IMPORTANCE OF DATA COLLECTION AND VALIDATION FOR SYSTEMATIC SOFTWARE DEVELOPMENT PROCESS

Mala.V.Patil[1] and Dr. A.M.Nageswara Yogi[2]

[1]Research scholar, Anna University, Coimbatore, INDIA
malapatil2002@yahoo.co.in
[2]Scientist, ADE, Defence Research and Development Organization, Bangalore, INDIA
amnyogi_jalaja@yahoo.co.in

ABSTRACT

Systematic software development process involves estimation of size, effort, schedule and cost of a software project and analysis of critical factors affecting these estimates. In literature there are many methods for software estimation and categorization of critical factors. More than 50% of the projects undertaken have challenged the initially proposed estimates. Even if we consider updating estimates at various phases of software development, the percentage of challenged projects reduces marginally. The reason for such a situation is that the decisions are made on historical and collected data. Therefore, software data collection to a reasonable accuracy and its validation is important both for decision making and validating software development process. In this paper an effort is made to highlight the importance of software data collection. Collected data is utilized to validate effort estimation model formulated by the authors. Comparison of effort values obtained from popular estimation models is also made. The data collected has also helped in identifying the critical factors affecting the estimates.

KEYWORDS

Software Size, Effort, Cost, Schedule, Risk, Estimation.

1. INTRODUCTION

Software project managers were in lot of trouble during 1950s, due to many failures in timely completion of projects and overshooting the estimated budget. This was because project managers were depending on crude estimates (size, effort, cost and schedule) for taking up projects. Estimates were calculated based on historical and collected data on completed projects. Estimation of effort and resources required for a project is still a challenge due to the neglected effect or under/over quantification of intangible parameters like inexperience, unclear requirements, unfamiliar future technologies, development environment and complexities in design and development. As a result the actual effort will be 4 times [4] the order of the magnitude of the estimated values. Estimates can be roughly grouped into three types: Ballpark or order of magnitude [35] would be two or three times the actual value when estimations are made during initial stages of the project proposal. Rough estimates which are 50 to 100% off the actual values can be estimated when we are working with well-understood need and familiar with domain and technology issues. Fair estimates which are about 25 to 50% off the actual values are possible to obtain when we know what needs to be done and have done many times before.

Software projects are typically controlled by four major variables time, requirements, resources (people, infrastructure, money) and risks. Unexpected changes in any of these variables will be disastrous to the project. Hence making good estimates of time and resources required for a project is crucial. Underestimating project needs can cause major problems. Overestimating needs can be very expensive for the organization. These problems led to the development of software estimation models during 1970s which were based on a single parameter 'size' in terms of Kilo Lines of Code (KLOC) to be delivered. In the initial stages of project proposal many things will not be clear; hence estimated KLOC will be approximate. According to Boehm [1] early models differed widely in their underlying assumptions and definition of size and hence cost equations varied substantially. Although the effort for a project is a function of many parameters, the primary factor that controls the effort is the size as generally agreed by many experts. A model developed by Doty Corporation in 1977 addressed fourteen factors including display requirements, memory, etc. During the same time RCA PRICE Systems released a parametric model viz. PRICE-S which used program size, complexity, memory, timing constraints, application and resource factors to estimate software development cost and schedule. In 1979, QSM Software Life Cycle model (SLIM) developed by Larry Putnum was made available to software engineering community. In mid 1980s, the CHECKPOINT, SEER-SEM (Jensen JS I and II models), SoftCost-R models were released. Dr. Boehm's COCOMO model [1] was made available free of charge. By mid 1980s, software project managers had several software estimation models to choose. However the predictive accurancy of models was not less than one and half times the actual values [9].

During 1990s and 2000s, we noticed that both hardware and software technologies had changed very fast. Development of Object Oriented technologies, distributed computing, web engineering with fourth generation languages (4GL), it was required to redefine the definition of size and other parameters required for effort estimation. Some of the estimation techniques for Object Oriented and WebApp projects are discussed in [26]. In addition to size of the software, estimations must be based on parameters like number of front-end forms to be designed, their complexities and interactivities, networking requirements, number of documents to be prepared, complexity of database to be created etc. Therefore software data collection became one of issues for software engineering community. It is wise to make the estimations at different stages [21] of the project such as Request for Proposal (RFP), Preliminary Design Review (PDR), Critical Design Review (CDR), and Prototype Testing (PT), since more and more specific data may be available as the project progresses. Even then overshoots to schedule and cost estimations for software projects cannot be avoided due to many other critical/risk factors affecting estimates.

Therefore, every project manager is required to perform methodical investigation of risk in order to avoid cost and schedule overruns. The word risk originated from the Italian term Risicare means to dare. As per the dictionary risk is the chance or possibility of loss/damage. Peter Drucker [34] says, "While it is futile to try to eliminate risk from a process, it is essential that risks taken are the right risks". Risk Management involves risk assessment and risk control [6]. Risk assessment involves risk identification, risk analysis and risk prioritization, on the other hand risk control involves risk management planning, risk resolution and risk monitoring. Over all aim of risk management is risk mitigation. Software risk is a measure of the probability of its occurrence and the loss due to its exposure. The type of the risks in software development are Technology risks – risk of chosen hardware and software becoming obsolescence, People risks – capability, skill, maturity etc, Organizational risks – environment

