UDC 65.01

Towards a Methodology to Estimate Cost of Object-Oriented Software Development Projects

Radoslav M. Rakovic

Energoprojekt-Entel Co.Ltd., Bulevar Mihaila Pupina 12,
11070 Belgrade, Serbia and Montenegro
rmrakovic@ep-entel.com

Abstract. Successful management of a software project, besides a well-defined project development process, requires an early estimate of project complexity. In a prevailing practice, software development costs usually have been determined *a posteriori* i.e. after software project implementation. It is essential, however, to know this estimate *a priori*, i.e., before commencement of works. This paper presents an attempt to construct a methodology that would enable an early estimate of software development cost and its refinements during subsequent development phases. The methodology assumes an object-oriented approach based on the Unified Modeling Language (UML) and Unified Software Development Process (USDP). It outlines an Use Case Driven, Architecture-Centric, Iterative and Incremental estimate process that could significantly improve and simplify early cost estimates. The presented methodology is illustrated on example of the POST software development project.

1. Introduction

Successful management of a software project, besides a well-defined project development process, requires an early estimate of project complexity. This would ensure adequate resource allocation to a project as a whole, as well as to each phase of development process. In most cases, the most important cost factor is labor. For this reason, estimation of software development effort is central to management and control of a software project. The main difficulty of such an estimate is the fact that software is a specific kind of product.

Cost and scheduling estimates provide highly valuable aid in a number of management decisions, budget and personnel allocations and in supporting reliable bids for contract competition. Managers feel more comfortable using estimate models than just relying on “rules of thumb” and entirely subjective judgments when planning budgetary and personnel resources for a new software project. Even though estimate models have some limitations that managers should be aware of, they may be viewed as valuable tools in the software engineering process. An estimate of software development effort has implications both for planning of a software project, and for its implementation. If this estimate is too low, then the software development team will be under considerable pressure to finish their job quickly, within allocated budget, and hence the resulting software may not be fully functional or tested. If this estimate is too high, then too much resource will be committed to this project and too much money will be spent unnecessarily.

In prevailing practice, software development costs usually have been determined *a posteriori* i.e. after software project implementation. It is essential, however, to know this estimate *a priori*, i.e. before commencement of works. In other words, if one wants to invest money, it is necessary to know estimate of related cost before making relevant decision.

The Importance of software development effort estimate has motivated considerable research in recent years. A brief survey of cost estimate models is given in Section 2 of the paper. The Section 3 briefly surveys foundations of the Unified Software Development Process, emphasizing the Use Case Driven, Architecture-Centric, Iterative and Incremental development processes paradigm. Also, details of the proposed methodology are described. The POST (Point-of-Sale Terminal) software development effort estimation is given as an example in the Section 4.

2. A Brief Survey of Cost Estimate Models

Many models have been proposed so far, for software development effort estimation. Table 2.1 shows a classification of estimate models, based on two criteria: (1) what complexity and size metrics are applied and (2) what effort and scheduling computation technique is used.

Table 2.1: Cost Estimate Models - classification

Effort & Scheduling Computation / Complexity &.Size Metrics	Parametric models	Non-parametric models (Machine Learning Approaches)
Source Lines of Code (KSLOCs)	❑ SLIM ❑ COCOMO	❑ Regression Trees
More complex elements (Dimensions)	❑ Function Points ❑ Object-Oriented Approaches	❑ Regression Trees ❑ Neural Networks ❑ Analogies

The simplest size and complexity metric, number of (thousands) source lines of code (KSLOC), is often and very successfully used. Very wide usage and large statistics for this metric makes it the most reliable one. Other more complex metrics, including object oriented approaches, are specific, insufficiently applied and confirmed, except to some extent Function Points.

Parametric models compute software development effort using formulas of fixed structure with parameters calibrated to fit historical data collected by measurments applied to already completed projects. A parametric model, besides software size and complexity metrics, may take into consideration the experience of the development team, the required reliability of software, the programming language and so on. In contrast, many non-parametric (machine learning) models make no or minimal assumptions about the structure of a model to study software development effort, but use learning algorithms to construct "rules" that fit historical data.

Putnam developed an early parametric model known as SLIM [15]. This model is based on an empirically confirmed assumption that a life-cycle effort varies with time and follows the Norden-Rayleigh distribution with some level of accuracy. The main result of this model is the well known "software equation" linking size of product in source statements (S_s) to effort (K) and development time (t_d) with state of technology (C_k) as a constant:

$$S_s = C_k K^{1/3} t_d^{\ 4/3} \qquad (2.1)$$

The COnstructive COst MOdel (COCOMO), based on regression analysis of 63 completed development projects, was developed by Boehm [3]. COCOMO relates the effort (E) required to develop a software project (in terms of person-months) to thousands of Delivered Source Instructions (KDSI) and the time of development (T_{DEV}) to effort, as follows:

$$E = a(KDSI)^b \qquad (2.2)$$

$$T_{DEV} = c(E)^d \qquad (2.3)$$

where a, b, c and d are parameters, which depend on the applied model. Prediction of the basic COCOMO can be modified using cost drivers which are classified into four groups relating to attributes of product, computer platform, personnel and project. These factors serve to adjust nominal effort up and down. There are several versions of the COCOMO ([4], [9],

[10]). The latest one, COCOMO II ([4],[10]) includes scale factors instead of the development modes of the basic COCOMO.

