Effort and Cost Allocation in Medium to Large Software Development Projects

KASSEM SALEH
Department of Information Sciences
Kuwait University
KUWAIT
saleh.kassem@yahoo.com

Abstract: - The proper allocation of financial and human resources to the various software development activities is a very important and critical task contributing to the success of the software project. To provide a realistic allocation, the manager of a software development project should account for the various activities needed to ensure the completion of the project with the required quality, on-time and within-budget. In this paper, we provide guidelines for cost and effort allocation based on typical software development activities using existing requirements-based estimation techniques.

Key-Words: - estimation, resource allocation, requirements, software project.

1 Introduction

One of the main reasons for failures in software development projects is the improper or unbalanced allocation of the resources over the different phases and activities preferred during the execution of the project. The project manager should ensure that enough budget is allocated to critical activities like quality assurance and validation and verification. During the initial planning for the project, requirements-based software cost and effort estimation techniques can be used to obtain an estimation of the overall budget and human resource requirements needed to successfully develop the software. In this paper, we propose a guideline for the proper allocation of effort and budget given an estimation of the overall required effort and budget. This guideline will provide the software project manager with information than can be used to appropriately develop the project schedule and deadlines.

The rest of the paper is organized as follows. Section 2 provides some background information on the software development phases and activities that are needed to develop typical medium to large software projects. Section 3 describes two existing requirements-based estimation techniques, namely, the function point and use case point estimation techniques. Section 4 introduces the proposed effort and cost estimation guidelines. Section 5 provides an illustrative example. Finally, we conclude in Section 6.

2 Software phases and activities

To properly allocate financial and human resources, the various phases and activities that must be undertaken in a typical medium to large software project need to be identified. In the following, we briefly describe two types of activities that must be performed to complete the project, namely, the phased activities and the ongoing life cycle activities.

2.1 Phased software activities

These activities exist in typical software development life cycle models like the waterfall model or the object-oriented model [1].

Analysis

The analysis phase includes the requirements and specifications activities [6]. The main activities involved in this phase include the definition of both functional and non-functional requirements, the definition of the various interfaces between external entities and the software to be developed, and a prioritization of the identified software requirements. The main deliverables of the analysis phase are the scope and vision document, the software requirements specifications document, and the acceptance test plan document. As a result of the analysis phase activities, the functional and non-functional software requirements are well defined and agreed upon by the various software stakeholders. The deliverables of this phase are considered binding documents that guide the rest of the software development activities. It is imperative to spend enough time in this phase to ensure that all aspects of the software are considered, including constraints, assumptions, functionalities, user needs, developmental context and environment, risks, quality requirements, among many other aspects. Studies have shown that most of the serious software errors are those errors that are not captured during the analysis phase. Fixing errors originating from the analysis phase is costly if the errors are not discovered until subsequent phases or during the software operations. Therefore, the deliverables of this phase must be carefully reviewed

and verified (tested) for completeness, correctness, and consistency among other quality requirements.

Design
The design phase activities include the high level architectural, database, interface, and detailed designs. The main deliverables of the design phase include the high level design and the detailed design documents. Design documents are reviewed for quality, completeness, and correctness with respect to the software requirements specifications document. The high level architectural design concentrates on the identification of software modules and their interfaces. In addition, concerns related to design robustness, scalability, security, fault-tolerance, and testability are addressed in the high level design. The detailed design document provides details on each of the modules identified in the high level design. The details include the data structures and algorithms needed to implement each module. The database design presents a detailed description of the database schema needed to support the high level and detailed design. The database design considers the data model described in the software requirements specifications document of the analysis phase. The interface design consists of the design of the graphical user interface components and artifacts needed to support the human interaction with the software system. In addition, all interfaces between the system and other external software and hardware systems, components, and devices are clearly designed.

Implementation
The main activity of this phase is the transformation of the detailed design into an executable code. The code is developed according to the coding standards adopted by the development firm. In addition, the database design is implemented and properly integrated with the produced code.