at developer as well as the users organizations, Requirements risks – clarity, completeness and confirmation of requirements by the end user, Estimation risks – judgment errors in estimation of size, effort and schedule of the project. In addition development of functions not required, using improper interface, beautifying interface screens, shortfall in externally performed tasks, integration of the off-the-shelf components, etc will also lead to failure or schedule overrun of a project. However Jingyue Li et al [13] have collected data from 133 projects and analyzed that off-the-shelf components do not contribute negatively to the off-the-shelf based software projects. Hence a project manager must spend sufficient time in choosing interface, vendors and also discuss with experts or learn from the past experience regarding the development of functions and the properties. Therefore a proactive and reactive risk management is necessary to reduce the risks involved. It is required to analyze risk events and drivers and their impact in terms of losses. Barry Boehm [6] says that identifying and dealing with risks early in development lessens long-term costs and helps prevent software disasters. Marvin J Carr et al [20] have developed a method namely "Taxonomy – Based Risk Identification" based on their previous experience and other published literature. This method facilitates the systematic and repeatable identification of risks associated with the development of software-dependent software. For software-intensive systems, effective system integration necessitates that all functions, aspects and components of the system must be accounted for along with an assessment of most risks associated with the system. C.G. Chittister and Y.Y. Haimes [8] says that the process of risk assessment and management is also the *sine qua non* requirement for ensuring against unwarranted time delay in a project's completion schedule, cost overrun and failure to meet performance criteria. The analysis of Janne Ropponen and K. Lyytinen [12] shows that awareness of the importance of risk management and systematic practices to manage risks have an effect on scheduling risks, requirements management risks and personnel management risks. In their paper the authors have derived six software risk components such as scheduling and timing, system functionality, subcontracting, requirement management, resource usage and performance and personnel management risks. Mark Keil, et al [19] says using a systematic approach tapped the experience of more than 40 software project managers from around the globe to identify a universal set of risk factors. The three most important critical/ risk factors were judged to be lack of top management commitment to the project, failure to gain user commitment, and misunderstanding the requirements.

Critical/ Risk factors that are likely to have a bearing on the success of a project are to be identified and listed. Boehm [3] and Marvin J Carr et al [20] have compiled several lists of possible critical/ risk factors. These critical factors have been classified as having Technology, People, Organizational dimensions etc in [8, 12, 16, 19 30]. Based on this classification the probability and severity of the loss for each critical factor can be assessed by risk analysis and risk prioritization. Risk mitigation tasks have to be performed for a software project by defining responsibilities, activities and resources. Risk mitigation must take into account the process, organization [2, 12 and 19] and technology [11]. Risk resolution produces a situation in which the risk items are eliminated. Risk monitoring involves tracking the progress of a project towards resolving risk items.

Risk management is particularly important in software development projects due to the inherent uncertainties that most software projects face. Project managers have to anticipate risks, understand their impact on the project, and take steps to avoid them [28]. It is unrealistic to develop universally applicable and complete risk taxonomy [31] since different risks classifications may be required by different projects because of changing technological

environment. Therefore, there is a need for empirical research to provide information about the effectiveness of proposed risk mitigation tools and techniques [24] even though several risk-reduction strategies have been proposed in [2, 10, 11 and 12]. Rasmita and Rajshree [23] conclude in their paper that proper and timely risk management control can provide enormous advantage to the organization by cutting down costs and keeping project schedule.

2. SOFTWARE DATA COLLECTION

The Standish Group publishes biennial reports [29] on the rate of success of software projects in "Chaos Report" from 1994 to 2009 based on the data collected either through surveys or questionnaire. Even though successful projects increased from 16% in 1994 to 32% in 2009, but 44% projects overshot cost and time and 24% projects failed in 2009 report. Chaos Report is challenged by Eveleens and Verhoef [15] by stating that there are discrepancies in the definition of challenged, failed and successful projects. Chaos report considers that a project is challenged if it exceeds initially estimated cost and time. Whereas in [15], it is brought out that estimates are updated as and when more data is available at every phase of project progress. Jorgesen and Molokken [14] examined the cost overrun statistics over the years in the Chaos report 1994 and estimated that the real average is close to 33% against 54% cost overrun figure of Chaos report. CG. Jones et. al. [7] analyzed Chaos report success factors by ranking them from 1994 to 2008. Eveleens and Verhoef in [15] claim that Chairman of Standish Group has replied that "All data and information in the Chaos reports and all Standish reports should be considered as Standish opinion". Therefore, software data collection is very important and critical to the analysis. Data collection is the term used to describe a process of preparing and collecting data. Processed data known as information is useful for process improvement and decision making. A formal data collection process is necessary as it ensures that data gathered is both defined and accurate and that subsequent decisions based on analyses of data are valid [27]. Right decisions can be made only with right data. A fundamental limitation to significant progress in software estimation is the lack of unambiguous, widely varied standards for software data. Roger S. Pressman [25] states that to be effective aid in strategic planning and/or cost and effort estimation, baseline data must have the following attributes:

1. data must be reasonably accurate – 'guestimates' about past projects are to be avoided,
2. data should be collected for as many projects as possible,
3. measurements must be consistent – for example, a line of code must be interpreted consistently across all projects for which data are collected or FP must computed or backfired in a predictable manner,
4. applications should be similar to work that is to be estimated – it makes little sense to use a baseline for batch information systems work to estimate a real-time microprocessor application.

2.1. Importance of data

Importance of data collection to a reasonable accuracy can be illustrated by the following two examples. The successful development of COCOMO models and its variants was due to the database which was created with clear data definitions and an associated project database which are available for general use and reasonably compatible [1]. Waltson and Felix [33] analyzed data of more than 60 projects done at IBM Federal Systems Division, ranging from 4K to 467K lines of delivered source code and found that if the size estimate is in thousands of delivered

lines of code (KLOC), the total effort E in person-months (PM) can be given by $E = 5.2 (Size)^{0.91}$.

Collection of data will not be easy [25]. Many things can go wrong and create problems during software data collection work. Some of potential problems are: Inadequate and/or Out-of-date, Faulty data, Improper analysis, Inappropriate presentation and Time lag. The authors felt many difficulties in collecting the software data for the validation of the formulated method [22] for 4GL environment. Some of the difficulties are:

1. Absence of systematic process for collection of data,
2. Project managers say that they do not have time either to collect the data or share the same,

3. Unwillingness to share the data/ information,
4. Bias towards the person requesting information,
5. Personal egos,
6. Careless or not bothering to reply the questionnaire.