The Function Points (FP) model was developed by Albrecht [2]. Function points are based on characteristics of project at a higher descriptive level than SLOC. The Function point model measures functionality from user point of view, i.e., what the user requests and receives in return. Adjusted size (Adjusted Function Points – AFP) is given by equation 2.4:

$$AFP = UFP * TCF \qquad (2.4)$$

The Unadjusted Function Points (UFP) factor is based on five types of user functions (external inputs and outputs, logical internal files, external interface files and external inquires). The Technical Complexity Factor (TCF) is a result of weighted sum of 14 general system characteristics, depending on their degree of influence [2]. Over the years, various refinements have been made ([1], [22]), and several successive versions have been published under the co-ordination of IFPUG (International Function Point Users Group).

Machine Learning approaches to estimate software development effort [21] are derived from a more general methodology of artificial intelligence systems. This methodology requires historical data on which to apply learning strategies. There are several methods of machine learning but we will mention only two of them – Regression Trees and Neural Networks. The first one constructs decision trees for classifying data. Each project is described over some dimensions (a set of attributes). Predicting the development effort of a project requires that one descends the decision tree along an appropriate path and the leaf value at the end of that path gives an estimate of the development effort for a new project. The second one constructs an artificial neural network which consists of several processing elements. Each of these elements gives an output depending on its inputs and the whole network generates outputs by propagating initial inputs through successive layers of processing elements to final output layer. Some experiments have indicated that these techniques are comparable with traditional estimate methods although they are sensitive to the historical data which they are based on.

An alternative non-parametric approach is based upon the use of analogies [20]. The underlying principle is to characterize all projects with the same set of features and then to find the completed project most similar to the current one. Similarity is defined as Euclidean distance in a n-dimensional space where n is the number of project features. Each dimension is standardized so that all of them have equal weight. The known values of the most similar project are then used as basis for effort prediction for the current one.

A variety of software metrics have been proposed for object-oriented development environments ([6]-[8],[14]). An overview of several estimate models for object-oriented development environments is given in [16].

From this brief review of cost estimate models, the following conclusions can be made:

- Models based on the most simple metric, KSLOC, are the most reliable because their parameters are calibrated from wide set of projects of different kind.
- Other models, except to some extent Function Points, are specialised and insufficiently applied and confirmed. Estimation using Function Points is very interesting approach but a lot of experience is desirable for its successful application.
- For Object-Oriented software development methods, a variety of new software metrics have been proposed. However, they are also insufficiently applied and confirmed.

3. Foundations of Methodology to Software Development Cost Estimate

Based on the disadvantages of existing methodologies, an attempt was made to construct a methodology for an early estimate of software project development costs, which will be subsequently refined along a software development process. The foundations of the methodology are:

- The Unified Software Development Process (USDP) with Unified Modeling Language (UML), as the dominant approach in software development nowadays;
- The simplest size and complexity metric-KSLOC (thousands of SLOC) and the most widely used COCOMO model to calculate effort and scheduling (according to the conclusions stated in the previous chapter).

3.1. The Unified Software Development Process

The Unified Software Development Process (USDP) is “use-case driven, architecture-centric, iterative and incremental” [11]. It is natural to devise a software development cost estimate process, which is “use-case driven, architecture-centric, iterative and incremental”.

The Unified Software Development Process suggests that large software projects should be decomposed into a set of smaller mini-projects. Each mini-project is *an iteration* in the overall project development and, at the same time, *an increment* of the final product. Within an iteration, a collection of use cases is specified, designed, implemented and tested, making a new increment of software system. (A use case is a textual description of a course of events and system actions to provide visible result for user).

Iterations are distributed over the USDP lifecycle model. It consists of repeating cycles, each having four phases: inception, elaboration, construction and transition (Figure 3.1, [11]). Each phase terminates in a major milestone, where management makes important decisions (on schedule, on budget, and whether to move into the next phase). Each cycle concludes with a product *release* to customers. However, it is worth noting again, that each iteration also results in an internal release (artifact), which adds an increment to the system. These artifacts may be shown to users to get their valuable feedback.

From Figure 3.1 it is possible to see the relationships among the iterations and the "Core workflows" (requirements, analysis, design, implementation and test), of a development cycle [11]. The curves approximate the extent to which the workflows are carried out in each phase.

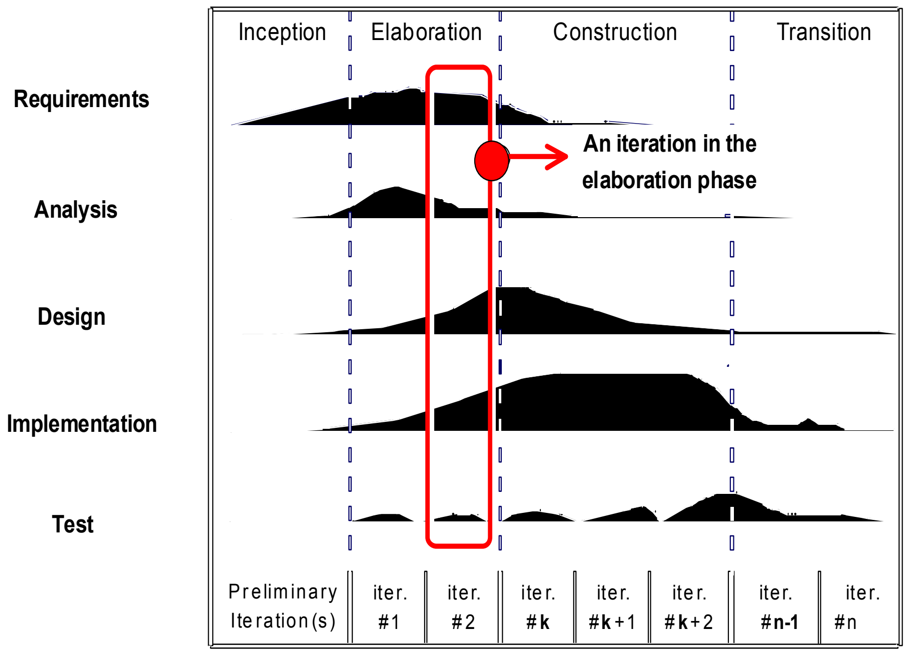


Figure 3.1: A cycle of the Unified Software Development Process [11]

The *Inception phase* launches the project. It is the most creative phase of system development - different ideas are developed into a vision of the end product, a simplified use case model, containing prioritized use cases, is developed, an outline of the system architecture, containing the most crucial subsystems is given, the elaboration phase is planned and the whole project is roughly estimated. In the inception phase, just a few iterations can go through the complete iteration workflows, to support a "proof-of-concept prototype".

