ABSTRACT

REDUCING THE RISK OF SOFTWARE COST ESTIMATION

by
Shixian Yang

Inaccurate cost estimation is a well-known problem in software development. The common cost estimation models are point estimates that are unable to quantify uncertainties. Furthermore, it is difficult to calibrate the uncertainties in cost estimation due to the lack of information. The purpose of this thesis is to prove that probability techniques could be synthesized into COCOMO (Constructive Cost Model) to quantify uncertainties. Another aim is to find out how to get more insight on reducing the risk of cost estimation. In this thesis, some historical data is presented to show the variance in factors of COCOMO. Monte Carlo simulation method is also introduced into COCOMO to quantify the uncertainties. Finally, a “What-if” study is facilitated to find the potential factor changes to affect the result of simulation. The result of the study reveals that process maturity has more influence than productivity on reducing variance of estimation. It indicates that synthesizing Monte Carlo simulation and “What-if” studies into COCOMO could produce insightful information to reduce the risk of software cost estimation.

REDUCING THE RISK OF SOFTWARE COST ESTIMATION

by
Shixian Yang

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Software Engineering

Department of Computer Science

May 2012

APPROVAL PAGE

REDUCING THE RISK OF SOFTWARE COST ESTIMATION

Shixian Yang

Mr. Larry Bernstein, Thesis Co-Advisor Date
Adjunct Professor of Software Engineering, NJIT

Dr. Narain Gehani, Dean, Thesis Co-Advisor Date
Dean of College of Computing Sciences, NJIT

Dr. Ali Mili, Committee Member Date
Professor of Computer Science, NJIT

BIOGRAPHICAL SKETCH

Author:	Shixian Yang
Degree:	Master of Science
Date:	May 2012

Undergraduate and Graduate Education:

- Master of Science in Software Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2012

- Bachelor of Engineering in Software Engineering,
  China University of Geosciences, Wuhan, P. R. China, 2007

Major: Software Engineering

I dedicate this research work to Professor Larry Bernstein who never failed to teach and guide me, to my parents who supported me to finish my degree, to my friends who helped me to complete this project, and most of all to the Almighty God who gives me strength and good health while doing this.

ACKNOWLEDGMENT

This thesis could not have been written without the help of Professor Larry Bernstein, who not only served as an advisor but also challenged me throughout my academic program. He guided me through the dissertation process, never accepting less than my best efforts. I would like to thank my co-advisor Dr. Narain Gehani for his help to my study. Deepest gratitude is also due to the member of the supervisory committee Prof. Ali Mili, without his knowledge and assistance this study would not have been successful.

I also would like to thank Hao Wang from Beijing University of Posts and Telecommunications. He provided me very important data of investigation about software productivity with its influencing factors.

I am especially grateful to MathWave Technologies and Palisade Corporation. They provided the necessary software (EasyFit and @Risk5.7) to my study.

.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

Table **Page**

LIST OF TABLES
(Continued)

LIST OF FIGURES

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Purpose

The risk in estimating software engineering costs, schedule and reliability has been a big problem in the software industry in recent years. Many people are seeking a way to find variance in estimation. The purpose of the thesis is to find an effective method to reduce the variance. The first step is to determine the feasibility of computing the variance for statistics used in software estimation. If it is proved that computing the variance for the variety of random variables and their distributions is feasible, then it can facilitate "What-if" studies to explore potential resource changes to improve the likelihood of meeting software project commitments. For example, it can be determined how much computed variance may be reduced from a certain investment on staff training. Then the manager could make a decision that whether or not to invest more resources on staff training or not. Therefore, variance analysis can help provide more information if it is introduced to the traditional point estimates. Overall, the thesis will utilize Monte Carlo simulation with COCOMO to calculate variance of cost estimation and facilitate "what-if" studies to reduce risk.

1.2 Scope

A. Estimation theories in Boehm's approach to Software Estimation

Software cost estimation is very complicated because it involves many factors and uncertainties. Barry Boehm introduces an approach named Constructive Cost Model which relates software development efforts to source lines of code. Combined with other cost drivers, COCOMO becomes a very pragmatic tool for software estimation. Boehm's basic COCOMO is introduced to the thesis first. Though it is an early model from the 1980s and the data is outdated compare to its late models, it is necessary to understand because it shows the basic idea of Boehm's theory in COCOMO. Meanwhile, it will include the study of Boehm's risk analysis, as it is very important to find the upper bound and lower bound of variance. The thesis will be mainly based on the data from COCOMO II. COCOMO II has made many improvements from COCOMO 81 which reflects the latest study from Boehm's work, and it is the core theory used in my thesis. To summarize, in Boehm's theory, size, productivity and scale factors determine how much effort and time a software project will cost. Of course productivity is quite complicated because it is also affected by many other factors. However, the basic estimation process is very clear: size the project with Function Point or other methods, estimate the productivity with experience or historical data, and combine with the cost drivers in final calculation.

B. Software Project Management

Software cost estimation is the most important and difficult part of Software Project Management. Because cost estimation is involved in the entire process of Software Project Management, it is necessary to understand the relationship between estimation and Software Project Management.

C. Monte Carlo simulation and implementation of its tools

Monte Carlo simulation can help people to find the variance of cost estimation. The method has been widely used in the financial industry, but it is merely used in software cost estimation. This thesis will discuss its feasibility in software estimation and implementation in sample issues.

D. “What-if” studies on reducing risk of cost estimation

There are many potential factors that could affect the risk of cost estimation but their effects are not quantifiable. A “What-if” study can produce a quantified result of factor changes to help people reduce the risk.

CHAPTER 2

OVERALL DESCRIPTION

2.1 Study Perspective

> “Being able to accurately estimate software deliverables in terms of schedule, scope, and quality is a prized objective for software development teams and management. Any company that relies on software to help drive revenue, either directly or indirectly, needs to be able to trust the estimation capability of its software development group. Business leaders directly correlate revenue projections to software features, so delivering on time with committed scope and quality will provide better budget projections to the company and its stakeholders. I’ve been involved with large software development companies whose business departments do not trust the development organizations, and it was not pretty.”
>
> by Neil Fox, The Two Metrics that Matter [1]

People need an accurate estimation method which can be trusted, such as COCOMO. Because COCOMO involves many factors in estimation, finding the relationship between these factors and COCOMO is very important to improve estimation procedure in software industry. Furthermore, introducing a Monte Carlo simulation to software estimation is significant in risk analysis. If it is feasible to implement Monte Carlo simulation in COCOMO, it may provide a solution to reduce estimation risk effectively.

2.2 Study Procedures

1. Study project management and estimation.
Study the content of software project management and find out the relationship between project management and estimation.

2. Study Boehm’s COCOMO.
Study Barry Boehm’s COCOMO and his later COCOMO II to learn the usage of COCOMO and understand all the cost drivers in COCOMO

3. Discuss the risks from COCOMO.
Find out the uncertainties of all factors used in COCOMO, including sizing, productivity, and cost drivers.

4. Find the tool to calibrate risks.
Search the tools which can run a Monte Carlo simulation on COCOMO and learn how to use them.

5. Implement the tool on COCOMO.
Design an appropriate plan according to the tool chosen to run the Monte Carlo simulation to get the result.

6. Analyze with “what-if” studies and conclusions.
Based on the result from the simulation, establish the relationship between the factors and COCOMO. Then make the conclusion.

