of the language and platform in which the system is designed and implemented.
• Incorporate the best possible practices as per the industry standards.
• Provide support for Object Orientation, design and apply frameworks and patterns.
• UML models respond well to an incremental development environment. It is possible to develop only those parts of the model that are required to satisfy the new requirements, and the code needed to fulfill those requirements is in place.
• The use case driven nature of modeling with UML ensures that all levels of model trace back to elements of the original functional requirements.

5. UML FOR ESTIMATION
UML is a widely recognized standard to describe software systems using object oriented concepts through visualization. It provides a well-structured architecture of a system through various diagrams representing different viewpoints of the target system. There are many approaches available that exploit knowledge from use case diagrams and class diagrams in the process of predicting efforts for a system using component based development. Their main focuses for estimations were use cases.

6. USE CASE
It captures system functionality as seen by users. Use case show user’s perspective of the system. It is built in early stages of development, use case specify the context of a system by capturing the requirements of a system [11]. Here the use case represents the functionality provided to the actors through which they perform the desired sequence of interactions. On the other hand, the actor represents the roles, the actors are actually responsible for initiating the use cases as per the interactions. Both actors and the use cases are comprised in a rectangular box named as a system. Now it clearly shows that the interaction of the actors and the use cases completes the system. It validate system’s architecture, drive implementation and ease of effort estimation. The sample use case has been shown in the figure as:

This construct provides easy analysis of the system with respect to its actors as well as the functionalities provided to the actors [7]. By viewing the actors and the use cases in context of their weights from the use case diagrams the system efforts were given in context of use points [3] therefore system; viewed at the fine grained level.

7. CLASS DIAGRAMS
This diagram represents the various components in form of classes with their attributes and operations. Also the classes are the core concept of the object oriented approach hence their role is unbeatable in the various estimation techniques. Apart, from the Use Case diagrams the class diagram also helps in the effort estimation. The way in which use cases formulate the use points in a similar fashion the class diagrams formulates the class points [1]. Most of the techniques proposed in this paper are based on the use case points and class points. The class diagram is shown in the figure as:

Fig. 3: Class Diagram

Now, few of the UML based effort estimation techniques incorporated into the paper for component based development are represented with their set of equations and estimations.

8. KARNER’S USE CASE POINT
As, use case diagram carry the main information for developing a system hence, this estimation technique was purely based on the concept of use case points and an extension to traditional functional point approach.
In this process, first of all actors are characterized as simple, average or complex by constructing a Use case and the total unadjusted actor weight (UAW) calculated by counting the number of actors in each category, multiplying each total by its specified weighting factor, and then adding the points. Next, categorization of the use cases is done as simple, average or complex, depending on the number of transactions, including the transactions in alternative flows.
Then the unadjusted use case weights (UUCW) are calculated by counting the number of use cases in each category, multiplying each category of use case with its weight and
adding the products. The UAW is added to the UUCW to get the unadjusted use case points (UUPC) [12].

Next, the use case points are adjusted based on the values assigned to a number of technical factors and environmental factors. Each factor is assigned a value between 0 and 5 depending on its assumed influence on the overall project. This step has 3 different formulæ:
1. The Technical Complexity Factor:
   \[ TCF = 0.6 + (0.01 \times Tfactor) \]
2. The Environmental Factor:
   \[ EF = 1.4 + (-0.03 \times Efactor) \]
3. Adjusted use case points:
   \[ UCP = UUCP \times TCF \times EF \]

Finally, the UCP is multiplied by a historically collected data representing productivity, such as a factor of 20- staff hours per use case point, to arrive at a project estimate. The result provided estimates of the total number of person hour which interact with the system in order to complete the project.