During the *elaboration phase*, most of the product's use cases are specified in detail and the system architecture is designed. At the end of this phase, the rest of the project is planned and iterations are sequenced. The plan for the first iteration will be given in detail, while plans for later ones will be iteratively refined. In the elaboration phase some (10% in average) iterations are completed and appropriate increments are built into the product.

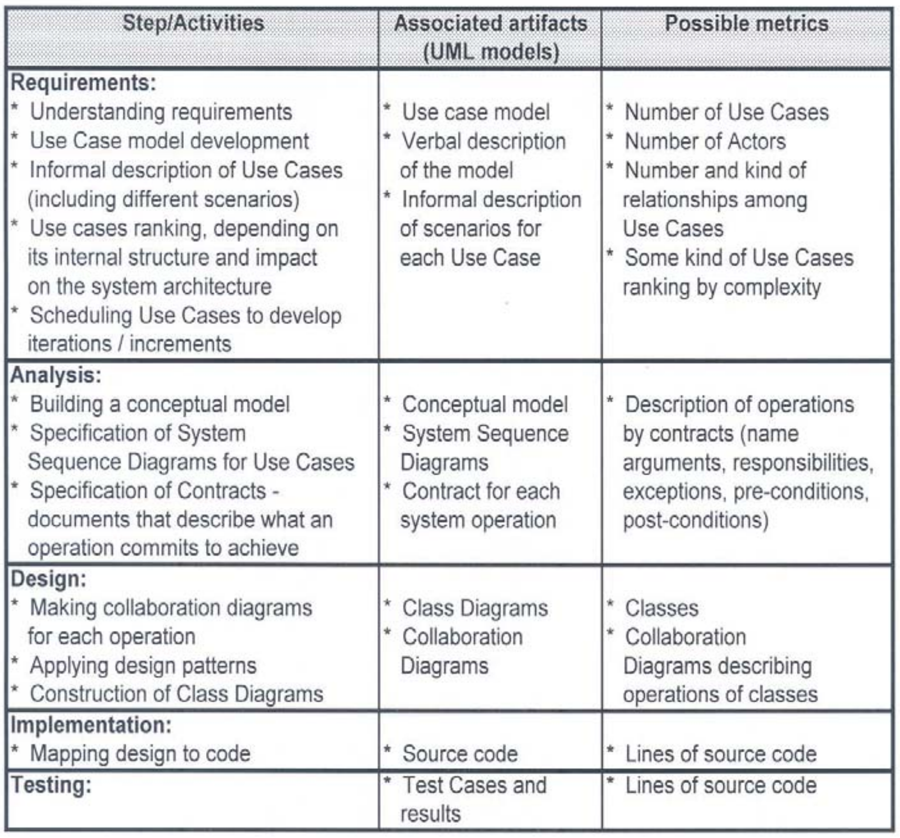
During the *construction phase,* the product is built. All iterations are completed and the product contains the complete functionality specified and agreed between management and the customer. The product (beta release) is ready to begin transition to the user community. The system is turned over to use by a small number of experienced users, who report errors and deficiencies.

The transition phase includes minor fixes and some fine tuning of the product, as well as product documentation, training, etc.

3.2. Iteration lifecycle and artifacts used for estimate

The "core workflows" of an iteration are requirements, analysis, design, implementation and test. Table 3.1. specifies the activities in each step, their resulting artifacts (UML models) and the possible metrics for the size and complexity estimate, as illustration of the relationships among them.

Table 3.1: Core workflows - Steps, activities, artifacts and metrics



Step/Activities	Associated artifacts (UML models)	Possible metrics
Requirements: * Understanding requirements * Use Case model development * Informal description of Use Cases (including different scenarios) * Use cases ranking, depending on its internal structure and impact on the system architecture * Scheduling Use Cases to develop iterations / increments	* Use case model * Verbal description of the model * Informal description of scenarios for each Use Case	* Number of Use Cases * Number of Actors * Number and kind of relationships among Use Cases * Some kind of Use Cases ranking by complexity
Analysis: * Building a conceptual model * Specification of System Sequence Diagrams for Use Cases * Specification of Contracts - documents that describe what an operation commits to achieve	* Conceptual model * System Sequence Diagrams * Contract for each system operation	* Description of operations by contracts (name arguments, responsibilities, exceptions, pre-conditions, post-conditions)
Design: * Making collaboration diagrams for each operation * Applying design patterns * Construction of Class Diagrams	* Class Diagrams * Collaboration Diagrams	* Classes * Collaboration Diagrams describing operations of classes
Implementation: * Mapping design to code	* Source code	* Lines of source code
Testing:	* Test Cases and results	* Lines of source code

3.3. Principles of the software cost estimation method

For the purpose of devising the software development costs estimation method, it is important to recognize several factors:

(1) At the end of the elaboration phase, most of the requirements and analysis workflows are completed. The artifacts resulting from this phase should be used to estimate resources required for completing the project. The estimate of the total development effort can be calculated as sum of the effort estimate for each planned iteration, i.e.

$$E_{tot} = \sum_{i=1}^{n} E_i \qquad (3.1)$$

where n stands for total number of iterations. The estimate of the time to develop an increment is usually calculated using the estimate of the corresponding effort. The estimate of the total time to complete a project may take into account scheduled paralelism of iterations.