In the opinion of the authors, there is no commitment across the organizations to establish and use a set of clear and uniform software data definitions or standards. It is important to set goal-directed data collections from the beginning of the project itself. The data so collected will help to evaluate methodologies and the claims made by software engineers. Analysis of collected data will answer questions of interest and also it may generate some questions related to research work. The collected data must be validated otherwise more than 50% of the data may be erroneous [32].

2.2. Data from developed projects

As stated earlier, software project data must be collected during the early stages of the software project itself. The data thus collected is required to be modified and refined as the project progresses. The methodology used by the authors was to circulate a questionnaire. The questionnaire was distributed to more than 20 teams working at different organizations for collecting relevant data of the software projects undertaken by them. The format of the questionnaire is given in Appendix A. The authors explained the goals and objectives of data collection to team leaders and team members and suggested to fill the data as and when available and validate the entered data. It was told to the team members to update the information as and when it changes and reasons for the changes. Out of 20 teams only 12 teams returned the questionnaire with reasonably useful and meaningful data as per our requirements and same is tabulated in Tables 1 and 2. It can be observed that data requested does not contain any classified information. In a similar way data was collected from engineering student trainees in [17] and from software professionals in [18] and the collected data was utilized to validate the formulated effort estimation methodology by AMN Yogi and Mala V Patil [22].

Table 1. Projects details of different organizations.

Sr. No.	Project title	Organization in Bangalore	NPT	FE	BD
1	MDBSS	Prateck Tech.	3	Java	NIL
2	HEIC	SPIRO	4	Java	NIL

3	Ed3m	INFICS Sols.	4	C#.net	SQL Server
4	NBPPCC	X Sys Tech.	4	Java Net beans	SQL Server
5	ASEPPN	Rajiv Gandi Institute Of Tech	4	Java	MS ACESS
6	BS	Prateck Tech.	4	Java	NIL
7	CASDP	LSI Tech.	4	Java	NIL
8	HDWSN	PASC Tech.	3	Java	SQL Server
9	TDMCI	X Sys Tech.	4	Eclipse	SQL Server
10	SOME	Master Skills	4	Java	SQL Server
11	OTFRTGE	Mindset Tech.	4	Java	NIL
12	BEMC	Aliphatic Tech.	4	Java	MSACESS

Table 2. Data about projects

Sr. No	KLOC	ED(PM)	AD(PM)	NFF	NPD	N/W	NT	RP
1	1.2	9	10.5	7	75	1	2	2
2	1.5	10	16	7	75	0	1	2
3	1.5	8	12	6	80	0	1	0
4	2.0	12	12	7	80	1	0	5
5	2.5	12	16	15	80	2.5	2	1
6	2.7	12	14	6	82	0	1	2
7	3	16	16	6	88	0	0	2
8	4.5	9	13.5	10	60	12	1	2
9	5	12	20	9	85	1	5	2
10	5	10	14	8	60	0	2	1
11	6	12	16	7	85	0	0	1
12	7	12	14	12	80	2	15	1

Note: Only codes names of the project titles have been given.

Expansion of the symbols used in Table 1 and 2 are as follows:

NPT = Number of Persons in Team,

FE = Front-End Documented

BD = Backend Database

ED(PM)= Estimated effort for project Development

AD(PM)= Actual effort for project Development

NFF = Number of Front-end Forms

NPD = Number of Pages

N/W = Network hours

NT = Number of data Tables

RP = Number of Reports generated

We observe that the majority of the projects have used JAVA for front-end forms and SQL for backend database. It can be observed that actual effort is not proportional to either Kilo lines of code or Number of front-end forms developed. We made linear, quadratic and exponential

regression analysis for effort (E) as a function KLOC. The results obtained are tabulated in Table 3.

Table 3. Analysis of effort data

Sr. No.	AD	$E = a_1x + a_0$	$E = a_2x^2 + a_1x + a_0$	$E = ax^b$
1	10.5	14	12	2
2	16	14	13	3
3	12	14	13	3
4	12	14	14	4
5	16	14	15	6
6	14	15	15	7
7	16	15	16	8
8	13.5	16	17	15
9	20	16	17	17
10	14	16	17	17
11	16	16	16	23
12	14	17	15	29

The values of linear regression constants found are $a_1 = 0.52$ and $a_0 = 12.67$. For quadratic, regression constants are $a_2 = -0.34$, $a_1 = 3.20$ and $a_0 = 8.71$. For regression formula $e = a\,x^b$, the constants are a = 1.56 and b= 1.5. It can be observed that the effort estimated by all three regression formulae varies directly to Kilo Lines of Code. All three methods give rather different values. The actual effort matches only for a few projects only. Therefore, effort values are not only the function of size that is KLOC, but also of many other parameters such as complexity of design, experience and maturity level of developers, environment, team cohesion, etc and most of these parameters are intangibles. Accordingly size of the project should be modified based on either number of modules to be developed or on the design parameters.

3. EFFORT ESTIMATION MODELS BASED ON KLOC

A typical estimation model is derived using regression analysis on data collected from past software projects as explained in the previous paragraph. The overall structure of such models takes the form

$$E = A + B\,(SIZE)^C,$$

Where A, B, and C are empirically derived constants, E is the effort in person-months and SIZE is the estimation variable either in KLOC or Function Points (FP). Most of the models use adjustment components to cater for other project characteristics. The constants derived for the following LOC-oriented models for which SIZE = KLOC available in the literature are as follows:

Bailey-Basili model A = 5.5, B = 0.73, C = 1.16.
Boehm simple model A = 0.0, B = 3.20, C = 1.05.
Doty model A = 0.0, B = 5.288, C = 1.047.
Walston- Felix model A = 0.0, B = 5.2, C = 0.91.