Testing and Integration
Once the modules of the executable code are tested individually, the developed modules are integrated with external modules, systems, and components. The integration test plans are executed. The obtained test results are analyzed and errors are dealt with accordingly. The deliverable of this phase is the integrated software.

Installation
Once the software is properly integrated, it is delivered to the client premises and installed according to the installation and deployment plan. The client runs the acceptance test plan and, ideally, accepts the software. The deliverables of this phase are the official acceptance document signed by the client and the properly installed software system.

2.2 Ongoing life-cycle activities
In addition to the phased activities, there are activities that are performed continuously while the phased activities are performed. These ongoing activities are briefly described below.

Project Management
During the progress of the life cycle activities, the project manager continuously performs project management-related activities. The progress of the activities is closely monitored using an appropriate reporting procedure. Risks are continuously monitored and corrective actions are taken when needed. In addition, new risks are identified and monitored. The project schedule is updated regularly as needed. Project- and process-related metrics are regularly collected and assessed. In addition, human resource management and project management activities, including delegation and evaluation, are performed.

Quality Assurance
During each phase, the quality assurance group performs its activities, including the review of the deliverables of each phase and ensuring the use of and conformance to internal and external standards. Reviews of parts of the deliverables are conducted during the phase execution and after the deliverables are produced. Quality reports and logs are maintained and relevant metrics are collected according to the metrics collection plan. The quality assurance plan is executed and updated if needed. Moreover, process improvement recommendations are provided by the group after completion of the project.

Evaluation and Testing
Evaluation and testing processes and activities, including validation and verification activities, are performed all along the various product development phases. At the end of each phase, the phase deliverable is evaluated and tested. Once the deliverable is approved, the next phase starts. Formal and informal processes are used for evaluation and testing purposes. The processes are performed by the developers themselves and by independent groups such as quality assurance. Informal evaluation processes include reviews that might involve walkthroughs, inspections, and audits. Formal evaluation processes include the use of formal techniques and automated tools for the verification of phase deliverables. Tools and techniques for design verification and code testing can be used for these purposes. Formal product validation techniques and methods can also be used to generate

and document effective tests, automate the validation process, and analyze and validate the results.

Configuration Management
Configuration management (CM) activities initially include the identification of all software documents, deliverables, and artifacts that will be produced, manipulated, and maintained during the development and maintenance of the software. Configuration control activities are then performed continuously during the software development and post-development phases. CM deals mainly with the management of the software resources and the overall support and control of the software development and maintenance processes. The CM control activities include revision and version control, process and workflow control, build control, and change control. Ideally, the control activities are performed using an integrated and automated CM tool for revision, version, build, and change management. Finally, a periodic CM audit and status report is generated by the automated tool. The report is then analyzed by the appropriate management team.

Technical support and internal training
During the development and maintenance of a software product, the developers might require some technical support and training. Support technicians help the developers in solving technical problems that arise while developing the software. The activities help improve the efficiency and productivity of the development and maintenance teams. A plan can be devised as part of a project plan to deal with the training of technical staff on the development process, or new tools and technologies that are needed during the development phases.

Documentation
During product development, various documents that target different audiences are produced. Some of them are internal technical documents that are needed for future software maintenance activities. Other documents target external users and include user manuals, installation manuals, and operations manuals. Standards and standard templates are normally used to guide the writing of the software-related documents. Technical writers are involved with the production of the external documents. Internal documents are typically written by the software developers themselves. The documents are evaluated by internal review processes for quality involving various stakeholders, including software development team members and representatives of the software quality assurance group.

Figure 1 shows the typical life cycle model including both the phased and ongoing activities we presented in this section.