(2) However, at the end of the elaboration phase, several iterations (increments) are already completed. This means that the estimates for these increments can be replaced by exact measurements. Using the measured values, the total estimate can be refined using the formula

$$E_{tot}{}^{(k)} = E_{meas}{}^{(k)}(1 + f_k) \qquad (3.2)$$

where
$E_{tot}{}^{(k)}$ is the total effort estimated after completing k-th iteration;
$E_{meas}{}^{(k)}$ is the measured effort for k completed iterations;
f_k is the weighting factor which represents ratio of the complexity of incompleted iterations and the complexity of completed iterations, i.e.

$$f_k = (c_{k+1} + \ldots + c_n)/(c_1 + \ldots + c_k) \qquad (3.3)$$

$$c_j = \sum_{i=1}^{m} \alpha_i^j \qquad (3.4)$$

where
c_j is the complexity of j-th iteration
α_i^j is the complexity factor of the i-th use case in the j-th iteration, and the sum is taken over the complexity factors of all the use cases included in the iteration.

(3) One can use either the formula (3.1) or the formula (3.2) to estimate total effort for software development. However, it is obvious that the formula (3.2) will give better estimate, since it takes into account values measured for already completed iterations. It also reduces the problem of cost estimate for iterative and incremental development to the estimate of the weighting factors given in (3.3).

The estimate obtained at the end of the elaboration phase may be further refined iteratively, after later iterations, using formula (3.2), with the new values of k, f_k and $E_{meas}^{(k)}$. The estimation process closely follows the software development process, becoming itself use-case driven and iterative, incrementally improving the estimate after each iteration.

(4) The Unified Software Development Process, being use–case driven, architecture – centric, iterative and incremental, generally decreases the effort and time needed to complete a project in comparison with "frontal"-"waterfall" approaches. The formula (3.1) corresponds to the "frontal" approach, while the formula (3.2) corresponds to the iterative and incremental approach. Consequently, the formula (3.1) provides the higher and the formula (3.2) the lower boundary of a project effort estimate.

3.4. An algorithm to estimate number of source lines of code

In order to apply the COCOMO model to estimate the effort and development time to complete an iteration, one has to estimate the number of source lines of code (SLOC) from the aritifacts obtained at the end of each step. For an early estimate, it should be done at the end of the analysis step.

There are two possible ways to estimate number of lines of code:

- from the description of typical course of events within expanded format of use cases,
- from contracts prepared for operations [13].

We have chosen the second one, because it gives more detailed insight into the complexity of operations.

The elements of an operation contract are [13] name, responsibility, type, cross reference, notes, exceptions, outputs, pre-conditions and post-conditions.

Conversion from operation contracts to number of SLOC can be performed by means of Function Points (FP). There are two reasons for this approach:

(1)There are several empirical relationships between number of Function Points and number of lines of code (LOC) depending on programming language ([10], [11]). Different languages are ranked into levels from 1 to 55 and a correspondence is established between levels and the number of

LOCs per FP. For most languages, the number of LOC per FP varies from 70 to 125 with an average of 100 LOC/FP.

(2) One can consider section elements of the system operation contract as a specific set of Function Points, because they describe functionality from the user's point of view (see FP definition in chapter 2). In this case, lines of description in a contract table can be treated as the number of the corresponding Function Points.

Of course, the number of lines of description depends on the level of detail of the description – the higher the level of details, the higher the number of FPs but the smaller the number of LOC/FP. To take this fact into account, we propose to establish three levels of detail of description – higher, medium and lower – and to establish three values for conversion of FP to LOC – 70, 100 and 125, respectively.

4. A Methodology for Software Development Cost Estimation

The estimation process starts after the following activities (typical for Object-Oriented approach) have been performed:

- Identification and specification of Use cases;
- Ranking of Use Cases by priority;
- Iteration identification and scheduling

The estimation process consists of the following six steps:

Step 1: Ranking of Use Cases by complexity
Complexity ranking of use cases is carried out based on its internal structure. The aim is to determine Use case complexities, in order to obtain relative complexities of the development iteration. The Use cases are ranked by complexity using a five-point scale, as shown in the Table 4.1. For this ranking, we take into account logical complexity (number of system events) and physical complexity (the number of actors and number of system event attributes).

Table 4.1: Ranking of Use Cases by complexity

Complexity Factor α	Use Case complexity level
0.8	simple
0.9	moderate
1.0	nominal
1.1	high
1.2	extra high

In the literature, five- or seven-point scales are usually applied for human ranking. We decided to use a five-point scale to avoid problems related to making distinctions among steps within scale. The range of complexity factors (from simple to extra high) is proposed as the simplest ranking scheme in the literature [13]. For our purposes, it is acceptable because relative, not absolute, complexity is of interest.

Step 2: Complexity estimate of iterations
The complexity c_k of the k-th iteration (increment) is given by the equation (3.4), as the sum of the complexity factors of all the use cases included in the k-th iteration.

Step 3: Software size estimate
Software size estimate is performed using the approach given in section 3.4.