4. FORMULATED EFFORT ESTIMATION MODEL FOR 4GL ENVIRONMENT

A methodology for effort estimation for a software project to be developed under 4GL environment was formulated and published as AMN Yogi and Mala V Patil method in [22]. Basic idea behind this method was drawn from Boehm's COCOMO (COnstructive COst MOdel) II model. COCOMO II application composition model uses Object Points as a basic parameter. Object Point is an indirect software measure that is computed using counts of the number of Screens, Reports to be generated and 3GL components likely to be required to build the application. Each object is classified into one of three complexity levels i.e. Simple, Medium and Difficult using the criteria suggested by Boehm [5]. In essence complexity is a function of the number and source of the client and server data tables that are required to generate the screen or report and the number of views or sections presented as part of the screen or report. Once the complexity is fixed, the number of screens, reports and components are weighted according to value given in Table 4. The object point count is then determined by multiplying the original number of object instances by the weighting factor from the table and summing to obtain total object point count. When component-based development or general software reuse is to be applied the percent of reuse (%reuse) is estimated and the object point is adjusted:

New of Object Points (NOP) = object points * (100 -%reuse)/ 100

To derive an estimate of effort based on computed NOP value, a "productivity rate" is derived and given in the Table 5.

Table 4. Complexity weighting for object types

Object type	Complexity weight		
	Simple	Medium	Difficult
Screen	1	2	3
Reports	2	3	8
3GL Component	-	-	10

Table 5. Productivity rate for object points

Developer's Experience/ Capability	Very low	Low	Nominal	High	Very High
Environment maturity / Capability	Very low	Low	Nominal	High	Very High
PROD	4	7	13	25	50

Productivity rate (PROD) = NOP/ person-month, for different levels of developer experience and development environment maturity. Once the productivity rate has been determined, an estimate of project effort is computed using

Estimated effort = NOP/ PROD

In the formulated effort estimation model, instead of Object Points of COCOMO II, we consider number of screens (front-end forms) to be designed and developed for the project. Each front-end form is assigned Design Complexity level by defining two parameters - Event Complexity (EC) and Interactivity Level (IL) and then effort required to design and develop each front-end form is calculated. Effort required for creation of data tables, reports to be

generated, documentation and networking requirements are considered separately. Brief details of the formulated method [22] by the authors are recapitulated in the following paragraphs for sake of completeness.

4.1. Estimation of the Number of Front-End Forms to be Designed and Developed.

Number of front-end forms (screens) likely to be developed will be estimated based on Software Requirements Specification. Based on design features there will be several events associated with each form. There would be data and message passing from one form to another. The forms will contain several controls and commands such as Command Buttons (CD), Radio Buttons (RB) , Check (CH), Combo (CB), List (LB), Text (TB), Label (LL) boxes etc.

4.2. Form Design Complexity as a function of EC and IL.

4.2.1. Event Complexity (EC)

Event complexity is that part of form design in which the commands and controls are created on the form. To arrive at the event complexity of 4GL forms following logic is adopted: Number of commands and text boxes on a form are considered for deciding the level of Event Complexity. It is assumed that if a form consists up to 10 text boxes, then effort required to create up to 10 TBs is equivalent to the effort required to create one command including its code. For instance if a form consists of 4 commands and 25 textboxes, then the effort required to develop this form will be equal to effort required to create 7 (4+3) commands. Let m is equal to the number of command buttons created on the form. EC level is categorized on a 5-point scale Very Simple (VS), Simple (S), Average (A), Moderate (M) and Complex (C). The criterion adopted for categorizing the EC level of a form is VS if $m < 3$; S if $3 \leq m \leq 5$; A if $6 \leq m \leq 8$; M if $9 \leq m \leq 11$; C if $m > 11$.

4.2.2. Interactivity Level (IL)

The Interactivity Level (IL) of a front-end form is the amount of interactions the form will have with other forms. For example if we select a radio button created on the form, the form may interact with number of forms equal to the number of options given for the radio button. Let n be total number of controls, which includes commands, radio buttons, list boxes, combo boxes created on the form. Again IL is categorized into five levels i.e. VS, S, A, M and Extensive (E). The criterion for choosing interactivity level of a form is VS if $n < 3$; S if $3 \leq n \leq 5$; A if $6 \leq n \leq 8$; M if $9 \leq n \leq 11$; E if $n > 11$.

In Table 6, sample data is given to show how a front-end form is classified in terms of its Event Complexity (EC) and Interactivity Level (IL).

Table 6. Number of controls and commands on a front-end form and its EC and IL level

Form	CD	TB	m	RB	LB	CB	n	EC Level	IL
1	1	1	2		-	-	2	VS	VS
2	4	10	5	-	1	-	6	S	A
3	3	45	8	-	-	1	9	A	M
4	9	-	9	-	2	1	12	M	E
5	16	12	18	1	2	1	22	C	E

For the case study given in [22], total number of front-end forms is 46. Based on the above procedure number of forms with different levels of EC and IL is calculated and given in Table 7.