2.3 Assumptions and Dependencies

1. It is asserted that the COCOMO is correct based on its current data.

2. It is asserted that all the probability software tools used in this thesis are working correctly.

CHAPTER 3

PROJECT MANAGEMENT

3.1 Introduction

In the past few years, more and more people focus on project management in the software industry due to the high failure rate of software project. There is a common sense that the failure rate of software projects is between 40% - 70% and it varies on type, scale and many other features of the project.

The Robbin-Gioia Survey (2001) reported that 51% viewed their ERP implementation as unsuccessful. Another report from The Conference Board Survey (2001) which interviewed executives at 117 companies that attempt ERP implementations gave a conclusion that 58% were “somewhat unsatisfied”, 8% were unhappy with what they got and 40% percent of projects failed to achieve within one year of going live [2].

What are the reasons for the failures of software projects and how to avoid the failure? It’s a complicated problem. There are factors that could cause a project to fail. The most common factors can be:

“1. Unrealistic or unarticulated project goals

2. Inaccurate estimates of needed resources

3. Badly defined system requirements

4. Poor reporting of the project's status

5. Unmanaged risks

6. Poor communication among customers, developers, and users

7. Use of immature technology
8. Inability to handle the project's complexity
9. Sloppy development practices
10. Poor project management
11. Stakeholder politics
12. Commercial pressure”[3]

There are several factors that relate closely to project management. In James McDonald’s theory, there are four major processes in software project management: plan, organize, monitor and control[4]. These processes are used sequentially, both forward and backward. The planning is the first and the most important process, which includes requirements, architecture and design phases of the software life cycle. It also requires determining how the project will be implemented, tested, deployed and maintained in the planning process. As a result, the planning process will cover the whole process of software project. The purpose of Organizing is to build a high-performance team and deal with the potential conflicts people may face. Monitoring is a process to find the deviations from the project plan, organized team and project status. There are many methods to monitor a project and the most common ways are seen as project meeting and Gantt chart. Controlling is closely related to monitoring because it will respond to the deviation generated by monitoring. For example, if team B is found to be behind of schedule in monitoring process, and then a solution is needed in controlling process such as change plan or team members.

3.2 Risk Management

"Risk always involves uncertainty. The uncertainty is usually because there is some information lurking in the background that we do not know. They uncertainty involved in software development projects can affect us in either a negative way, in which case we will call it a risk, or it can affect our results in a positive way, in which case we will call it an opportunity."[5]

Risk Management is an important part of Project Management. The goal is to help the managers to make right decisions that minimize the risk and maximize the opportunity. Barry Boehm has given such decision rules for complete uncertainty in his book:

"Maximin Rule: Determine the minimum payoff for each alternative. Choose the alternative which maximizes the minimum payoff.

Maximax Rule: Determine the maximum payoff for each alternative. Choose the alternative which maximizes the maximum payoff.

Laplace or Equal-Probability Rule: Assume all of the states of nature are equally likely. Determine the expected value for each alternative, and choose the alternative with the maximum expected value."[6]

How to quantify uncertainties? There are two quantitative techniques that can help with it: simulation and event analysis. The simulation methods such as Monte Carlo and Bayesian Network can help to find the probability of completing a certain task. The event analysis method can identify the risky events that could cause loss of the project, thus people can prepare the plan to deal with the expected loss.

3.3 Value of Estimation in Project Management

Planning is one the most important processes in project management: good planning will lead to a successful project. However, good planning requires accurate estimation on the size of the project, the effort of staff power, cost and schedule. The goal of a good estimation is to provide the cost and schedule to the manager. If the actual cost of a project is out of budget, no matter how complete the product is, the project is regarded as failure because there is no profit. Similarly if the promised deliverables are late to meet its delivering date, it is still regarded as failure because the contract is not fulfilled and the stakeholder may sustain loss. In addition, monitoring and controlling processes are iterative and estimation is required between the processes to find the deviations from the plan. For the same example in 3.1, how to know a solution can fix the problem and make Team B catch up with the schedule? A re-estimation would be preferred in this case. Therefore, estimation is needed throughout the entire project management process.

CHAPTER 4

RISKS IN ESTIMATION

4.1 Estimation and COCOMO

COCOMO (Constructive Cost Model) is an estimating model which presents the equations for calculating the effort and schedule required to develop a software product. Though COCOMO is considered as a successful estimating model, it still involves risks that shouldn’t be ignored.

The Basic COCOMO 1981 consists of effort and schedule equations for organic, semidetached, and embedded modes of software development. (See table 4.1).

Table 4.1 Basic COCOMO Effort and Schedule Equations

Mode	Effort	Schedule
Organic	2.4 (KDSI)$^{1.05}$	2.5 (Staff Month)$^{0.38}$
Semidetached	3.0 (KDSI)$^{1.12}$	2.5 (Staff Month)$^{0.35}$
Embedded	3.6 (KDSI)$^{1.20}$	2.5 (Staff Month)$^{0.32}$

Source: Barry W. Boehm, Software Engineering Economics, p75, Prentice-Hall, INC., 1981[7].

KDSI stands for Thousands of Delivered Source Instructions. It means that the estimated effort will include the work of design, coding, testing, and integration. Though today people are using KLOC instead of KDSI, the meaning is still the same. However, Basic COCOMO is still limited when it is used on a complicated product. Thus 15 cost drivers are introduced to COCOMO 1981. These cost drivers are RELY (Required software reliability), VEXP (Virtual machine experience), LEXP (Programming language experience), ACAP (Analyst capability), AEXP (Applications experience), PCAP (Programmer capability), DATA (Data base size), CPLX (Product complexity), STOR (Main storage constraint), VIRT (Virtual machine volatility), TURN (Computer turnaround time), MODP (Modern programming practices), TOOL (Use of software tools), SCED (Required development schedule).

To better suit for modern software projects, COCOMO II is developed. It updates the data in the model and makes the model more flexible.

“The amount of effort in person-months, PM, is estimated by the formula:

PM_{NS} = A * $Size^{E}$ * $\prod_{i=1}^{n} EMi$ where E = B + 0.01 * $\prod_{j=1}^{5} SFj$

The amount of calendar time, TDEV, it will take to develop the product is estimated by the formula:

$TDEV_{NS}$ = C * $(PM_{NS})^{F}$ where F = D + 0.2 * 0.01 * $\prod_{j=1}^{5} SFj$ = D + 0.2 * (E - B)”[8]

A = 2.94   B = 0.91   C = 3.67   D = 0.28

EM = Effort Multipliers

SF = Scale Factors

Effort Multipliers are inherited from the cost drivers from COCOMO 81 and Scale Factors are new introduced in COCOMO II. The five Scale Factors are Precedentedness (PREC), Development flexibility (FLEX), Architecture/risk resolution (RESL), Team cohesion (TEAM), and Process maturity (PMAT).

4.2 Sizing

Sizing is the most significant factor that affects COCOMO. There are many sizing methods such as Source Lines of Code (SLOC), Function Point, Use Case Point, Object Point, UML Metrics, etc. No matter what sizing method is used, it involves uncertainty.

The point is how accurate it can be.

