UCPNet: Enhancing the Use Case Points Method for Software Cost Estimation Using Petri Nets

Twinkle Rachel Thomas, Sathish Kumar B
School of Computing Science and Engineering, VIT University, Chennai, India

ABSTRACT

Cost Estimation is one of the most difficult tasks in the development cycle of a software. Since this takes place in the requirements phase itself, if proper care is not taken while estimating the cost then it will adversely affect the entire project. Various methods have been adopted in estimating the cost in the past but some or the other flaws have always been discovered to improve upon for the next enhancement. This paper also explains such an enhancement. Here Use Case Points (UCP) method which makes use of use cases has been taken into account for introducing the enhancement.

This paper proposes UCPNet where the concept of Petri Nets has been integrated into UCP in order to estimate cost with a higher accuracy compared to UCP. This has been illustrated using a case study.

Keywords: Cost Estimation; Use Case Points (UCP); Use Case; UCPNet; Petri Nets.

I. INTRODUCTION

Estimating the cost of software has become a very daunting and important task today. The difficulty in determining cost lies in the uncertainty in calculation while deducing the cost of software. It has become a necessity now to include estimating the cost of a software right in the beginning stages of the development cycle of software. There is a need to include cost estimation in the modelling stage of the software. Such care as to including cost estimation of software in the modelling stage arises due to danger as overrunning of the costs thereby affecting the entire requirements engineering.

Software cost estimation is a hard management activity due to the lack of information in making decisions in the early phases of the development, frequently characterized by uncertainty. Software which is an intangible product is hard to develop because there are so many unknowns in the development cycle. Even if we use a well-defined methodology, the development cost of such a well-defined application is not able to be predicted easily. Keeping into consideration that the software developed suffers from the cost overruns that occur frequently, software cost estimation has become a key open issue for the software industry.

The costs to consider include personnel costs, hardware costs, software costs, communication costs, training costs, infrastructure costs, etc. However some key factors which contribute to the difficulty in cost estimation include the precise set of functionalities to be implemented, the risks which occur during the development process, and the knowledge and experience of the development team. The most influential factor among these are the set of functionalities to be implemented. Often, software developers guess the size of a new software product.

Software cost estimation models developed in early years used the Line of Code (LOC) or Delivered Source Instruction (DSI) metrics. For example, COCOMO model [2] used the DSI
metrics while some others used LOC metrics. The disadvantages associated with these include: (i) the lack of precise definition of LOC or DSI, (ii) there is no reasonable methodology by which one can estimate upfront the number of source code lines or instructions when the product is delivered and (iii) these are programming language dependent i.e. these metrics were defined for procedural languages such as FORTRAN and COBOL. With the development of block structured languages such as Pascal, Algol, C etc., LOC or DSI was difficult to define precisely.

Albrecht came up with the Function Point (FP) metrics in 1979[1]. FP uses five parameters: number of inputs, number of outputs, number of inquiries, number of internal logical files and number of external logical files. FP uses number of interactions and the size of data to be used in the end product. FP eliminated the need for LOC or DSI, and was widely used as it was platform independent. However it does not seem to be applicable to software products that are developed using the object-oriented methodology. In particular, the internal and external logical files’ notion is somewhat harder to identify in the object-oriented paradigm.

Gustav Karner[3] came up with the notion of Use Case Point (UCP) which is somewhat similar to the concept of Function Point. UCP is based on use cases. The use case model is the front end model of the Unified Modelling Language (UML) [4]. UML, being the most commonly used notation to design and model object-oriented software products, the application of use cases for size estimation and hence cost estimation seems to fit perfectly. However, Karner’s method involves disadvantages such as: (i) it does not take into account some of the application domain details such as the relationships between actors, (ii) relationships between use cases, (iii) the uncertainty of the cost factors and (iv) the abrupt classification. The objective of this research is to extend the UCP model with the use of Petri Nets thereby increasing the accuracy of estimating the cost of a product. This paper enhances a method for cost estimation based on the Use Case Points method using Petri Nets (UCPNet).