Table 7. Number of Forms with different EC and IL Levels

	IL					
EC	VS	S	A	M	E	Total
VS	2	3	0	1	0	6
S	0	6	2	1	0	9
A	0	0	4	3	0	7
M	0	0	0	9	1	10
C	0	0	0	0	14	14
Total	2	9	6	14	15	46

4.3. Derivation of Effort Unit Matrix.

To deign and develop a front-end form we need to have discussions within the team about design, controls and commands to be put on the form, setting properties to each control, coding and validating the calculations made and interconnecting with other forms, database and reports. Many times there will be modifications to many controls. Beautification of the form to attract the user's attention is also required to be done with matching colors for commands and controls. Second author was Project Head for more than 25 software development projects related to Defence systems. He was leading software projects right from FORTRAN days. After 1992, his teams started using Visual Basic and MSACCESS for developing software. Based on his experience in leading many projects, initially we assumed that the effort required for designing and developing a front-end form with VS levels for both EC and IL is equal to one effort unit which is equal to the work done in 8 hours (one person day). This form with VS-EC and VS-IL is taken as standard form. The effort required for developing a form with a VS event complexity and extensive (E) interactivity level is assumed to be 5 times (because 5 level of complexity) the effort compared to the standard form. We met more than fifteen experts /project heads to verify and validate the above assumption. Most of the experts were of the view that the assumption is reasonably fine but suggested validating the assumption after working out a few case studies. After estimating effort for four case studies [22] and making extrapolation and interpolation, effort unit matrix was refined and is given in the Table 8.

Table 8. Effort unit matrix of forms

IL	**EC**				
	VS	S	A	M	C
VS	2.5	5.0	7.5	10	12.5
S	5.0	7.5	10	12.5	15
A	7.5	10	12.5	15	17.5
M	10	12.5	15	17.5	20.
E	12.5	15	17.5	20	22.5

The effort unit matrix for 4GL forms is a 5 X 5 symmetric matrix, in which a row depicts a situation of increasing effort as EC increases. The column depicts a situation for a form of a specified event complexity level, increasing effort as IL increases. The first element in the

above matrix indicates that the effort required for designing, developing and integrating (with other forms, reports and database) a form with VS-IL and VS-EC is 2.5 effort units which is equal to the work done in 20 hours.

4.4. Calculation of Effort Required for Form Design and Development

Total Effort Units (TEU) required for form design, development and integration can then be calculated as follows:

$$\text{Forms\_ effort (in TEU)} = \Sigma^{5}_{i,j=1} a_{ij} * b_{ji},$$

where, a_{ij} is the matrix element in Table 7 (Number of forms) and b_{ij} is the Elements in the effort unit matrix (Table 8). For the case study discussed in [22], total effort units required for designing and developing 46 forms (Forms_ effort) was 690 TEU.

4.5. Calculation of Effort for Creation of Database

Database is one of the important components of the software project. The database is usually created using MySQL/ MSACCESS/ Oracle. The effort required to create a database (Db_effort) is estimated on the basis of number of tables which includes design of tables along with the effort for entering the values for the tables. A few data tables will get filled after the software is executed. It is assumed that again based on experience, one effort unit will be required to create a data table including design and entries.

4.6. Calculation of Effort for Reports Generation

The outputs of the software are in the form of reports. The quality of reports generated reflects the quality of the developed software in term of user friendliness for analysis. Reports generated should not create any doubt in the minds of the user. The results must be tabulated in such way that analyses can be made easily. In addition to reports, facility to view bar/ pie charts or x-y graphs must be provided. We treat graphs also as reports. It is difficult to analyze the results from large reports, but a graph/ chart is sufficient to draw a number of conclusions. The effort required to generate a report (Rp_effort) is nothing but the effort required to design a report and interlink with the database and integrate with appropriate forms associated with the report. Again based on the experience of the project teams, it is assumed that each report generation requires about two effort units.

4.7. Calculation of Hours Required for Networking

Initially, software required to work on local networks will be developed and tested on a standalone mode at the developer's site. Once the user is satisfied with functioning of the software then the software will be installed on network and will be tested for its functionality. Actual hours put in for completing networking at the user's site was taken as networking effort (NW_effort).

4.8. Calculation of Effort for Documentation

Software is basically an invisible product. Though the code can be studied, in most cases it cannot be easily read/ understood. This becomes a major difficulty in the maintenance of software. Therefore, considerable documentation is required for all software products.

Documentation requires considerable effort and skilled manpower. Knowledge of software and working capabilities are essential to estimate the total documentation effort and to find its cost. The documents commonly produced for most projects are Software project definition, Software project plan, Feasibility study, Software requirement specifications including minutes of review meetings and discussions held, Risk management plan, Software quality assurance plans, Test plan and specifications, User manuals, Installation manual, Instructors manual, Software design documents, Software maintenance manuals, Change/Version control, Acceptance report etc. Documentation effort (Doc_effort) is usually based number of pages of documents produced per day. Based on experience and the discussion with various project teams and also the data collected [18] from software engineers trained at NTTF, on an average a person can produce 10 pages of document in a day.

4.9. Total Effort for Software Project.

Therefore, total effort required for software project

= Forms_effort + Db_effort + Rp_effort + NW_effort + Doc_effort

For the case study given in [22], there were 24 tables in database and hence Db_effort was 24 TEU, Rp_effort for 15 reports was 30 TEU, NW_ effort was 20 TEU (initially developed on standalone mode and ported to LAN at user's site) and Doc_effort for 333 pages of document was 34 TEU. Hence total effort estimated was 798 TEU (6369 person hours or 64 PM). From project data the actual effort was 77 months and hence the ratio of actual effort to the estimated effort is 1.2. The estimated effort value is 20% off from the actual effort and therefore the estimate obtained by AMN Yogi and Mala V Patil method can be called fair estimate.

Now, we estimate the effort for projects whose data is given in Section 2.2 from the parametric models given Section 3.0 and AMN Yogi and Mala V Patil method. AMN Yogi and Mala V Patil method [17, 18 and 22] for effort estimation for software projects requires information as per the questionnaire given in Appendix A. The results are given in Table 9. It can be observed that the values estimated by AMN Yogi and Mala V Patil method are in reasonable agreement with the actual effort values.