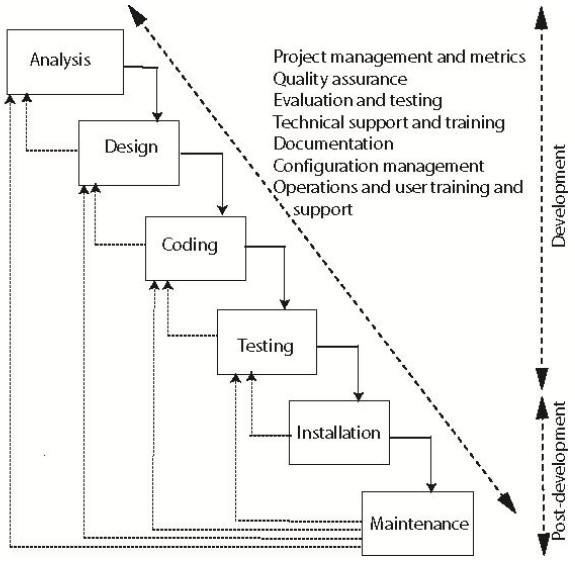


Figure 1. Phased and ongoing activities.

3 Effort and cost estimation and reconciliation

In the following, we briefly present two well-known requirements-based cost and effort estimation techniques. We then show how we can reconcile using a third input from an expert opinion.

Function point estimation
Function point (FP) metrics, introduced by Allan Albrecht in 1979 [2-5], are specification-based metrics that are used to estimate the effort needed to develop a software system. Function points are implementation independent. Computing the number of function points, for the project is independent of the design choices taken, the tools used, or the programming language utilized to implement the system. According to FP metrics, the complexity of software and the effort needed to develop it are a function of the number and type of five kinds of functional components that can be obtained and assessed at the requirements specifications phase. These five functional components include:

1. Internal files corresponding to the database files that are created and maintained within the application
2. External files corresponding to the files that are owned and maintained by other applications but used by the application
3. External inputs corresponding to the inputs that affect the control flow and internal logic of the application leading to the creation and maintenance of data

4. External outputs corresponding to the data leaving the application to different output devices, files, or external systems
5. External inquiries corresponding to simple user queries resulting in responses to them

These functional components should also be classified under one of the three complexity levels: simple, average, and complex. Simple internal and external files include few records with simple structures. Complex internal and external files have a large number of records with complex structures. External inputs is information from the outside to the inside, originating from user interfaces or from other applications. Simple external inputs include control information and simple business information affecting one internal file. Complex external inputs contain business information originating from the outside to the inside and triggering the update of two or more internal files. Complex external outputs refer to many data subsets across many files. The complexity of an external inquiry is obtained by taking the greater of the complexities of the input and output parts of the inquiry. Various weighting factors are then used for each of the five types of functional components and for each of the three complexity levels. The number of unadjusted function points (UFP) is obtained by summing up all factors assigned to an identified component according to its complexity. To take the context and the type of software project into account, Albrecht has introduced a list of 14 technical factors that influence the effort needed to complete the project. The product of these factors is then used to adjust the number of function points. Each of these factors must be rated on a scale 0 to 5. A rating of 0 means that the factor is irrelevant or has no influence, and a rating of 5 means that the factor is essential and has a strong influence. A rating from 0 to 5 is assigned to each factor. Their sum is computed to obtain a value for S. The overall complexity factor CF is then computed using the equation: $CF = 0.65 \times 0.01 \times S$. CF is within the range 0.65 to 1.35. The number of adjusted function points (AFP) is UFP × CF.

To map the number of adjusted function points (f) to the needed effort in person-months, Jones proposed a first-order estimation as a function of f and an exponent j to compute the effort in person-months using the equation: $m = f^{3*j}/27$ person-months. j depends on the type of the software application involved and the capabilities and expertise of the development team and varies from 0.39 to 0.48