"A Source Line of Code is generally meant to exclude nondelivered support software such as test drivers. However, if these are developed with the same care as delivered software, with their own review, test plans, documentation, etc., then they should be counted [Boehm 1981, pp.58-59]. The goal is to measure the amount of intellectual work put into program development. …For general source code sizing approaches, such as PERT sizing, expert consensus, analogy, top-down, and bottom-up."[9] Sometimes people even convert used source lines of code to equivalent new code to simplify the estimation process. In COCOMO's reuse model, it is explained:

$AAF = 0.4(DM) + 0.3(CM) + 0.3(IM)$

$ESLOC =ASLOC [AA+AAF (1+0.02(SU) (UNFM))]/100$, for $AAF <= 0.5$

$ESLOC = ASLOC [AA + AAF (SU) (UNFM)]/100$, for $AAF>0.5$

AAF: Adaptation Adjustment Factor

DM: Percentage of Design Modified

CM: Percentage of Code Modified

IM: Percent of Integration Required for Modified Software

ASLOC: Adapted Source Lines of Code

ESLOC: Equivalent Source Lines of Code

AA: Assessment and Assimilation effort

SU: Software Understanding

UNFM: Unfamiliarity

Function Point counts the number of inputs, outputs, files, Inquiries, and interfaces of the application. The weight of each category can be evaluated based on the numbers. Table 4.2 shows the Unadjusted Function Point Complexity Weights.

Table 4.2 UPF Complexity Weights

	Complexity - Weight		
Function Type	Low	Average	High
Internal Logical Files	7	10	15
External interfaces Files	5	7	10
External Inputs	3	4	6
External Outputs	4	5	7
External Inquiries	3	4	6

Source: Barry W. Boehm, Software Cost Estimation with COCOMO II, pp.19, Prentice-Hall, INC., 2000[10]

Unadjusted Function Points can be converted to Adjusted Function Points but it is not required in COCOMO. In COCOMO, it is preferred to relate UFPs to SLOC in the implementation language such as Java, C++, PHP, etc. According to the data from QSM, it will result in significant variation in the number of source statements per function point which refers to uncertainty or risks. (See Table 4.3).

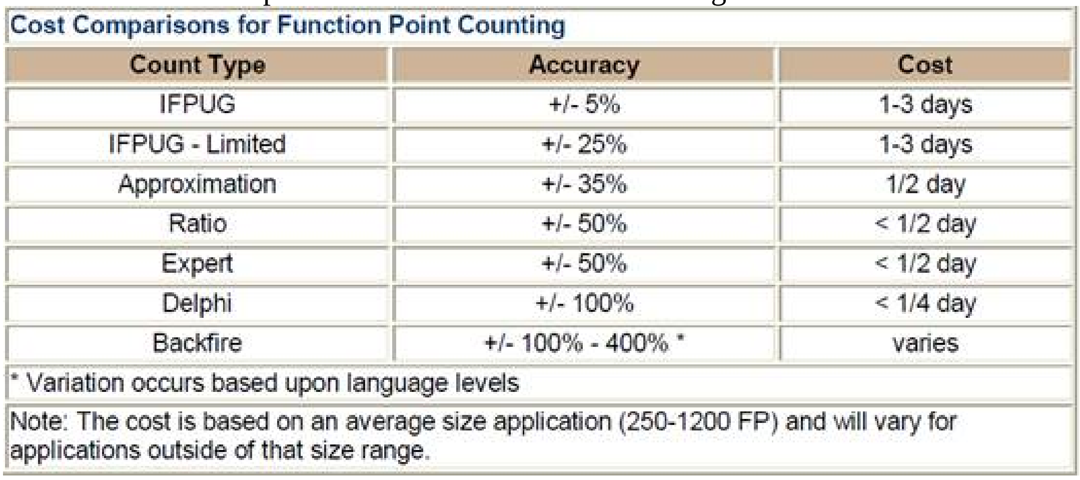
Table 4.3 Function Point Languages Table

Language	QSM SLOC/FP Data				David Consulting Data
	Avg	Median	Low	High	
ABAP (SAP)	18	18	16	20	-
Access *	36	38	15	47	-
Ada	154	-	104	205	-
Advantage	38	38	38	38	-
APS	86	83	20	184	-
ASP *	56	50	32	106	-
Assembler *	209	203	91	320	575 Basic/ 400 Macro
C *	148	107	22	704	225
C++ *	59	53	20	178	C++ *
C# *	58	59	51	66	C# *

Source: http://www.qsm.com/resources/function-point-languages-table[12]

DCG (David Consulting Group) conducted a detailed study during 1999 to determine the cost and accuracy of various counting techniques. In Table 4.4, "the results indicate that it is not always practical or necessary to invest in full counting for a given project or application." [13]

Table 4.4 Cost Comparisons for Function Point Counting



Cost Comparisons for Function Point Counting

Count Type	Accuracy	Cost
IFPUG	+/- 5%	1-3 days
IFPUG - Limited	+/- 25%	1-3 days
Approximation	+/- 35%	1/2 day
Ratio	+/- 50%	< 1/2 day
Expert	+/- 50%	< 1/2 day
Delphi	+/- 100%	< 1/4 day
Backfire	+/- 100% - 400% *	varies

* Variation occurs based upon language levels

Note: The cost is based on an average size application (250-1200 FP) and will vary for applications outside of that size range.

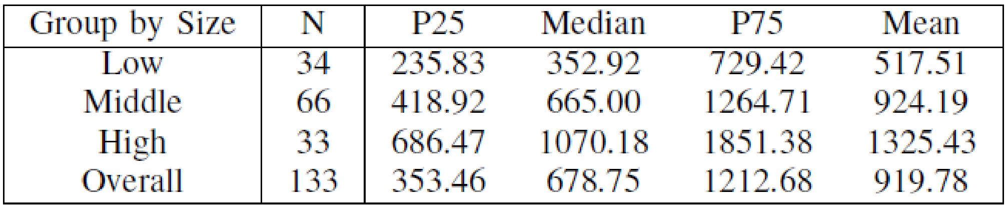
Source: DCG, Software Sizing with Function Points, http://www.davidconsultinggroup.com[12].

4.3 Small Team Productivity

According to the equation of COCOMO, the coefficient represents the value of 1/Productivity (KLOC/PM). In COCOMO II, the value is 2.94. It hints the average productivity is about 340 SLOC per PM. But productivity is influenced by many factors.

First of all, productivity is influenced by project size. In Table 4.5, the projects are classified into four groups by their size. Low is small size and high is large size. “There is an increasing trend for productivity with size increasing. Median of productivity of the high group is more than three times larger than the low group. The median of productivity of the middle group is more than 1.8 times larger than the low group.”[14]

Table 4.5 Basic Summary Data for The CSBSG Data



Group by Size	N	P25	Median	P75	Mean
Low	34	235.83	352.92	729.42	517.51
Middle	66	418.92	665.00	1264.71	924.19
High	33	686.47	1070.18	1851.38	1325.43
Overall	133	353.46	678.75	1212.68	919.78

Source: Hao Wang, Software Productivity Analysis with CSBSG Data Set, 2008[13].

Figure 4.1 also shows an increase in productivity as the project gets bigger. However, it is inappropriate to conclude that bigger projects are more productive. “The effective productivity based on the new functionality and the work involved in redesign, reimplementation, and retest, forming an effective productivity. Some studies leave out such detail." [15] If the projects are not of ultra-size or they are enhancement projects that mainly involve in reuse work, the result will make sense. And there is another reason productivity may increase when project gets bigger: the learning curve effect. The developing team often improves during the project to get more maturity in their working process.