Table 9. Effort Estimation from different Methods

Methods+→ PT No.↓	Actual effort	AMNYogi & Mala.V. Patil	Bailey-Basili	Barry-Boehm	COCOMO-II	Doty	Waslton-Felix
1	10.5	12	6.	4	5.5	7.1	5.4
2	16	13	7	5	5.0	9.2	7.0
3	12	10	7	5	4.05	9.2	7.0
4	12	11	7	7	6.0	12.0	9.2
5	16	15	8	8	6.9	15.3	11.8
6	14	12	8	9	4.9	16.6	12.8
7	16	12	8	10	4.6	18.4	14.1
8	13.5	16	10	16	5.4	27.7	21.2
9	20	17	10.	17	7.5	30.7	23.6
10	14	13	10.	17	4.9	30.7	23.6
11	16	13	11.	21	4.3	36.9	28.3
12	14	11	13	25	12.1	43.0	33.1

5. PERFORMANCE OF THE MODELS

There are various measures for assessing the accuracy of the models: Root Mean Square Error (RMSE), Mean Relative Error (MRE) and so on. RMSE is calculated based on a number of actual data observed and estimated by the models; it is derived from basic magnitude of relative error which is defined as

$$\text{RMSE} = \sqrt{(1/N)\sum(E_a - E_i)^2}\ ,\ i=1,N,$$

where, E_a is the actual effort and E_i is the effort estimated and N is the number of case studies or number of observations. Mean Relative Error (MRE) is the main performance measure. It is estimated from relative error, which is the relative size of the difference between the actual and estimated value. MRE is the percentage of the absolute values of the relative errors averaged over the number of observations. Here number of observations is the number of case studies worked out. Hence,

$$\text{MRE} = 100*(1/N)\ \sum(|E_a - E_i|)/E_i$$

In Table 10 MRE and RMSE for all the six methods are tabulated. It is found that MRE and RMSE of Yogi and Patil method are the minimum.

Table 10. Performance analysis of methods

Method	AMNYogi & Mala.V.Patil	Bailey-Basili	Barry-Boehm	COCOMO-II	Doty	Waslton-Felix
MRE	21.26485	86.60706	95.61854	192.9971	44.99181	56.76339
RMSE	2.44097	6.298589	6.633388	9.012003	12.79021	8.390931

6. ANALYSES OF CRITICAL FACTORS

The critical factors are those which have direct bearing on the cost and schedule of a software project and are known as risk drivers. A methodical analysis of these factors is essential for project success. There are hundreds of such factors that affect project development. In literature the classification of these factors is made depending on their impact on success or failure. Impact is being measured as catastrophic, critical, marginal and negligible effect on the cost and schedule of the project. Experts classify risks in various ways. For example the US Air Force [25] methodology which gives excellent guidelines for software risk identification and abatement classifies risk drivers (factors) into Performance, Cost, Support and Schedule risks. These risks viz. Performance, Cost, Support and Schedule are the degrees of uncertainty in meeting the requirements, maintaining the budget, enhancing and delivering in time respectively. Probability of each of risk drivers will be quantified based on the previous experience or by judgment. Waman [34] classifies the risk drivers into Technology risks – risk of chosen hardware and software and network communication technology becoming obsolete, People risks – concerns with capability, skill, maturity etc, Organizational risks – environment at developer as well as the users organizations, Requirements risks – clarity, completeness and confirmation (volatility) of requirements by the end user, Estimation risks – judgment errors in estimation of size, effort and schedule of the project. In addition critical factors are also categorized as known, predictable and unpredictable risks. Susan. A. Sherer [30] categorized

critical factors as having Technical, Organizational and Environment Dimensions. Technical risk results from uncertainty in tasks and procedures. The organizational risk results from poor communication and poor organizational structure, environmental dimension results from changing environment and problems with external relationships For example if we take personnel as a risk driver, then personnel lacking necessary technical skill will be categorized as having technical dimension, interpersonal relationship hindering the development will be categorized as having Organizational dimension and if the personnel cannot locate or effectively manage external software development then it is categorized in Environmental dimension.

All these classifications and categorization of critical risk factors have been done to analyze their impact on the software development process and to mitigate them as much as possible. The questionnaire given in Appendix 'A' specifically requests team members to indicate or list out the problems faced by the team leader and team members at serial number 20. Inherent risks faced, like inexperience, lack of guidance from superiors, non-availability of computers and library facilities etc are required to be filled at Serial number 21. There were many critical factors mentioned by the teams. After analyzing the data, a few important critical factors are given in the following lines.

1. Project team members had less expertise in Java. So they learned and practiced with the help of SUN-ORACLE and java DOC websites to mitigate this problem. Expertise of team members is Performance driver as per [25], Technology dimension as per [30] and it is People risk as per [34].

2. Many team members were not aware of executing server software. They found difficulty in installation of software. The problems were tackled by browsing internet and consulting system administrator. Lack of training is again a Performance driver as per [25], Technology dimension as per [30] and categorized as People risk as per [34].

3. It requires capabilities and skills to understand design. Hence teams found difficulties in designing the logic. But solved the problem with the help of seniors. Lack of skill is placed under Support category in [25], Organizational dimension in [30] and People risk in [34].

4. At the beginning, team members found that there were network and LAN problems, and non-availability of the required software platform. There was incompatibility of operating system to run the software. Management came to the rescue of the teams and provided the resources. Lack of required resources is categorized as Schedule constraint in [25], Organizational dimension in [30] and Organizational risk in [34].