Use case point estimation

The use case point (UCP) is a software effort estimation technique that was introduced by Gustav Kamer in 1993 [3-5]. It is based on the use cases existing in the use case model of a software system. In UCP metrics, actors and use cases are classified under three categories: simple, average, and complex. For example, an external system interacting with the system using defined application programming interfaces is typically a simple actor. External systems interacting with the system using some standard protocols and data stores are typical actors of average complexity. A user interacting with the software using graphical-user interface components, such as forms and dialog boxes, is considered a complex actor. The complexity assessment of a use case is based on the number of transactions or steps that are included in the use case description. These steps are included in the normal and alternative flow of events in the use case description. A use case is classified as simple if the number of transactions does not exceed 3. Similarly, an average complexity use case includes 4 to 7 transactions and a complex use case includes more than 7 transactions. Factors are assigned to the various complexities of both actors and use cases. The unadjusted actor weight (UAW) is the sum of complexity values assigned to each actor. Similarly, the unadjusted use case weight (UUCW) is the sum of complexity values assigned to each use case. The total unadjusted use case point (UUCP) is the sum of UAW and UUCW. The number of adjusted use case points (AUCP) is computed by multiplying the UUCP with the product of two adjustment factors: the technical complexity factor (TCF) and the environmental factor (EF).

The TCF is obtained using the equation: $TCF = 0.6 + (0.01 \times TF)$, where TF is the sum of all the weighted values computed for each of the 13 technical factors. Technical complexity factors are mainly related to the product and its complexity in terms of functional and non-functional requirements (NFRs). Each factor has its own weight, and a value ranging from 0 to 5 is assigned to a factor, depending on the technical complexity of the corresponding factor. Similarly, the EF is obtained using the equation: $EF = 1.4 + (0.03 \times ENVF)$, where ENVF is the sum of all the weighted values computed for each of the 8 environmental factors. Environmental factors are related to the people, process, and project aspects of the software. Each factor has its own weight, and a value ranging from 0 to 5 is assigned to a factor, depending on its relevance. For example, the stability of requirements is given the highest weight of 2 and if the requirements are felt to be volatile, a high value of 5 is assigned to it, making the weighted value 10. The equation to obtain the number of adjusted use case points is: $AUCP = (UAW + UUCW) \times TCF \times EF$.

To obtain the estimated effort in person-hours needed to develop the software according to the UCP metrics, Kamer stated that 20 person-hours are needed for each use case point. However, other refinements and empirical studies of the UCP technique suggested a

range between 15 and 30 person-hours per UCP. Assuming we use p person-hours per UCP and a work day of h hours, the number of work days would then be: ((p × AUCP)/h) days.

Reconciliation

If the two estimates obtained using the FP and UCP techniques are far from each other, we can obtain a third estimate from an expert in the application domain. If the three estimates, E_{low}, E_{high} and E_{mid} are obtained such as $E_{low} < E_{mid} < E_{high}$, the reconciled value for E is then: $E = (E_{low} + 4 \times E_{mid} + E_{high}) / 6$.

4 Effort and cost allocation

Reported experiences on the time and budget spent on the different software development activities show the following facts:

a. Quality and testing related activities, including integration testing, quality assurance, and evaluation and testing, account for about 37% of the overall effort.
b. Ongoing activities, covering project management, configuration management, documentation, and support and training, account for about 21% of the overall effort.
c. Phased software development activities, including requirements, specifications, design, implementation and deployment and not including evaluation and testing) account for about 42% of the overall effort.

Table 1 shows the details of the effort distribution for each type of activity and the approximate pay rate relative to the project manager pay rate. According to the above facts and to the pay rate shown in Table 1, the costs of performing the activities in the above facts a, b and c, are 36%, 20% and 44%, respectively, of the overall development cost.

Table 1. Effort distribution on activities and relative pay rate.

Software phases	**% effort**	**Pay rate**
Requirements	7.5	0.95
Specifications	7.5	0.95
Design	10	0.95
Implementation	10	0.85
Integration testing	7.5	0.9
Acceptance & deployment	7.5	0.9
Ongoing life-cycle activities		
Project management	8.34	1.0
Configuration management	4.16	0.75
Quality assurance	8.34	0.8
Documentation	4.16	0.7
Training and support	4.16	0.8
Evaluation And testing (V&V)	20.84	0.9

In this table, we are assuming that in medium to large software projects, the team follows a functional organization, in which a team member specializes in one activity. However, we can estimate the average salary of a non-managerial resource to be the average for all other activities. This average would then be 87.5% of the project manager salary.