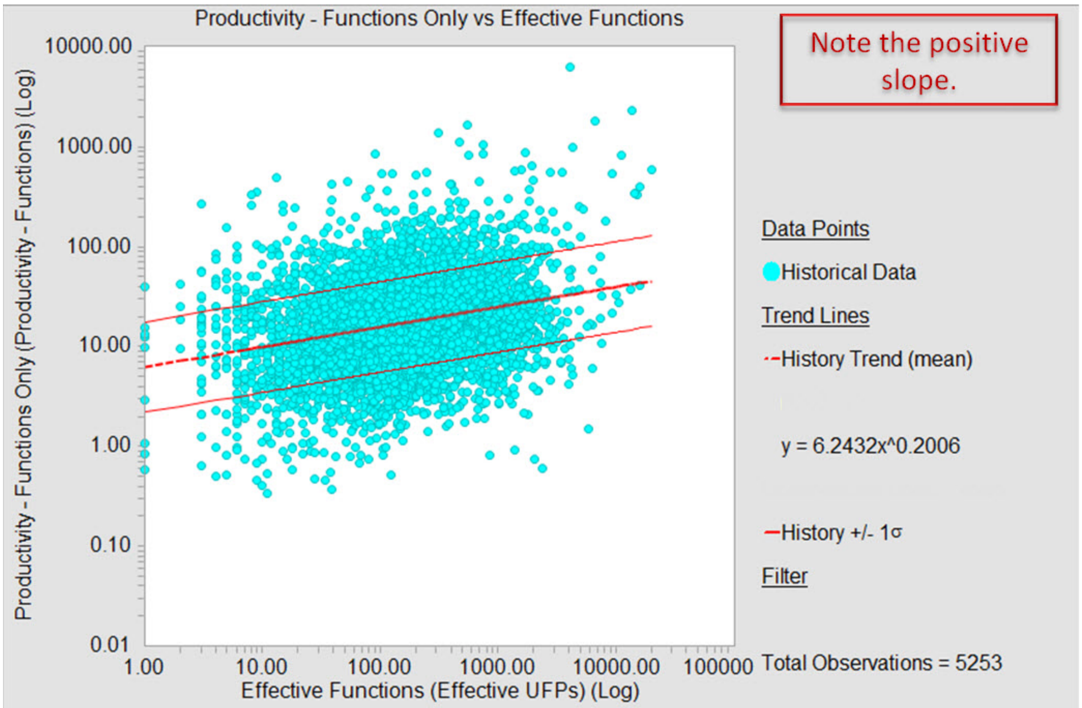


Figure 4.1 Productivity – functions only vs. effective functions.
Source: Dan Galorath, Software Staff Size Still Impacts Productivity: Brooks Law Lives!, 2010[15]

Secondly, productivity is more likely impacted by the staff size. As Brooks Law stated, the staff gets bigger the productivity gets lower in Figure 4.2.

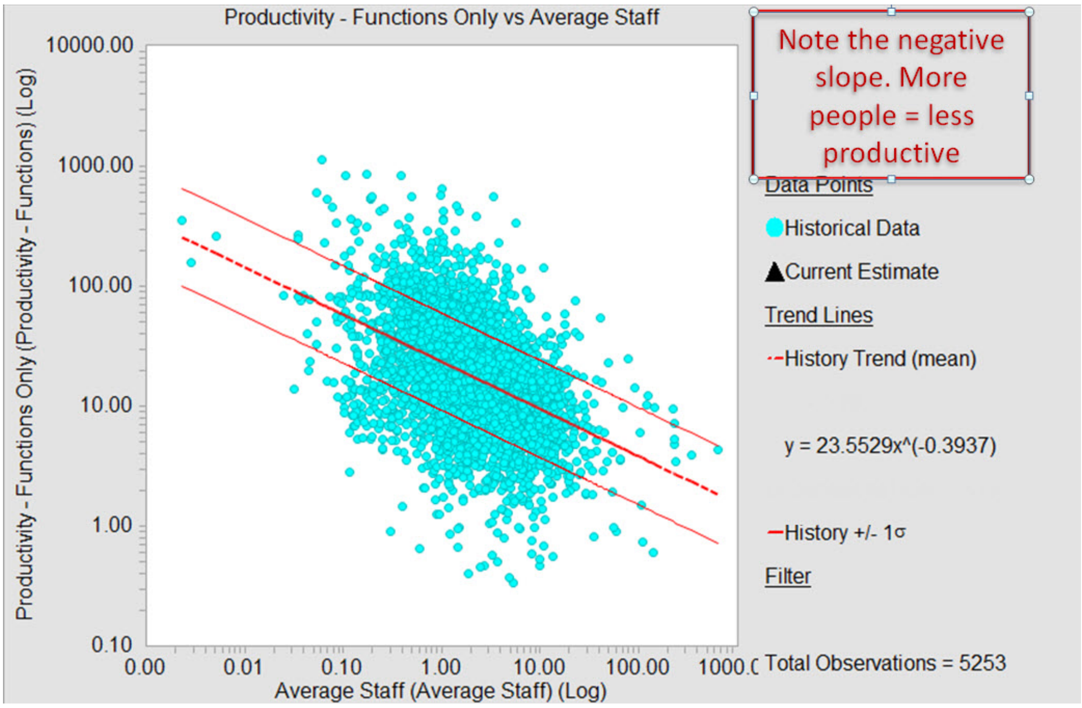


Figure 4.2 Productivity – Functions Only vs. Average Staff
Source: Dan Galorath, Software Staff Size Still Impacts Productivity: Brooks Law Lives!, 2010[15]

Furthermore, productivity is influenced by Project Type and Programming Language. (See Figure 4.3 and 4.4).

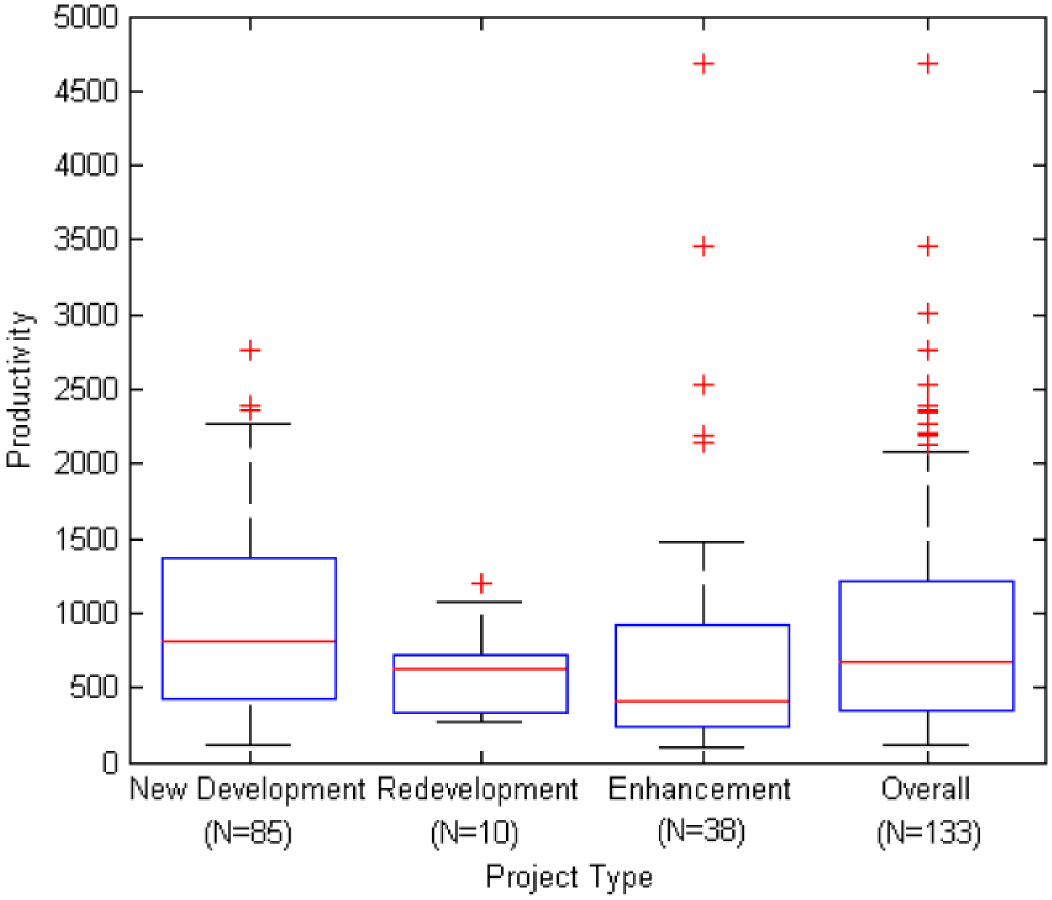


Figure 4.3 Boxplots of productivity for project groups classified by project type. N is the number of samples collected in a project type.
Source: Hao Wang, Software Productivity Analysis with CSBSG Data Set, 2008[14]

In Figure 4.3, New Development projects have larger range of productivity which means New Development projects have more uncertainties in productivity.