II. RELATED WORKS

A. Cost Estimation

In early times software cost (SC) was estimated by counting the lines of code and multiplying it with the functions. This was considered to give a good number for what the source code is worth. The retail cost (RC) is calculated by dividing the software cost by 30 or such.

\[
SC = \text{LOC} \times \text{func} \\
RC = \frac{SC}{30}
\]

B. Cost Estimation Models

Cost estimation models are mathematical algorithms or parametric equations which are used for estimating the costs of a product or project. Approval has to be obtained for the results of the models in order to proceed further. The results are characterized into business plans, budgets, and other financial planning and tracking mechanisms. Typically models functions when parameters are given as input. The parameters describe the attributes of the product or project in question, and possibly physical resource requirements. The output which the model provides is various resources requirements in cost and time. Applications include: construction, software development, manufacturing, new product development.
Some of the common cost estimation models are:

- COCOMO
- COCOMO II
- Program Evaluation and Review Technique (PERT)
- Functional Point Analysis (FPA)
- Use Case Points (UCP)

C. Use Case Diagrams

A use case diagram represents the user's interaction with the system and depicts the specifications of a use case. A use case diagram portrays the different types of users of a system and the various ways that they interact with the system. This type of diagram is used along with the textual use case. This is also accompanied by other types of diagrams as well. A higher-level view of the system is provided by using use-case diagram. It has been said before that "Use case diagrams are the blueprints for your system". They provide a simplified graphical representation of what the system must actually do.

Use case diagrams can be a good communication tool for stakeholders due to their simplistic nature. This mimics the real world. It provides the stakeholder a view to understand how the system is going to be designed. A complete functional and technical view of the system can be provided by additional diagrams and documentation.

D. Use Case Points (UCP)

Use Case Points (UCP) is one of the techniques for estimating software size. Software design and development uses methods like modelling languages e.g. Unified Modelling Language (UML) and process like Rational Unified Process (RUP). UCP uses these mentioned concepts. From the term use case points itself we can deduce that use case points depends on use cases. Thus the elements of these use cases are considered in order to calculate the software size. The estimated effort for a project can then be calculated by using the UCP for a project.

The method for size estimation to develop a system can be determined based on a calculation of the following elements:

- Unadjusted Use Case Weight (UUCW)
- Unadjusted Actor weight (UAW)
- Technical Complexity Factor (TCF)
- Environmental Complexity Factor (ECF)

The final size estimate can be calculated once the above four elements have been calculated. This final outcome so obtained is known as Use Case Points or UCP for a software development project.

Unadjusted Use Case Weight (UUCW)

The type of a use case is characterized based on the number of transactions they make. The total number of use cases that belong to the same classification are multiplied with their corresponding weights. Each value obtained is then added in order to produce Unadjusted Use Case Weight.
Table I. Classification of Use Cases

<table>
<thead>
<tr>
<th>Use Case Classification</th>
<th>No. of Transations</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td>1 to 3</td>
<td>5</td>
</tr>
<tr>
<td>Average</td>
<td>4 to 7</td>
<td>10</td>
</tr>
<tr>
<td>Complex</td>
<td>8 or more</td>
<td>5</td>
</tr>
</tbody>
</table>

**Unadjusted Actor Weights (UAW)**

The type of an actor is characterized based on the actor classification. The total number of actors that belong to the same classification are multiplied with their corresponding weights. Each value obtained is then added in order to produce Unadjusted Actor Weight.

Table II. Classification of Actors

<table>
<thead>
<tr>
<th>Actor Classification</th>
<th>Type of Actor</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td>External system that must interact with the system using a well-defined API</td>
<td>1</td>
</tr>
<tr>
<td>Average</td>
<td>External system that must interact with the system using standard communication protocols (e.g. TCP/IP, FTP, HTTP, Database)</td>
<td>2</td>
</tr>
<tr>
<td>Complex</td>
<td>Human actor using a GUI application interface</td>
<td>3</td>
</tr>
</tbody>
</table>

**Technical Complexity Factor (TCF)**

Each of the technical factors as mentioned in the table is given a rank between zero to five based on their implication level. This is then multiplied with their already existing weighted value to form a value which is then divided by hundred and finally added with 0.6 to give the Technical Complexity Factor.