Step 4: Effort and scheduling estimate by means of COCOMO
After estimating number of LOC for an iteration, one can apply a COCOMO, for example COCOMO II.98, to determine effort and development time using number of KSLOCs, cost drivers and scale factors. However, if an existing component is used, it is necessary to correct the number of KSLOC that is used in formulas for effort and development time estimate, in accordance with the following equation:

$$KSLOC = KSLOC_{step3} - ASLOC + ESLOC \qquad (4.1)$$

where ASLOC is the amount of software to be substituted by an existing component and ESLOC is an equivalent number of new instructions to be written to integrate the existing component into the application. The parameter ESLOC which will be used as COCOMO size parameter, is given by the following equation [4]:

$$ESLOC = ASLOC * (AA + SU + 0{,}4DM + 0{,}3CM + 0{,}3IM)/100 \qquad (4.2)$$

where

AA is the Assessment and Assimilation increment, and deals with the degree of assessment and assimilation needed to determine whether even a fully-reused software module is appropriate to the application, and to integrate its description into the overall product description;
SU is the Software understanding penalty, the cost of understanding and checking the interface;
DM is the percentage of Design Modification;

CM is the percentage of Code Modification;
IM is the original integration effort required for integrating the reused software.

To calculate effort and development time for the k-th increment (iteration), following equations are applied ([4], [9], [10]):

$$E_k = 2{,}94 * EAF * (KSLOC_k)^B \qquad (4.3)$$

$$B = 0{,}91 + 0{,}01(\sum_{i=1}^{5} SF_i) \qquad (4.4)$$

$$EAF = \prod_{i=1}^{17} CDi \qquad (4.5)$$

where the values of cost drivers (CD_i) and scale factors (SF_i) are given in the appropriate corresponding COCOMO tables.

In accordance with estimates established using equations (4.3) – (4.5) and discussion in subsection 3.3 of previous section (See Eq. 3.2), the total effort and development time after k-th iteration $E_{tot}^{(k)}$ and $T_{tot}^{(k)}$ is calculated using following equations:

$$E_{tot}^{(k)} = E_{meas}^{(k)}(1 + f_k) \qquad (4.6)$$

where f_k is given by equations (3.3) and (3.4) and

$$T_{tot}^{(k)} = 3{,}67 * (E_{tot}^{(k)})^{0{,}28+0{,}2*(B-0{,}91)} \qquad (4.7)$$

Step 5: Iterative and Incremental Estimate
Steps 2-4 are repeated for each iteration, incrementally improving the functionality of the system. After the last iteration, the final estimate of effort E_{tot} and development time $T_{dev\text{-}tot}$ are established, because all identified use cases are included.

Step 6: Allocation of total effort and scheduling across global workflows
To allocate total estimated effort and development time across global workflows, the software project is ranked in accordance with number of iterations in one of the following categories – small (S), intermediate (IM), medium (M), large (L), very large (VL) – and Table 4.2 is applied. The

table gives the percentage of workflow's participation in the total effort and development time [4] for each category.

Table 4.2: Workflow participation in total effort and development time

Workflow	E (%)				
	S	IM	M	L	VL
Requirements	7	7	7	7	7
Analysis	17	17	17	17	17
Design	27	26	25	24	23
Implementation	37	35	33	31	29
Testing	19	22	25	28	31
TOTAL	100	100	100	100	100
Workflow	Tdev (%)				
	S	IM	M	L	VL
Requirements	16	18	20	22	24
Analysis	24	25	26	27	28
Design	24	22	21	19	18
Implementation	32	30	27	25	22
Testing	20	23	26	29	32
TOTAL	100	100	100	100	100

5. The POST example

To illustrate the methodology proposed in Section 4, a *Point-of-Sale Terminal (POST)* example is presented in this section. This example is based on the literature [13] and CASE tool *Rational Rose* is applied [19].

POST is a computerized system used to record sales and handle payments, typically used in a retail store. It includes hardware components such as a computer and bar code scanner, and software to run the system. This example is representative of many information systems and touches upon common problems that a developer may encounter.

In general, the goal of the POST system is increased checkout automation, to support faster, better and cheaper services and business processes. More specifically, these include:

- Quick checkout for the customer;
- Fast and accurate sales analysis;
- Automatic inventory control.

A reduced use case diagram for the POST system is shown on Figure 5.1.

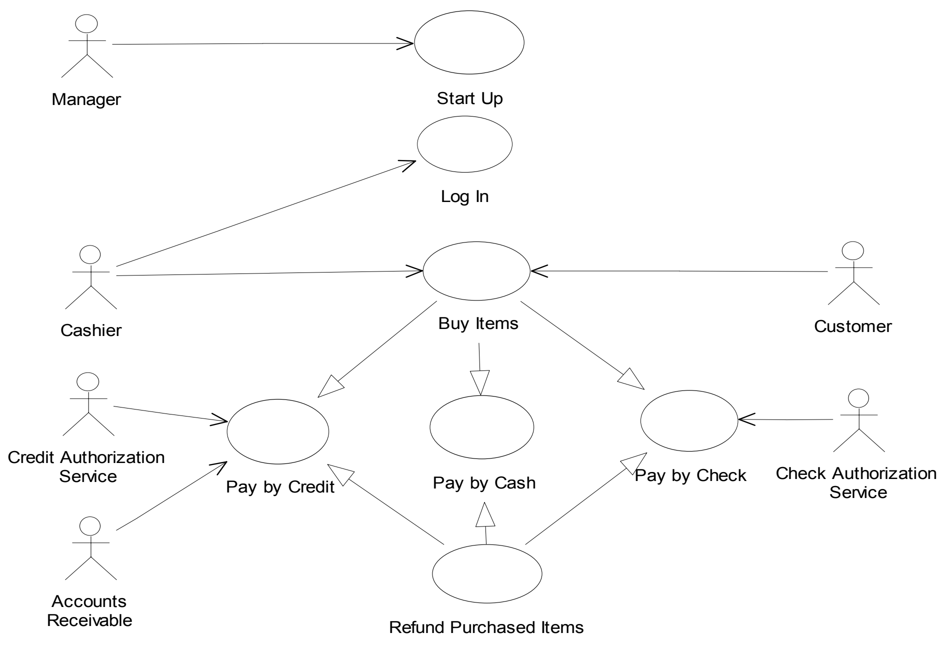


Figure 5.1: Reduced Use Case diagram for the POST [13]