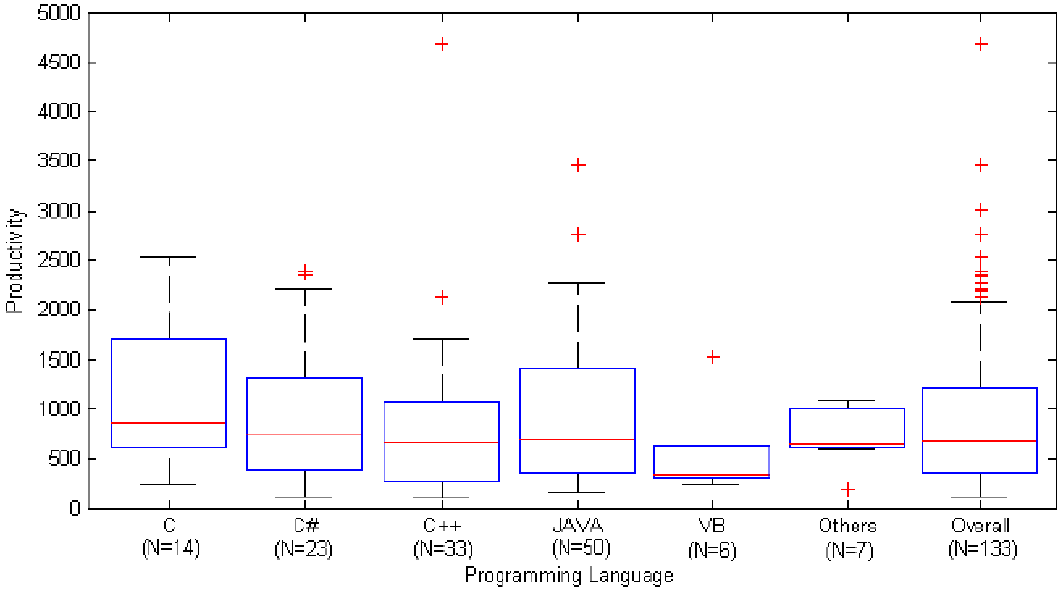


Figure 4.4 Boxplots of productivity for project groups classified by programming language. N is the number of samples collected in a programming language type. Red line is the median value and blue box represents the range of distribution of 50% of data.
Source: Hao Wang, Software Productivity Analysis with CSBSG Data Set, 2008[14]

4.4 Reliability

> "A software product possesses reliability to the extent that it can be expected to perform its intended functions satisfactorily. Quantitatively, we can define software reliability as a probability: the software performs its intended functions satisfactorily over its next run or its next quantum of execution time."[16]

Normally, reliability can be calculated with following steps:

Choose N inputs or input sequences randomly from the probability distribution over the space of possible inputs or input sequences to the software.

Execute the software N runs or a certain execution time with the inputs.

Based on the success criterion to determine the M number of runs resulted in the satisfactory outputs. Meanwhile, collect the elapsed time between failures to calculate Mean Time to Failure.

Calculate the estimator R = M/N.

Theoretically, estimator R should be constant when the code remains unchanged. "However, field experience shows that the software product failure rate often gets smaller with time, even when there are no code changes. This may be due to users learning to avoid the situations that cause failures or their using a small number of features."[17] Apparently when the inputs are not randomly chosen, the result will not show the exact reliability.

To improve reliability, all of the development phases need to be enhanced. But the phase weights are different: product design, 15%; detailed design, 30%; code and unit test, 30%; integration and test, 25%.[18]

According to COCOMO II, the overall effort Multipliers for different Rating Levels of Reliability is

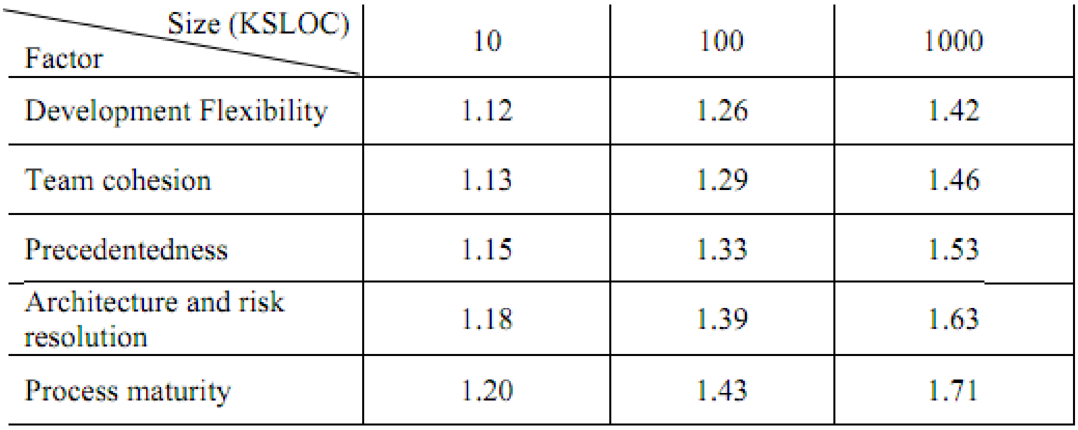
Rating Levels	Very Low	Low	Nominal	High	Very High
Effort Multipliers	0.82	0.92	1.00	1.10	1.26

As a result, a product requires very high reliability may cost 50% more effort than a product requires very low reliability. The effort may include the work on code inspection, unit test and program proving.

4.5 Effects of the learning curve

The effect of the learning curve shows that repetition of the same work results in higher productivity on that work. It is also effective in software development. If a development team begins working on a new applications domain and few people in the team are familiar with their new jobs, productivity will be relatively low. When they enhance their understanding of the new system, productivity will go up faster. However, there is a cap when the productivity reaches a certain point. Then they can hardly make any improvements unless they introduce the new technique. The Scale Factor PREC gives the same explanation: “If a product is similar to several previously developed projects, then the precedentedness is high”. Table 4.6 shows the productivity range in size-dependent.