Table III. Classification Based On Technical Factors

<table>
<thead>
<tr>
<th>Factor</th>
<th>Description</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>T1</td>
<td>Distributed System</td>
<td>2</td>
</tr>
<tr>
<td>T2</td>
<td>Response Time / Performance Objectives</td>
<td>1</td>
</tr>
<tr>
<td>T3</td>
<td>End-user efficiency</td>
<td>1</td>
</tr>
<tr>
<td>T4</td>
<td>Internal Processing Complexity</td>
<td>1</td>
</tr>
<tr>
<td>T5</td>
<td>Code Reusability</td>
<td>1</td>
</tr>
<tr>
<td>T6</td>
<td>Easy to Install</td>
<td>0.5</td>
</tr>
<tr>
<td>T7</td>
<td>Easy to Use</td>
<td>0.5</td>
</tr>
<tr>
<td>T8</td>
<td>Portability to other platforms</td>
<td>2</td>
</tr>
<tr>
<td>T9</td>
<td>System maintenance</td>
<td>1</td>
</tr>
<tr>
<td>T10</td>
<td>Concurrent / Parallel Processing</td>
<td>1</td>
</tr>
<tr>
<td>T11</td>
<td>Security features</td>
<td>1</td>
</tr>
<tr>
<td>T12</td>
<td>Access for third parties</td>
<td>1</td>
</tr>
<tr>
<td>T13</td>
<td>End-user training</td>
<td>1</td>
</tr>
</tbody>
</table>
Environment Complexity Factor (ECF)

Each of environmental factors as mentioned in the table is given a rank between zero to five based on their implication level. This is then multiplied with their already existing weighted value to form a value which is then multiplied with \(-0.03\) and finally added with 1.4 to give the Environment Complexity Factor.

<table>
<thead>
<tr>
<th>Factor</th>
<th>Description</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>E1</td>
<td>Familiarity with development process used</td>
<td>1.5</td>
</tr>
<tr>
<td>E2</td>
<td>Application experience</td>
<td>0.5</td>
</tr>
<tr>
<td>E3</td>
<td>Object-oriented experience of team</td>
<td>1</td>
</tr>
<tr>
<td>E4</td>
<td>Lead analyst capability</td>
<td>0.5</td>
</tr>
<tr>
<td>E5</td>
<td>Motivation of the team</td>
<td>1</td>
</tr>
<tr>
<td>E6</td>
<td>Stability of requirements</td>
<td>2</td>
</tr>
<tr>
<td>E7</td>
<td>Part-time staff</td>
<td>-1</td>
</tr>
<tr>
<td>E8</td>
<td>Difficult programming language</td>
<td>-1</td>
</tr>
</tbody>
</table>

Use Case Points (UCP)

Once the above factors have been calculated UCP can then be determined. The unadjusted use case and actor weights are added and then multiplied with the technical and environment factors to give the ultimate Use Case Point value.

III. PETRI NETS

Petri Nets offer a graph like notation for processes that follow an iterative manner. These processes are either iterative or concurrent and carried out in a step wise manner. However Petri Nets are not like UML diagrams. Petri Nets are purely mathematical and have precise mathematical equation for each of the steps in the procedure. The analysis and design of Petri Nets are very accurate and reliable as they are derived mathematically.

A. Description

A Petri Net diagram is in the form of a graph which consists of nodes that stand for the places and these being connected via the transitions. The state of a Petri Net is denoted by the one or more tokens that are present in these places. This is used to describe how a system changes from one state to another via transitions.

Petri Nets was discovered initially for describing processes such as chemical equations. A state comes into action when it contains at least one token ready for firing. A state is determined by the amount of tokens it possesses. The set of tokens are fired to another state through the transition. Thus the state that now contained the tokens loses its tokens and these tokens are added to the state to which it has been fired to.