Use cases are identified using an "actor-based" approach. The actors are:

- *Customer* – initiates events when arrives at the POST checkout with items to purchase;
- *Cashier* – handles payments of purchased items;
- *Credit Authorization Service* – authorizes payment by credit;
- *Accounts Receivable* – records and implements payment by credit;
- *Check Authorization Service* – authorizes payment by check;
- *Manager* – Powers on the POST at the beginning of working time in order to prepare it for use by Cashier.

The use case diagram shows main use cases ("Buy Items", "Pay by Cash", "Pay by Credit", "Pay by Check"), special situations ("Refund Purchased Items") and general topics ("Start Up" of the POST by Manager and "Log In" of the Cashier).

The highest priority is the "Buy Items" use case which has four scenarios. At the next priority level are the use cases related to alternative ways of payment. Among them, the most important is the "Payment by Credit" use case which includes not only authorization service but also "Accounts Receivable" as actor.

Use cases are scheduled into four iterations: In the first, only one way of payment is included (payment by cash), in the second all types of payment are included as well as UPC entry by code reader, in the third, refund purchased items as well as inventory maintenance are included and in the fourth, cashiers and POST identification numbers and date of purchasing are included on receipt.

Following the methodology of chapter 4, the estimate process consists of six steps:

Step 1: Ranking of Use Cases by complexity
The complexity of use cases is estimated based on number of actors and taking into account their internal structure (typical and alternative courses of events, number of interfaces etc.). Within the "Buy items" use case there is a branching related to alternatives of payment, which are implemented by separate use cases. Also, this use case includes cancelling a transaction if the customer changes his/hers decision, and covers wrong item identification number entering etc. The "Refund purchased Items" use case has a little less complexity as canceling of transaction is not applicable. The "Pay by Credit" use case includes the most complex procedure of authorization and also the "Accounts Receivable" actor takes part in this way of payment. Other use cases have less complex internal structure.

Step 2: Complexity estimate of iterations
Iterations complexity estimates are calculated using Use Case complexity factors and equation (3.4). These complexities are 2.8, 3.3, 2.2 and 1.9 respectively.

Step 3: Software size estimate
System operations are identified from Use Case Sequence diagrams. Typical diagrams for the "Buy Items" and "Pay by Credit" use cases are shown in Figures 5.2 and 5.3 [13]. Using the contracts written for each system operation, the numbers of lines of description i.e. FPs are determined and converted into number of lines of code.

Based on the assumption that the level of details of description in the POST example is medium (100 LOC/FP), the total number of lines of code is 15 KSLOC, with iterations having 5.5, 4.8, 3.7 and 1.0 KSLOC respectively.

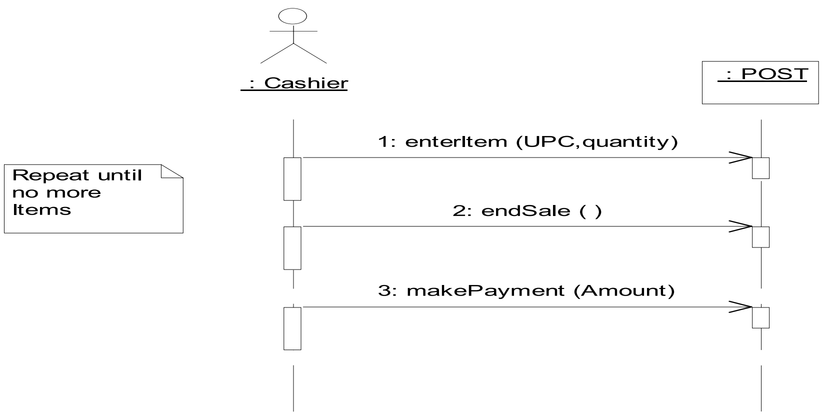


Figure 5.2: Sequence diagram for use case *Pay by Cash*

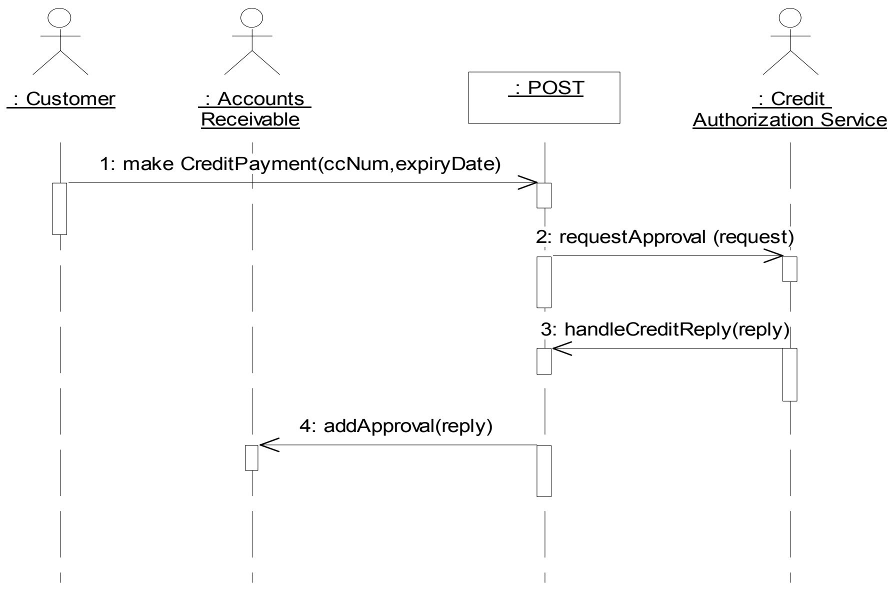

: Customer
: Accounts Receivable
: POST
: Credit Authorization Service
1: make CreditPayment(ccNum,expiryDate)
2: requestApproval (request)
3: handleCreditReply(reply)
4: addApproval(reply)