Table 4.6 Size-Dependent Productivity Range



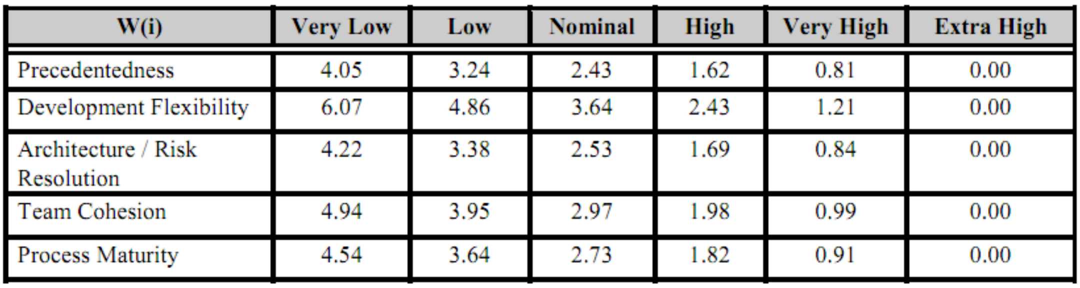
Size (KSLOC) Factor	10	100	1000
Development Flexibility	1.12	1.26	1.42
Team cohesion	1.13	1.29	1.46
Precedentedness	1.15	1.33	1.53
Architecture and risk resolution	1.18	1.39	1.63
Process maturity	1.20	1.43	1.71

Source: Barry Boehm, Safe and Simple Software Cost Analysis, 2000[19]

4.6 Other Cost Drivers.

COCOMO II involves five Scale Factors and seventeen Effort Multipliers. The five Scale Factors account for the relative economies or diseconomies of scale encountered for software projects of different sizes [Banker et al. 1994a]. The Effort Multipliers can be classified to Product Attributes, Computer Attributes, Staff Attributes, and Project Attributes.

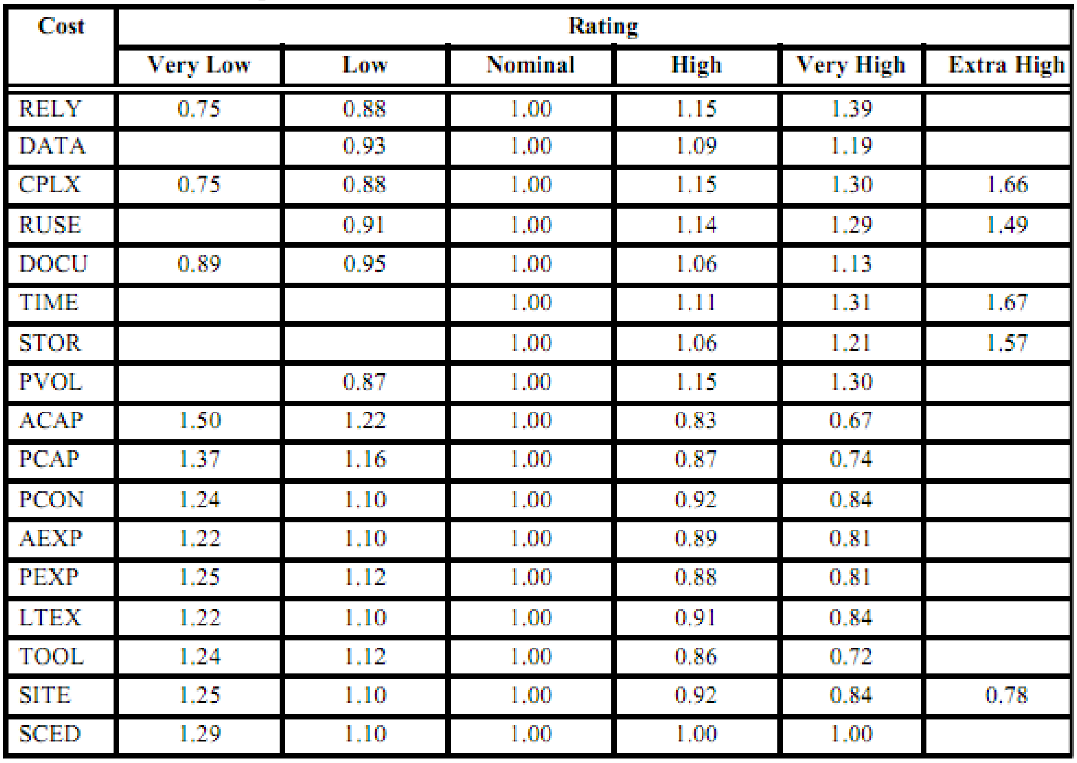
Table 4.7 Scale Factors



W(i)	Very Low	Low	Nominal	High	Very High	Extra High
Precedentedness	4.05	3.24	2.43	1.62	0.81	0.00
Development Flexibility	6.07	4.86	3.64	2.43	1.21	0.00
Architecture / Risk Resolution	4.22	3.38	2.53	1.69	0.84	0.00
Team Cohesion	4.94	3.95	2.97	1.98	0.99	0.00
Process Maturity	4.54	3.64	2.73	1.82	0.91	0.00

Source: COCOMO II Definition Manual version 1.4, http://sunset.usc.edu/research/COCOMOII/Docs/modelman.pdf[20]

Table 4.8 Effort Multipliers



Cost	Rating					
	Very Low	Low	Nominal	High	Very High	Extra High
RELY	0.75	0.88	1.00	1.15	1.39	
DATA		0.93	1.00	1.09	1.19	
CPLX	0.75	0.88	1.00	1.15	1.30	1.66
RUSE		0.91	1.00	1.14	1.29	1.49
DOCU	0.89	0.95	1.00	1.06	1.13	
TIME			1.00	1.11	1.31	1.67
STOR			1.00	1.06	1.21	1.57
PVOL		0.87	1.00	1.15	1.30	
ACAP	1.50	1.22	1.00	0.83	0.67	
PCAP	1.37	1.16	1.00	0.87	0.74	
PCON	1.24	1.10	1.00	0.92	0.84	
AEXP	1.22	1.10	1.00	0.89	0.81	
PEXP	1.25	1.12	1.00	0.88	0.81	
LTEX	1.22	1.10	1.00	0.91	0.84	
TOOL	1.24	1.12	1.00	0.86	0.72	
SITE	1.25	1.10	1.00	0.92	0.84	0.78
SCED	1.29	1.10	1.00	1.00	1.00	

Source: COCOMO II Definition Manual version 1.4, http://sunset.usc.edu/research/COCOMOII/Docs/modelman.pdf[20]

CHAPTER 5

CALIBRATING RISK IN COCOMO

5.1 Risks Identity

No one denies that COCOMO has risks too. According to Boehm’s data, COCOMO II 1998 estimation accuracy is shown in Table 5.1:

Table 5.1 COCOMO 1998 Estimation Accuracy

	Prediction Accuracy	General	Calibrate to Organization
Effort	PRED(.30)	75%	80%
Schedule	PRED(.30)	72%	81%

Source: Barry Boehm, COCOMO/SCM Forum #13, page 6, USC, 1998[21]

There are several factors that result in uncertainties in COCOMO.

The first reason is sizing. Table 4.4 shows the accuracy of different methods of Function Point counting. The accuracy may range from +/-100% to +/-5%. Even if the number of Function Point is correct, it still needs to be converted to SLOC in COCOMO. Table 4.3 shows a huge distribution of source lines of statement per function point. Thus the SLOC used in COCOMO may be overestimated or underestimated. Due to the exponential increasing of effort by size, the larger the size of the project, the bigger risks it takes.

The second reason is productivity. In COCOMO, productivity is determined by a nominal value and a group of effort multipliers. The effort multipliers have both positive and negative effects on productivity. The effects are based on the rating of effort multipliers which is shown in Table 4.7. Then the productivity ranges of these effort multipliers can be calculated from the data. Here is an example in Figure 5.1.