5. More than 50% of the project members were lagging in skills required for the project, because of inexperience. Team members found difficulty in coding to develop a project. They found difficulty in understanding the requirements of the projects. They were not sure about the concepts. With the help of seniors, the problems were taken care of. These factors including understanding requirements are associated with Cost drivers in [25], Organizational dimension in [30] and People risk in [34]

7. CONCLUSIONS AND FUTURE WORK

Accurate software data collection is one of many strategies for improving systematic software development process which involves effort estimation and analysis of critical/ risk factors. The methodology adopted for data collection is distribution of a suitable questionnaire to the project teams to extract data. It is very important that data must be updated and corrected as and when it changes. Noting the reasons will help in updating software estimates. This will also help project managers to make appropriate decisions and software engineers in their endeavor to improve the success rate of software projects. The data collected by the authors by distributing a questionnaire to the project teams has been utilized for validation of the software estimation model formulated by the authors for 4GL environment. Comparison of results obtained by the other popular methods available in literature is made. MRE and RMSE estimated for all the methods shows that the formulated method gives results to a reasonable accuracy when compared with the actual development effort. Software engineers are required to evaluate and analyze the environment under which project is taken and use suitable estimation models for decision making. Accurate estimates alone will not be enough for success of the project. It is required to analyze various critical factors that affect the cost and schedule. In literature we find that experts have classified and categorized various critical factors based on the impact on the project success. Taxonomy of risk drivers will help project managers to understand their impact and to take appropriate measures to mitigate them as early as possible. We have utilized the same questionnaire to bring out some of the critical/risk drivers affecting the success of the project...

In our future work, we will be analyzing the ways in which the formulated method can be improved for its results by including more number of parameters. It may be required to improve the questionnaire so that teams will be able to give more useful information.

ACKNOWLEDGEMENTS

Authors express their thanks and gratitude to the team leaders and team members for providing unclassified data about their projects.

REFERENCES

[1] Barry W .Boehm, (1981) *Software Engineering Economics*, Prentice -Hall, Inc., Eaglewood Cliffs, New Jersey.

[2] Barry W. Boehm, (1988) "A Spiral Model of Software Development and Enhancement", *Computer,* Vol. 21, No. 5, pp. 61-72.

[3] Barry W. Boehm, (1989) "Software Risk Management, tutorial"*, IEEE CS Press.*

[4] Barry W. Boehm, Bradford Clark, B, Ellis Horowitz, Chris Westland, Ray Madachy, and Richard Selby, (1995) "Cost Models for Future Software Life Cycle Processes: COCOMO 2.0", *Annals of software Engineering, Special Volume on Software Process and Product Measurement,* pp. 1-35.

[5] Barry W. Boehm, (1996) "Anchoring the software process", *IEEE software,* Vol.13, No.4, pp.73-82.

[6] Barry W. Boehm, (1991) "Software Risk Management: Principles and Practices", *IEEE Software,* Vol. 8, No. 1, pp. 32-41.

[7] Christopher G. Jones, Glen L. Gray, Anna H. gold and David W. Miller, (2010) "Strategies for Improving Systems Development Project Success", *Issues in Information Systems.* Vol. 9, No. 1, pp. 164-173.

[8] Clyde G. Chittister and Y. Y. Haimes, (1996) "System Integration via Software Risk Management", *IEEE Trans on Systems, Man and Cybernetics,* Vol. 26, No. 5, pp. 521-532.

[9] Daniel V. Ferens, (1999) "The Conundrum of Software Estimation Models", *IEEE AES Systems Magazine*, pp. 23-29.

[10] A. Gemmer, (1997) "Risk Management Moving beyond process", *Computer,* Vol. 30, No. 5, pp. 33-41.

[11] H,Hecht, (2003) *Systems Reliability and Failure Prevention*, Artech House.

[12] Janne Ropponen and Kalle Lyytinen , (2000) "Components of Software Developments Risk: How to Address Them? A Project Manager Survey", *IEEE Trans. on Software Engineering* ,Vol. 26, No. 2, pp. 98-112.

[13] Jingyue Li, Reidar Conradi, Odd Petter N. Slyngstad, Marco Torchiano, Maurizio Morisio and Christian Buns, (2008) " A State-of-the Practice Survey of Risk Management in Development with Off-the-Shelf Software Components", *IEEE Trans. On Software Engineering,* Vol. 34, No.2, pp.271-286.

[14] M. Jorgensen and K. Molokken, (2006) "How Large are Software Cost Overruns? A Review of the
1994 Chaos Report", *Information and Software Technology,* Vol. 48, No. 8, pp. 297-301.

[15] J. Laurenz Eveleens and Chris Verhoef, (2010) "The Rise and Fall of the Chaos Report Figures", *IEEE Software,* Vol. 27, No. 1, pp. 30-36.

.

[16] Linda Wallace and Mark Keil,(2004) "Software Project Risks and their effects on Outcomes", *Comm. of the ACM*, Vol. 47, No.4, pp. 68-73.

[17] Mala V Patil and AM Nageswara Yogi, (2010) "Software Development Projects by Engineering Students – Analyses of Difficulties and Effort including Risk Elements", *International Journal of Computer Applications in Engineering, Technology and Sciences,* Vol. 2, No. 2, pp. 132-137.

[18] Mala V Patil and AM Nageswara Yogi, (2010) "Effort Estimation and Risk Analyses for Software
Projects by Data Analyses of Developed Projects", *ACS - International Journal on Computational Intelligence,* Vol. 1, No. 2, pp. 43 -52.

[19] Mark Keil, Paul E. Cule, Kalle Lyytinen, and Roy C. Schmidt, (1998) "A Framework for Identifying Software Project Risks", *Communication of the ACM*, Vol. 4, No. 11, pp. 76-83.

[20] Marvin J. Carr, Suresh L. Konda, Ira Monarch, F. Carol Ulrich and Clay F. Walker, (1993) "Taxonomy-Based Risk Identification", *Technical Report No. CMU/SEI-93-TR-6, ESC-TR-93-183.*

[21] AM Nageswara Yogi, (2006) "A model for Life Cycle Cost Estimation for Defence Equipment", *Proceeding of International Conference on Trends in Product Life Cycle Modeling, Simulation and Synthesis PLMSS-2006*, pp. 415-423.