Figure 5.3: Sequence diagram for use case *Pay by Credit*

Step 4: Effort and scheduling estimate by means of COCOMO
The first iteration of the POST system includes use cases “Start Up”, “Buy Items” and “Pay by Cash”. The size estimate of the software after this iteration is 5.5 KSLOC (relative to total estimation of 15 KSLOC) and the estimate of the iteration complexity factor is 2.8.

Assume that in the POST software development we use an existing software component for the “Pay by Cash” use case. This way of payment is common situation in a retail store and it is reasonable to assume that there is a software component that covers this case. It means that the amount of software to be adopted (ASLOC) is 1.2 KSLOC. Based on assumption that modifications required are small, that difficulties for understanding of the existing component are not too high and that the module is applicable in the POST application with some modifications, from equation (4.1) we obtain that equivalent number of new instructions to be written ESLOC = 0.305 ASLOC i.e. ESLOC=0.366 KSLOC. It means that reduced number of lines of code to be written in the first iteration is 4.7 KSLOC instead of 5.5 KSLOC.

To calculate effort and development time for the POST example using equations (4.3)-(4.5), it is necessary to determine the effort adjustment factor EAF (product of cost drivers) and the sum of the scale factors. The values of the cost drivers and scale factors are determined based on COCOMO II.98. In determining the cost driver's values, we assumed that low experienced personnel are implementing the project. The other parameters are predominantly set to their nominal levels. The scale factors are determined based on the assumption that level of maturity of the organization is 2, team cohesion is large and the application is well known and requires no special constraints. Based on these assumptions, we obtain EAF=1.494 and ΣSF=13.8 and, as a consequence, B = 1.048 and E_1 =22.1 m-m.

<u>Step 5</u>: Iterative and Incremental Estimate

The results of the calculations for all four iterations are given in Table 5.1.

Table 5.1: Results of calculations

k	*SW size (KSLOC)*	$E_{k\text{-}estim}$ *(m-m)*	$E_{k\text{-}meas}$ *(m-m)*	f_k	$E_{tot}^{(k)}$ *(m-m)*	$T_{tot}^{(k)}$ *(months)*
1	4.7	22.1	22.1	2.6	80.5	14.1
2	4.8	22.7	22.7	0.7	74.8	13.8
3	3.7	17.3	17.3	0.2	75.7	13.9
4	1.0	4.4	4.4	0.0	66.5	13.3
4	14.2	66.5	66.5		66.5	13.3

From the Table 5.1 we can conclude that the estimate of total effort after the first iteration is only for 15 m-m (approx. 22%) larger than the final estimate. In case of total development time this difference is negligible (0.8 months i.e. 6%). Also, the final values of effort and development time (66.5 m-m and 13.3 months) are consequence of assumption that experience of the personnel in application domain, platform, language and tool is low. Variation of this group of cost drivers from very low to extra high gives the effort in the range from 142 m-m to 8 m-m. It is a proof of the intuitive fact that effort is very dependent of the personnel capabilities.

In accordance with total software size (14.2 KSLOC), "frontal"/"waterfall" approach gives the estimate of the effort and schedule 70.7 m-m and 13.6 months, respectively, for the software project as a whole. This is an illustration of the fact that iterative and incremental approach decreases the effort and schedule in comparison with traditional "waterfall" approach.

It is interesting to consider sensitivity of the results to variation of number of lines of code. Analysis of equations (4.3) and (4.7) shows that for variation of number of KSLOC within interval ±20%, effort E is changed in interval ±25% and development time in interval ± 9%. If the number of KSLOC changes twice up and down (from – 50% to +100%), effort E varies in interval (-47% to +134%) and development time in interval (-35% do +34%). From this analysis one can conclude that the results obtained are not sensitive to variations of number of KSLOC.

Step 6: Allocation of total effort and scheduling across global workflows
Table 5.2 shows allocation of total effort and development time to global workflows. We assumed that this project, with four iterations and 15 KSLOC, belongs to category "medium" and applied values given in Table 4.2.

Table 5.2: Total effort and development time allocation per workflows

Workflow	Effort (m-m)	Development time (months)
Requirements	4.5	2.7
Analysis	11.3	3.4
Design	16.6	2.8
Implementation	22.0	3.6
Testing	16.6	3.5
Total development	66.5	13.3
TOTAL PROJECT	71	16

6. Directions for future research

Several issues for future work and study are suggested:

Complexity ranking of use cases: We proposed five-point scale to rank use cases per complexity. A difficulty in application of this part of methodology is the fact that criteria for those rankings are neither precise nor exact. Also, formula for iteration complexity estimate will need to be refined and justified by extensive analysis of multiple real software projects. Some research in this area is necessary to quantify the impact of use cases on a system architecture;
Conversion of number of lines of description to KSLOC: We assumed, as a first approximation, that the number of lines of description is equal to number of Function Points. It will be necessary to reconsider and modify (if necessary) this assumption on the basis of further experience with software projects. Also, the conversion from FPs to KSLOC for different languages is an important topic for future research;
Effort and scheduling estimate: Special attention in future research should be paid to effects of reusability of existing components on effort and scheduling estimates. This is very important because we expect that the use of existing software components will dominate software development in the future.
Allocation of total effort and scheduling: This allocation is done based on size categorization as well as the planned distribution of effort and scheduling across global workflows. These assumptions are based on COCOMO. In future works, these assumptions need to be checked and improved, if necessary.