As an example, let us assume that the reconciled estimate of the effort needed to develop a software product is 240 person-day and the project manager daily pay rate is 500 dollars. Based on Table 1, we can then obtain the total cost of the software and the effort distribution as shown in Table 2.

The total cost is 106650 dollars and the average rate for one person-day is 445 dollars. We can also schedule this project to be completed by 12 persons within 20 days according to the effort distribution shown in Figure 2.

Table 2. Example effort and cost allocation to software activities

Software phases	**effort**	**cost**
Requirements	18	8550
Specifications	18	8550
Design	24	11400
Implementation	24	10200
Integration testing	18	8100
Acceptance & deployment	18	8100
Ongoing life-cycle activities		
Project management	20	10000
Configuration management	10	3750
Quality assurance	20	8000
Documentation	10	3500
Training and support	10	4000
Evaluation And testing (V&V)	50	22500
Total	240	106650

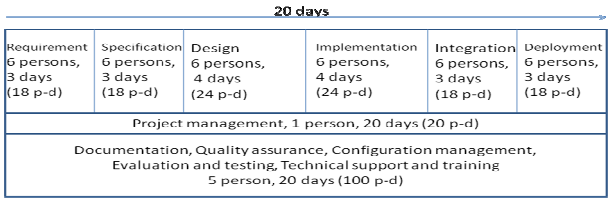
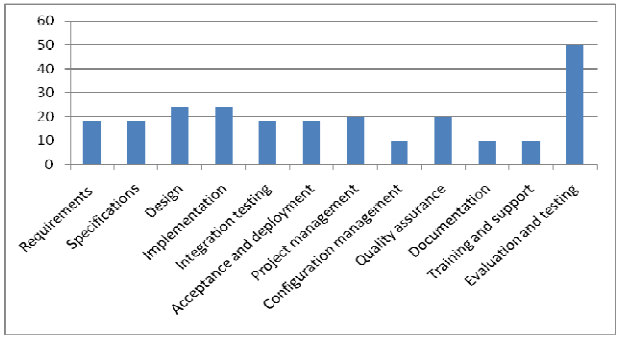


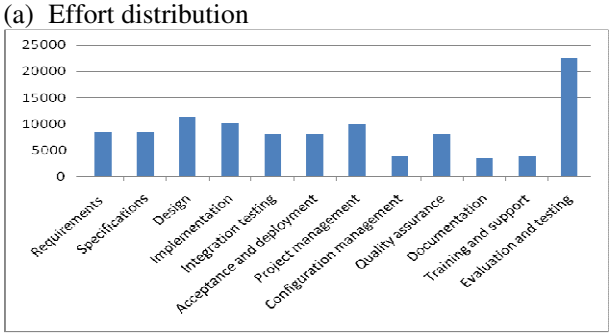
Figure 2. Schedule and effort distribution.

The charts showing the effort and cost distributions among the phased and ongoing activities of the 240-day software development project areare shown in Figure 3.

Excel sheets for the cost estimation and effort distribution are available from the author upon request.



(a) Effort distribution



(b) cost distribution

Figure 3. Effort and cost distribution.

6 Conclusion

In this paper, we have provided guidelines for the proper allocation of budget and human resources on the various activities of the software development process. The guidelines are based on some critical activities needed in medium to large software projects and on two requirements-based cost and effort estimation techniques. Further study on the margin of error of this guideline should follow in an empirical study based on metrics collected from various software projects. It is also interesting to analyze the guideline for various types of software applications.

References:

[1] K. Saleh, Software Engineering, *J. Ross Publishing*, Florida, USA, 2009.

[2] A. Albrecht and J. Gaffney, "Software function, source lines of code and development effort prediction: a software science validation", *IEEE Transactions on Software Engineering*, November 1983.

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

[4] S. McConnell, Software Estimation: Demystifying the Black Art, Microsoft Press, 2006.

[5] M. A. Parthasarathy, Practical Software Estimation: Function Point Methods for Insourced and Outsourced Projects, Addison-Wesley Professional, 2007.

[6] K. Wiegers, Software Requirements, Microsoft Press, 2nd edition, 2003.