This was the earliest work based on the use case points which described effort considering the technical and the environmental factors. All other work in the Use Case points is an extension to the karner’s proposed research. Although, it is not a very fine approach but still is the base of the further development process concerned with estimation of efforts in component based systems. This approach is completely based on the use case points but later on it discovered that efforts can also be estimated using class points. The diagrammatic representation is as follows:

![Diagram](image)

**Fig. 4:** Karner’s approach

### 9. KIM ET AL. USE CASE POINT

Kim et al. [13] proposed a new approach that is easier to calculate, excludes the expert’s decision, and focus more on the use case itself. They used UML points to estimate project effort and size measurement. UML points, calculated by adding use case points and class points. According to their work use case diagram has much information about the early development stage’s concept and the target system’s dynamic viewpoint. The developer uses these diagrams for communicating with the customer to decrease the conceptual gap between them, which has sufficient knowledge of the target system. The formulæ used are:

**Estimation Using Use case Points (UCP)** – This definition represents the use case points of the target system.

\[ UCP = \sum (NOA+NOUC+NOR) \]

Where, NOA = Number of actors, NOUC = Number of use cases, and NOR = Number of roles.

If the value of NOA is high then the system will have a great deal of interaction with its environment. UCP, calculated by adding up all the use case points. Next, the Class points calculated for giving the final estimated effort. The class points defined as follows:

**Final Estimated Effort**

\[ CP=\sum (NOC+NOIR+NORR+ NOM+NOCA+NOASS) \]

Where, NOC = Number of Classes, NOIR = Number of Inheritance Relationships, NOUR = Number of Use Relationships, NOR = Number of Realize Relationships, NOM = Number of Methods, NOCA = Number of Class Attributes and NOASS = Number of Associations.

This combination of Use points and Class points helped in better understanding of the complexity and the efforts of CBS.

**Fast&&Serious : Carbone & Santucci**

As, Object Oriented (OO) has three canonical dimensions namely, functionality (behavior of objects), amount of communication among objects, and percentage of reuse through inheritance [14]. Fast&&Serious, focuses on these dimensions combining several measures extracted by UML diagrams and associating each class with a size estimation, in terms of source lines of code (SLOC). The approach worked on the data about the project under analysis tool used for and produces an estimation of the software complexity in terms of SLOC.

The method starts with the analysis of the class diagram and on the basis of the diagram level of detail, applying a rough (Fast) or a detailed (Serious) estimation method. Numerical data computed for all the classes belonging to the class diagram, further assessed through additional pieces of information coming from other UML diagrams (use cases, sequence diagrams, and state diagrams). The process contained six major steps shown as figure in the subsequent section.

### 10. FAST& SERIOUS STEPS

1. Determined the method to be applied: Fast && Serious.
2. Calculated the complexity of each class using class points from the class diagram.
3. Assessed the class points using use case diagrams.
4. Assessed the class points by interaction diagrams.
5. Assessed the class points by state diagrams.
6. Sum the CPs computed in phase 2, 3, 4, and 5 for the number of SLOC of the whole system.

In the next section the detailed approach of all these steps has been elaborated.
Step 1: Fast or Serious?
In this step the class diagrams, analyzed and the following metrics are extracted:
DIT (Depth in Inheritance Tree)
MPC (Number of Methods per Class)
NAC (Number of Association per Class)
PMS (Percentage of methods with signature) Average of PMS predicted the method adapted in further steps. PMS greater than a threshold value (say 70%) use Fast method, otherwise Serious method needs to be applied. DIT, MPC, and NAC further used in next steps.

Step 2: Computing class complexity
In this step a complexity value for each class calculated with the help of few intermediate support results. For each class, the number of Class Points (CP), calculated as follows:
CP(c) = 2* SP(c) + 3* BP(c)
Where, CP(c) estimates the complexity of class c; SP(c) as State Points being the sum of weights of each list attributes in class c; and BP(c) as Behavioral Points calculated for each method in class c.
Note: If a class c has no attributes and methods, it is not included in the estimation process.