Also, in future research it would be very interesting to couple the use of CASE tools (such as Rational Rose) with project management tools (such as Microsoft Project).

7. Conclusions

This paper presents an attempt to construct a methodology that would enable an early estimate of the software development cost, and enable refinements of the estimate during subsequent development phases. The methodology assumes an Object-Oriented approach based on the Unified Modeling Language (UML) and the Unified Software Development Process. The methodology is illustrated on a POST (Point-of-Sale Terminal) software development project. It is shown that it is possible to construct Use Case driven, Architecture-centric, Iterative and Incremental estimate process that significantly improves and simplifies early cost estimate of software projects.

We can conclude, from the results given in the previous chapter, that the estimates of total effort and total development time after the first iteration are satisfactory and promising. It is worth noting that the sensitivity analysis has shown that the results are not very sensitive to variations of number of KSLOCs.

Also, several questions for future research are emphasized to improve software cost estimate and to enable managers to successfully manage software projects, to implement them within budget resources, in time and to the satisfaction of the user

8. Acknowledgement

The author would like to thank Professor Branislav Lazarevic from the Faculty of Organizational Sciences, University of Belgrade, for his many useful comments and suggestions.

9. References

1. Abran A.,Robillard P.N. Function Point Analysis : An Empirical Study of its Measurement Processes, IEEE, Vol. SE-22, No 12, DEC96, pp. 895-909
2. Albrecht A.J.,Gaffney J.E. Software Function, Source Lines of Code and Development Effort Prediction : A Software Science Validation IEEE, Vol. SE-9, No 6, NOV83, pp. 639-648
3. Boehm B.W. *Software Engineering Economics* IEEE, Vol. SE-10, No 1, JAN84, pp. 4-21
4. Boehm B.W., Abts C., Brown A.W., Chulani S., Clark B., Horowitz E., Madachy R., Reifer D., Steece B. *Software Cost Estimation with COCOMO II,* Prentice Hall, 2000
5. Booch G.,Rumbaugh J.,Jacobson I. *The Unified Modeling Language User Guide,* Addison Wesley, 1999.
6. Cant S.N.,Henderson-Sellers B.,Jeffery D.R. *Application of cognitive Complexity Metrics to Object-Oriented Programs* JOOP, Vol. 7, No 4, JUL/AVG94, pp. 52-63
7. Chidamber S.,Darcy D.,Kemerer C.F. *A Metrics Suite for Object Oriented Design* IEEE, Vol. SE-20, No 6, JUN94, pp. 476-493
8. Chidamber S.,Darcy D.,Kemerer C.F. Managerial use of Metrics for Object Oriented Software: An Exploratory Analysis, IEEE, Vol. SE 1998
9. COCOMO, http://www.softstarsystems.com/
10. Costar, V5 and Calico V5.04, Demo, 20/10/97, http://www.softstarsystems.com/

11.Jacobson I.,Booch G.,Rumbaugh J. *The Unified Software Development Process* Addison Wesley, 1999.
12.Jones C. *Programming Languages Table, Release 8.2* Software Productivity Research, MAR96
13.Larman C. Applying UML and Patterns : An Introduction to Object-Oriented Analysis and Design Prentice Hall, Englewood Cliffs, New Jersey, 1998.
14.Li W.,Henry S.,Kafura D.,Schulman R. *Measuring Object-Oriented Design* JOOP, Vol. 8, No 4, JUL/AVG95, pp. 48-55
15.Putnam L.H. A General Empirical Solution to the macro Software Sizing and Estimating Problem IEEE, Vol. SE-4, No 4, JUL78, pp. 345-361
16.Rakovic R. *Survey of Information System Development Cost Estimation Approaches* Info Science, Vol. 7, No. 2-3, APR/AVG99, pp. 37-48
17.Rakovic R., Lazarevic B. New Methodology to estimate the cost of Software development projects, Info M, Vol.2, No. 5/2003, pp 4-9
18.Rakovic R. A Contribution to Methodology to estimate the Cost of Software Development Projects in Process of Information System Planning and Development, Ph.D. Thesis, Faculty of Organizational Sciences, 2000.
19.Rational Rose 4.0, Rational Software Corporation, 1996.
20.Shepperd M.,Schofield C. *Estimating Software Project Effort Using Analogies* IEEE, Vol. SE-23, No 12, NOV97, pp. 736-743
21.Srinivasan K.,Fisher D. *Machine Learning Approaches to Estimating Software Development Effort* IEEE, Vol. SE-21, No 2, FEB95, pp. 126-136
22.Symons C.R. *Function Point Analysis: Difficulties and Improvements* IEEE, Vol. SE-14, No 1, JAN88, pp. 2-10

Radoslav M. Rakovic was born in 1955. He graduated from the Faculty of Electrical Engineering, University of Belgrade, Yugoslavia, in 1979, received his M.Sc. degree at the same faculty, in 1981 and his Ph.D. degree at the Faculty of Organizational Sciences, University of Belgrade, Yugoslavia, in 2000. Throughout his entire professional career, he was engaged in the problems of planning and designing the telecommunication and information systems and networks. He has published several scientific and specialized papers regarding digital transmission of information and application of information technologies. He has been working as a consulting engineer since 1979 and more than 10 years as the Head of Telecommunication and Information Technology Department in Energoprojekt Entel, Co.Ltd. His current position is the Head of Quality Management System Department.