B. Behaviour

Places that contain the tokens are the ones that are ready for firing. Once these are fired via the transition they reach the destination places on the other side of the transition.
According to the figure given below states one and three contain tokens. These are ready for firing. Once these are fired through the transition, they reach states two and four.

*Fig. 1. Petri Net diagram showing States and Transitions*

C. Enhancing Performance Measures

Petri Nets are used to enhance the performance measures of a system. It takes into account the response time, number of events processed or denied in some interval of time throughput, capacity, usage ratio, jitter, loss of information, latency, etc.

D. Example

\[2H_2 + O_2 \rightarrow H_2O\]

*Fig. 2. Before Firing*

The above diagram shows the Petri Nets diagram of the example of the given chemical equation before firing. Shown below is the Petri Net diagram of the same after firing.

*Fig. 3. After Firing*

IV. PROPOSED WORK - UCPNET

Here a method is proposed known as UCPNet wherein Petri nets have been integrated into UML Use Case diagrams to enhance the performance of the system and thereby estimating the cost more accurately. For illustrating the concept of UCPNet, a case study of a Bank ATM has been taken up.

*Fig. 4. Use Case Diagram for a Bank ATM*
Fig. 5. UCPNet for the Bank ATM

Using the Use Case Points method for the above Use Case diagram using the formulae described in the previous sections UCP is calculated as follows:

\[
UUCW = (7 \times 5) + (2 \times 10) + (0 \times 15) = 55
\]

\[
UAW = (2 \times 1) + (4 \times 2) + (3 \times 3) = 19
\]

\[
TF = 42
\]

\[
TCF = 1.02
\]

\[
EF = 10.5
\]

\[
ECF = 1.085
\]

Therefore, UCP = \((55 + 19) \times 1.02 \times 1.085 = 81.89\)

Unadjusted State Weight (USW)

The type of a state is characterized based on the number of states that follow after a transaction. The total number of states that belong to the same classification are multiplied with their corresponding weights. Each value obtained is then added in order to produce Unadjusted State Weight.

**Table V. Classification of States**

<table>
<thead>
<tr>
<th>State Classification</th>
<th>No. of States After a Transaction</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td>0 or 1</td>
<td>2</td>
</tr>
<tr>
<td>Average</td>
<td>2 to 5</td>
<td>4</td>
</tr>
<tr>
<td>Complex</td>
<td>6 or more</td>
<td>6</td>
</tr>
</tbody>
</table>

Unadjusted Transition Weight (UTW)

The type of a transition is characterized based on the transition’s impact. The total number of transitions that belong to the same classification are multiplied with their corresponding weights. Each value obtained is then added in order to produce Unadjusted Transition Weight.

**Table VI. Classification of Transitions**

<table>
<thead>
<tr>
<th>Transition Classification</th>
<th>Impact of Transition</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Light</td>
<td>Affects one state</td>
<td>5</td>
</tr>
<tr>
<td>Heavy</td>
<td>Affects more than one state</td>
<td>10</td>
</tr>
</tbody>
</table>

Use Case Points using Petri Nets (UCPNet)
Using the above equations UCPNet can be calculated by summing up USW and UTW.

UCPNet = USW + UTW

Using the above formulae the following calculations have been deduced for Figure 5.

USW = (5 X 2) + (6 X 4) + (0 X 6) = 34

UTW = (3 X 5) + (1 X 10) = 25

Therefore, UCPNet = 34 + 25 = 59

Hence, we have obtained a precise and lesser value as compared to UCP.

V. CONCLUSION AND FUTURE WORK

A more accurate method for cost estimation has been formulated. This has been obtained by using the concept used for calculating Use Case Points (UCP). Apart from considering the weights for each of the actors, use cases and technical and environmental factors alone UCPNet takes into account the weights of the states and transitions of the Petri Net, which is used to enhance performance and thereby providing a more accurate measure.

Future work involves adding more components in order to enhance UCPNet and considering the other aspects that include the non-functional requirements too while estimating the cost thereby obtaining an even more accurate figure.

REFERENCES