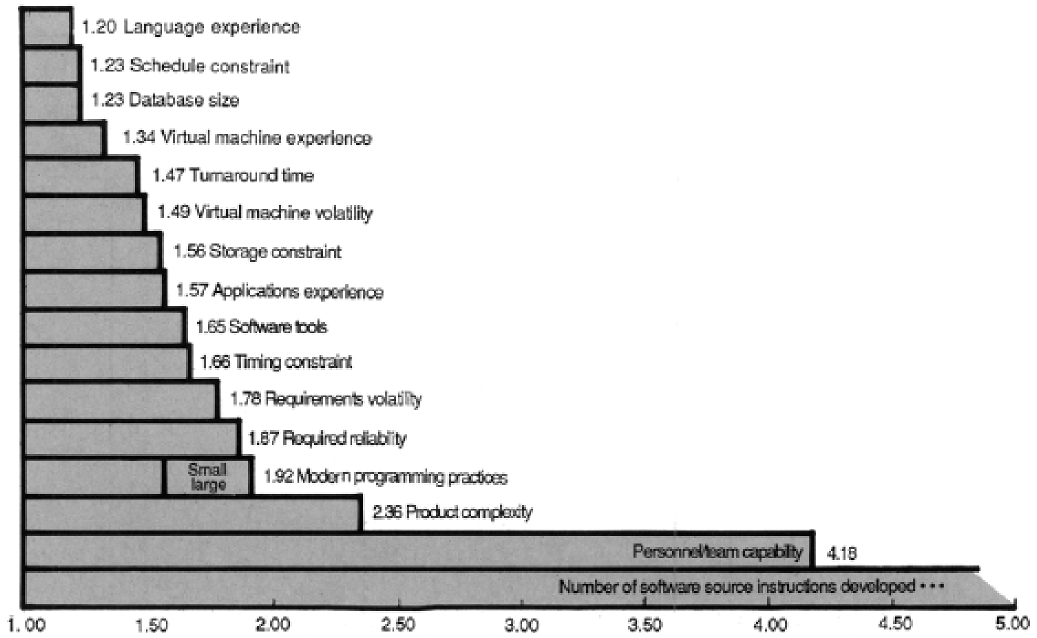


Figure 5.1 COCOMO software life-cycle productivity ranges, 1985
Source: Barry W. Boehm, TRW, Improving Software Productivity,
http://csse.usc.edu/csse/TECHRPTS/1987/usccse87-502/usccse87-502.pdf[22]

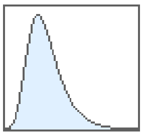
These productivity ranges show the relationships between all effort multipliers and productivity which reflects one effort multiplier's ability to increase or reduce effort required to develop a software product. However, the numbers in Table 4.8 are gathered by statistical method based on historical data. The number of each rating is possibly an average value that cannot be always accurate.

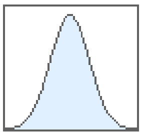
5.2 Uncertainties in Effort Multipliers

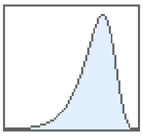
To find out the uncertainties in productivity, it is necessary to know the accuracy of COCOMO effort multipliers. The point is how to know the variance of each Effort Multiplier. In Boehm's book "Software Engineering Economics", he provides the data that COCOMO effort multiplier versus ideal effort multiplier. Because the ideal effort multipliers are the real project data, they can be considered as the range of the accuracy.

If the ideal effort multipliers are close to the COCOMO effort multiplier, the accuracy is relatively high. If not, the accuracy is relatively low. Because there are more than 60 projects which are included in the data, it is possible to find a statistical distribution for Effort Multipliers.

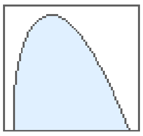
The procedure of selecting a statistical distribution that best fits to a data set generated by some random process is called Distribution Fitting. The purpose here is to find the best distribution for the Effort Multipliers because statistical distribution is a perfect way to express uncertainty. There are several basic characteristics in distribution, which are shown below.

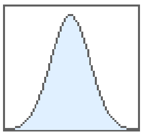

Right-Skewed

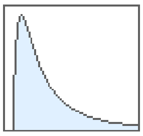

Symmetric


Left-Skewed

Normally, the distributions of Effort Multipliers are right-skewed but sometimes they can be symmetric.


Bounded


Unbounded


Non-Negative

The normal distribution is defined on the entire real axis (-Infinity, +Infinity), and if the nature of your data is such that it is bounded or non-negative (can only take on positive values), then this distribution is almost certainly not a good fit. However, the distribution of Effort Multiplier is obviously bounded and our purpose is to find the upper bound and lower bound to determine the range of uncertainty. Therefore, the normal distribution is apparently not a good fit to Effort Multipliers. To simplify the estimation process, a triangular distribution is usually a good choice. Hence, it is important to find a tool to calculate the triangular distribution of Effort Multipliers. EasyFit[23] is such software that can do the calculation to save our time.

Here is an example of how to find a distribution for an effort multiplier.

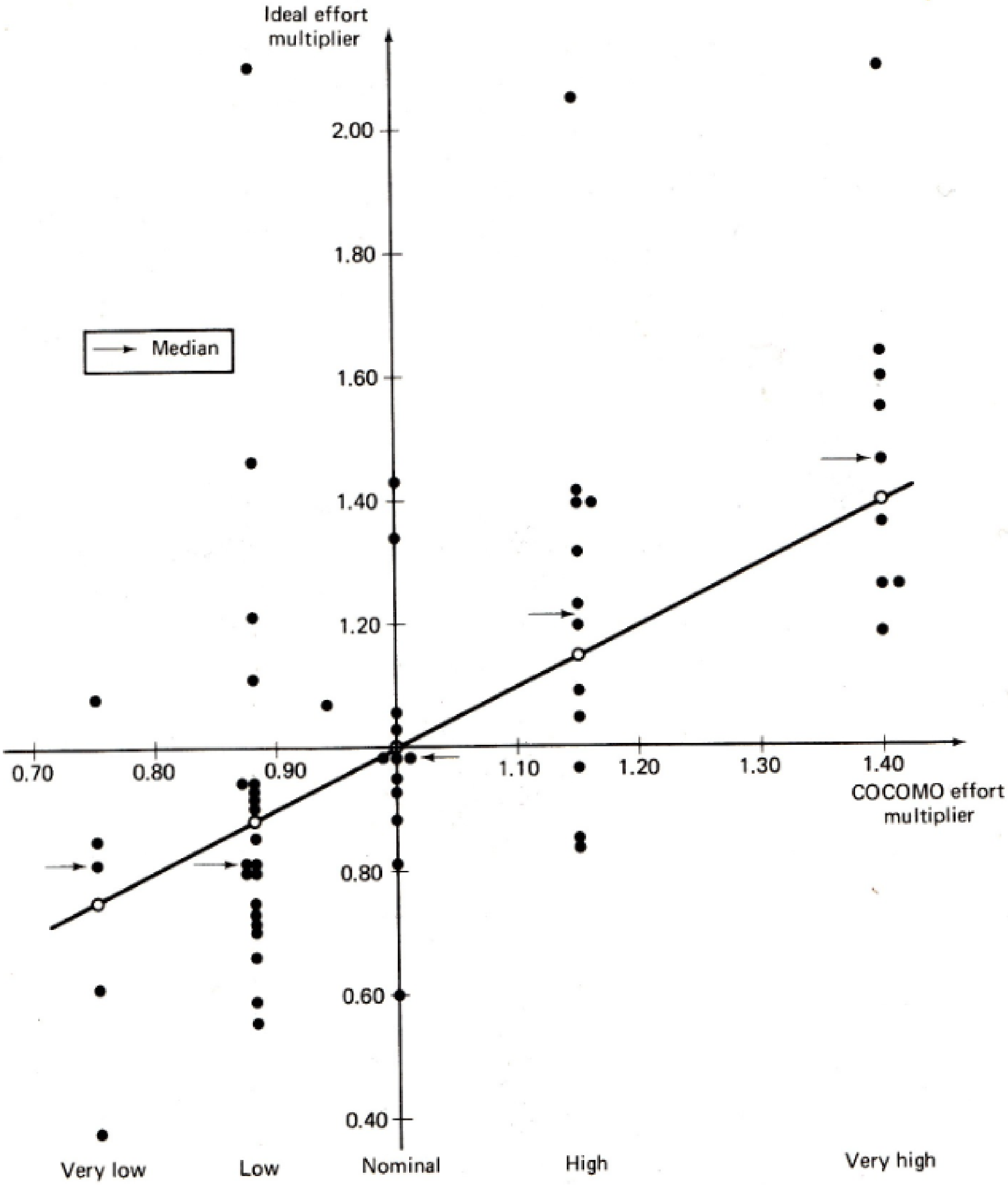


Figure 5.2 COCOMO versus ideal effort multiplier- required reliability
Source: Barry W. Boehm, Software Engineering Economics, p379, Prentice-Hall, INC., 1981[24]

Step 1: Collect the ideal effort multipliers from Figure 5.2 as shown below.