[22] AM Nageswara Yogi, Mala V Patil, (2009) "Software Effort Estimation Models and Performance Analysis with Case Studies," *International Journal of Computer Applications in Engineering, Technology and Sciences,* Vol. 1, No. 2, pp. 558-565.

[23] Rasmita Dash and Rajashree Dash, (2010) "Risk Assessment Techniques for Software Development", *European Journal of Scientific Research*, Vol. 42, No. 4, pp. 615-622.

[24] Robert L. Glass, (2001) "Frequently Forgotten Fundamental Facts about Software Engineering", *IEEE software*, Vol. 18, No. 3, pp. 110-112.

[25] Roger S. Pressman, (2005) A *Manager's Guide To Software Engineering,* Tata McGraw-Hill.

[26] Roger S. Pressman, (2010) *Software Engineering A Practitioner's Approach,* Seventh Edition, Tata McGraw-Hill.

[27] Dr Roger Sapsford, (2006) *Data Collection and Analysis,* Amazon.

[28] I. Sommerville, (2004) S*oftware Engineering*, Seventh Ed, Addison-Wesley.

[29] Standish group website, (2010) *"*< http://www.standishgroup.com>", Accessed on 15-02-2011.

[30] Susan A. Sherer, (1995) "The Three Dimensions of Software Risk: Technical, Organizational and Environmental", *Proceeding of the 28th Annual Hawaii InternationalConference on System Sciences, IEEE*, pp 369-378.

.

[31] Tony Moynihan, (1997) "How Experienced Project Managers Assess Risk", *IEEE Software,* Vol. 14, No. 3, pp. 35-41.

[32] Victor R, Basili and David M. Weiss, (1984) "A Methodology for Collecting Valid Software Engineering Data", *IEEE Trans on software Engineering,* Vol. SE-10, No. 6, pp. 728-738.

[33] C. E. Walston and C. P. Felix, (1977) "A method of programming measurement and estimation", *IBM Systems Journal,* Vol. 16, No.1 pp. 54-73.

[34] Waman. S. Jawadekar, (2009) Software *Engineering Principles and Practice*, Tata McGraw-Hill Pvt. Ltd.

[35] www.developer.com/mgmt/article.php/1463281.

Appendix A:

Questionnaire: Data for the formulated effort estimation model

Part I (General)

1. Name (optional):
2. Organization:

3. Project Title:
4. Application Software required:
5. Computer language with Backend database:
6. Hardware requirements:
7. Size of the project team?
8. Qualification (BE/MCA/M.Tech)
9. Experience of team members(Low/Normal/High):
10. Type of Numerical algorithms used if any:
11. Number of lines of code:
12. Estimated project duration starting date: ending date : or in months
13. Actual duration for completion of the project in months:

PART II (Specific to the software design and development)

Number of front-end forms designed in the software project:

Kindly read the following paragraph and fill the Table1

The authors have formulated a method to estimate the effort required for a software project based on the complexity of the front-end forms. To decide complexity level for designing and developing a front-end form, we define Event Complexity (EC) and Interactivity Levels (IL) for each form. EC of a form is classified on a 5-point scale as Very Simple (VS), Simple (S), Average (A), Moderate (M) and Complex (C). EC depends on a number m which is equal to sum of number of commands (CD) created on the form + the integral part of (number of text boxes (TB)/10) + 1. Event complexity of a form is VS if $m < 3$; S if $3 \leq m \leq 5$; A if $6 \leq m \leq 8$; M if $9 \leq m \leq 11$; C if $m > 11$. Interactivity level (IL) of a VB form is again classified on a 5-point scale as VS, S, A, M and Extensive (E). IL depends on n and n is the sum of number of Radio Buttons (RB), List Boxes (LB), Combo Boxes (CB) and m. Interactivity level of a form is VS if $n < 3$; S if $3 \leq n \leq 5$; A if $6 \leq n \leq 8$; M if $9 \leq n \leq 11$; E if $n > 11$.

Table1. Number of forms with different EC and IL

	IL					
EC	VS	S	A	M	E	Total
VS						
S						
A						
M						
C						
Total						

14. Number of data tables created in the back-end database in the project:
15. Average number of fields per record in each of the table if possible.
16. Number of hours for networking with respect to the project (0 if no networking):
17. Types of documents prepared:
18. Total number of pages documented:
19. Number of hours taken for 10 page documentation.
20. List out problems faced and how you tackled to solve the problems (Use separate sheet):

21. Inherent risk faced, like inexperience, guidance from superiors, non-availability of computers and library facilities etc.
22. Place and Date:

Authors

Mrs. Mala. V. Patil is a research scholar of Anna University, Coimbatore, INDIA. She is working as a Professor and Head of the Department of Information Science, Rajiv Gandhi Institute of Technology Bangalore. She has 17 years of teaching experience. She has published three research papers in international journals. She has presented 5 research papers in National and International Level.
Her mail address is malapatil2002@yahoo.co.in

Dr. AM Nageswara Yogi obtained Ph. D from Indian Institute of Science, Bangalore and presently working as Scientist and Head Knowledge Centre at Aeronautical Development Establishment, Defence R & D Organization (DRDO), Bangalore. He has worked in the areas of Modeling and Simulation, Weapon Systems Analysis, Defence Perspective Plans, Software Development etc. He has also conducted many Continuing Education Programs for the Scientists, Technical Officers and Staff on Computer Application, Software Engineering, Optimization Techniques, and Numerical methods. He has published and presented more than 24 papers in journals/conferences and workshops etc. He also prepared Technical Documents for more than 25 studies related to defence related projects.
His mail address is amnyogi_jalaja@yahoo.co.in