Step 3: Assessing class complexity using Use Case Diagrams
The step intended the assessment of the CP(c) computed in the previous step using the information extracted from use case diagrams. This were obtained by first associating a complexity value with each use case and each actor; then using CUCA(c) and CAA(c) to assess the complexity of c, and finally found the increment/decrement percentage for CP(c).
The proof supported by the example discussed next, example CUCA(c) = 8, CAA(c) = 15 gives the increment of 9% through Use Case assessed as by Carbone and Santucci. Hence, if CP(c) = 1000 the new value for CP(c):
CP(c) = 1000 + 1000*9/100.

Step 4: Exploiting Interaction Diagrams
Similar to step 3 where the strategy; assessed the CP of a class when new information got available, the further focus shifted over to exploit interaction diagrams; in particular sequence diagrams. The sequence diagram exploited as: For each sequence diagram sed j where j=1...nsed and nsed implies the number of available sequence diagrams, now looked at the instances of c in each sed j calculating the numMess(c, sed j) as the number of messages sent or received from instances of class c in sed j. Finally, let totMess(c) as the sum of numMess(c, sed j) and rsed(c) as the ratio between the number of sequence diagrams wherever c appears. Here, totMess(c) gives an idea of the amount of communication involving class c, while rsed(c) corresponds to the percentage of code they require to communicate with such a class. These values seemed useful to find the percentage of increment/decrement for CP(c) as stated by Carbone and Santucci approach.

Step 5: Exploiting State Diagrams
A State Diagram std remained directly associated with a class c or to a method of c thus, extracts the following values from each std: the number of states as numSta(std) and the number of actions (i.e., entry and exit actions associate with a state) as numAction(std), assessing CP(c) after computing the total amount of states totSta(c) and total number of actions totAction(c) for the class c and finding the percentage of increment/decrement as guided by Carbone and Santucci approach.

Step 6: Computing the whole system size
So far, CP(c) associated with each class in the class diagram (CD) assessed; CP(c) is correlated with the size of c and the whole system through the following formulae:
Size(c) = 4 + 10 * CP(c) ^ .7
Size(System) = ΣSize(c).
The number of FP’s associated with the System obtained using this approach.
Apart from these basic approaches other more refined approaches which were found useful for the research were addressed for effort estimation. They are discussed in the coming section.

11. OTHER UCP BASED APPROACHES
11.1. USE CASE TRANSACTIONS & ENTITY OBJECTS
Robiolo and Orocosco [9] work were analyzed as an alternative method to estimate effort through use case diagram. Their work primarily focused towards improving the magnitude of relative error in the effort estimation as well as the time at which the estimation of effort performed. It was done through statistical computations as first objective and use case diagrams for second.
In this approach, two alternative variables were used to express a notion of size: transactions [5] and entity objects and finally effort were estimated through mean productivity value. Evaluation was designed to check whether such proposal was useful to estimate effort in a real life environment.
Transactions and entity objects, calculated from the use cases textual descriptions whereas, function points are obtained at a later stage. There are following formulae for computing size of the application (Size Ap):
Size Ap. = Σ Number of Use Case Transactions.
Size Ap. = Σ Number of Module Entity Objects.
The improvement of effort estimation was achieved by using the size notion. But, these estimates can only be applied to projects carried out in terms of transaction. Moreover, it is possible to track a project because transactions and entity objects can be easily counted in each primary use case and that they may be used to measure the changes imposed by the client throughout the development process. Thus, productivity quotient between size and effort may be easily calculated as changes occur. Such productivity value may be compared to the productivity value used to make the original estimation in order to measure the deviation resulting from the modifications inherent to a real life development cycle.

11.2. DIEV USE CASE POINTS
In this approach based on UCP, Diev [8] reviewed several situations that occur when a use case model is used for estimation. The emphasis addressed was to provide accurate use case based estimate, it is necessary to take into consideration the existing system’s design as well as the details of the project. It extends the earliest work done by Karner by correlating it with system’s complexity and the supplementary efforts needed for various management activities. The work done primarily focuses on application of UCP in financial domain. According to his approach, UCPm i.e. modification to use case points is given as:

\[
\text{Size} = \text{UUCP} \times \text{TCF} \quad \text{and} \\
\text{Effort} = \text{size} \times \text{EF} \times \text{BSC} \times \text{R} + \text{supplEffort}
\]

Where UUCP is unadjusted use case points, TCF is technical complexity factor, EF is environmental factors (as given in Karner’s method ) and BSC is base system’s complexity, R is Person-Hour per use case point and SupplEffort is the supplementary efforts required to manage the system. This work was an extension of Karner’s method only, yet it demonstrated the issues like use case complexity of the base system and the supplementary efforts required for management activities like configuration management, project management.

11.3. ANDA ET AL. USE CASE POINTS

In this UCP approach, Anda et al. [2] investigated the application of use cases in estimating software development effort in a multiple case-study, in which four companies were chosen to actually develop a system based on the same requirement specifications. The teams from the four companies had very similar qualifications, and the functionality of the four resulting systems was almost equivalent. The teams followed different development processes and placed different emphasis on the quality of the code. Employing use case point method, necessary effort was estimated to 413 hours where the actual effort of the development teams lied between 431 to 934 person hours. In this work too, it was addressed that there is some adaptation required to handle the increase in effort due to development process and quality requirements on the code. The other issue emphasized was again the granularity of the use case. They also discussed it with the number of transactions in a use case. They had taken average complexity for TCF and EF and in some way sideline the need of these. However, the case studies in question were explicitly made out of the scope of component based development. Hence, it did not concern about the nature of technical and environmental factors of component based development.

11.4 Mohagheghi Use Case Points

Mohagheghi et al.[15] adapted and tested use case points method on a large industrial system with incremental changes in use cases. With main assumptions of use cases potentially being used as a measure of the size of a system, changes in these to be used as a measure of changes in functionality between releases. They implemented the UCP method in correlation with COCOMO method [16, 33] to predict the effort required to build a large and incremental real time system. Their method was independent of automated UCP method tools, paradigms or programming languages, and could promote high quality use cases. This method is cheap, transparent and easy to understand and also suitable when development is being outsourced. They proposed an overhead factor (OF) to be employed for complex systems. They observed the granularity of each use case with the number of transactions it contains, the effect of changes it incorporates in each of the iteration of development cycle, and computed the efforts required in the form of primary (new work) and secondary (modification) efforts. Again, the TCF and EF are arbitrarily taken as average or rather with no effect on size of the system. It also excluded the influence of non functional requirements but put forward to be included in the technical factors. Proposed overhead factor (OF) was used to estimate the total effort based on effort used to develop the system before system test and was empirically derived to be 2. This empirical observation relies on judgments made by researchers and is subjected to further adjustments. As suggested, future studies can help to understand the degree of modification in incremental development of a system (use cases, code and integration costs), and how the method works on other types of systems.

12. CONCLUSION

In this work, the established and popular methods of UML were used in component based system development. Unlike other development strategies, CBSD inherently incorporates functional and non functional requirements into the desired system. Thus, addressing of these issues with the help of newly defined technical factors and environmental factors suited the CBS effort estimation. Also, the use of use case points and the class points were found useful in the event of effort estimation. Thereby; the results from the study supported previous claims that UML based methods supports early estimation of software development effort for CBS. It addressed the specific concerns of CBSD in the context of early effort estimation and the encouraging and accurate results acknowledged the work.

13. FUTURE SCOPE

The future work in this context could be development of an automated tool for fetching information directly from UML models and to apply this framework on large industrial development. However, during this study we encountered few issues that are not covered under effort estimation but directly affects the overall cost of the system. Till date, research in UML based effort estimation demonstrates its interest in the base theory being extended or improved. People have reviewed the inline approaches used for effort estimation or defining new metrics. Thorough comparison of different approaches is as such nowhere present. There should be some normalized
comparison for these approaches so that future directions can be outlined.

REFERENCES


[17]. Smith J., “The estimation of effort based on use cases”, Rational Unified Process (RUP) white papers, 1999