Table 5.2 Ideal Effort Multipliers of Reliability

Very low	Low	Nominal	High	Very High
0.75	0.88	1	1.15	1.4
1.1	2.1	1.42	2.05	2.1
0.88	1.48	1.35	1.4	1.63
0.8	1.2	1.08	1.4	1.6
0.6	1.1	1.05	1.4	1.52
0.4	0.95	1	1.3	1.45
	0.95	1	1.22	1.35
	0.94	1	1.2	1.25
	0.93	1	1.1	1.25
	0.92	0.95	1.05	1.2
	0.85	0.93	0.95	
	0.8	0.9	0.83	
	0.8	0.8	0.82	
	0.8	0.6		
	0.8			
	0.75			
	0.74			
	0.73			
	0.72			
	0.71			
	0.66			
	0.58			
	0.55			

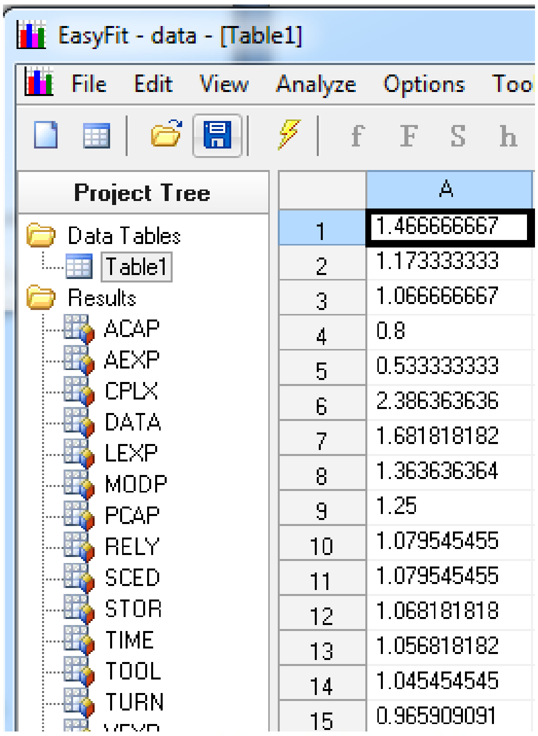
Step 2: Calculate the ratio between the ideal effort multiplier and COCOMO effort multiplier, and the result will be:

Table 5.3 Adjusted Effort Multipliers of Reliability

Very low	Low	Nominal	High	Very High
0.75	0.88	1	1.15	1.4
1.466667	2.386364	1.42	1.782609	1.5
1.173333	1.681818	1.35	1.217391	1.164286
1.066667	1.363636	1.08	1.217391	1.142857
0.8	1.25	1.05	1.217391	1.085714
0.533333	1.079545	1	1.130435	1.035714
	1.079545	1	1.06087	0.964286
	1.068182	1	1.043478	0.892857
	1.056818	1	0.956522	0.892857
	1.045455	0.95	0.913043	0.857143
	0.965909	0.93	0.826087	
	0.909091	0.9	0.721739	
	0.909091	0.8	0.713043	
	0.909091	0.6		
	0.909091			
	0.852273			
	0.840909			
	0.829545			
	0.818182			
	0.806818			
	0.75			
	0.659091			
	0.625			

Step 3: Import the result to EasyFit in one column.

Step 4: Press F9 to run the distribution.



Project Tree

- Data Tables
  - Table1
- Results
  - ACAP
  - AEXP
  - CPLX
  - DATA
  - LEXP
  - MODP
  - PCAP
  - RELY
  - SCED
  - STOR
  - TIME
  - TOOL
  - TURN

	A
1	1.466666667
2	1.173333333
3	1.066666667
4	0.8
5	0.533333333
6	2.386363636
7	1.681818182
8	1.363636364
9	1.25
10	1.079545455
11	1.079545455
12	1.068181818
13	1.056818182
14	1.045454545
15	0.965909091

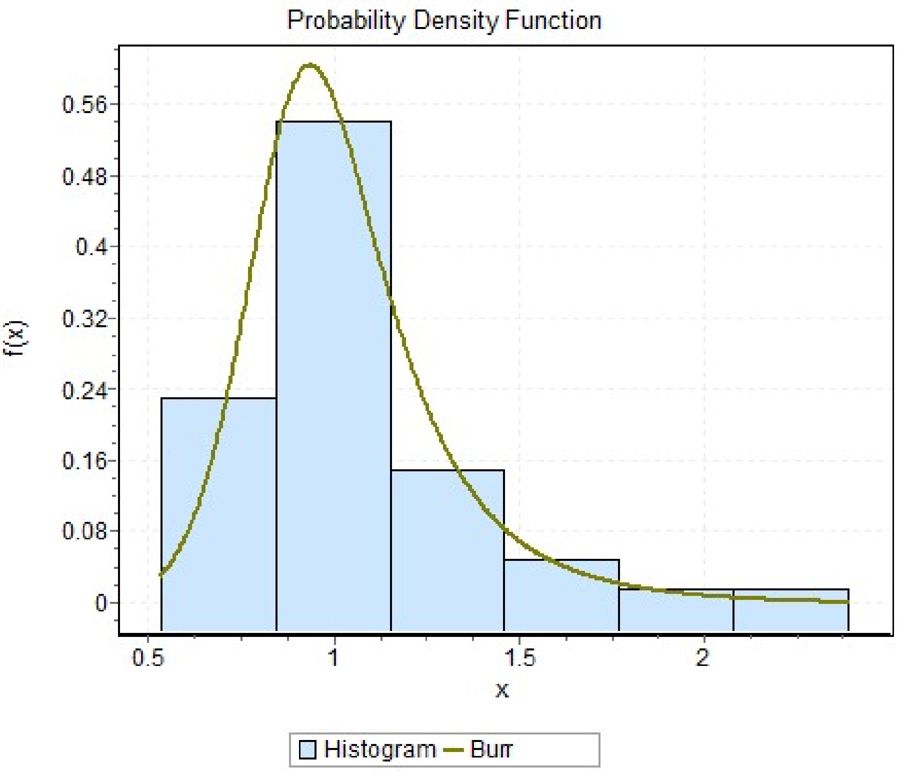


Figure 5.3 Distribution of Required Software Reliability. Burr is a default statistics model in EasyFit and it can be ignored in this figure. X axle represents the percentage.

Step 5: Due to noise in the data, there are some samples which can be considered as outliers. Thus after refining the data, it shows a triangular distribution in Figure 5.4:

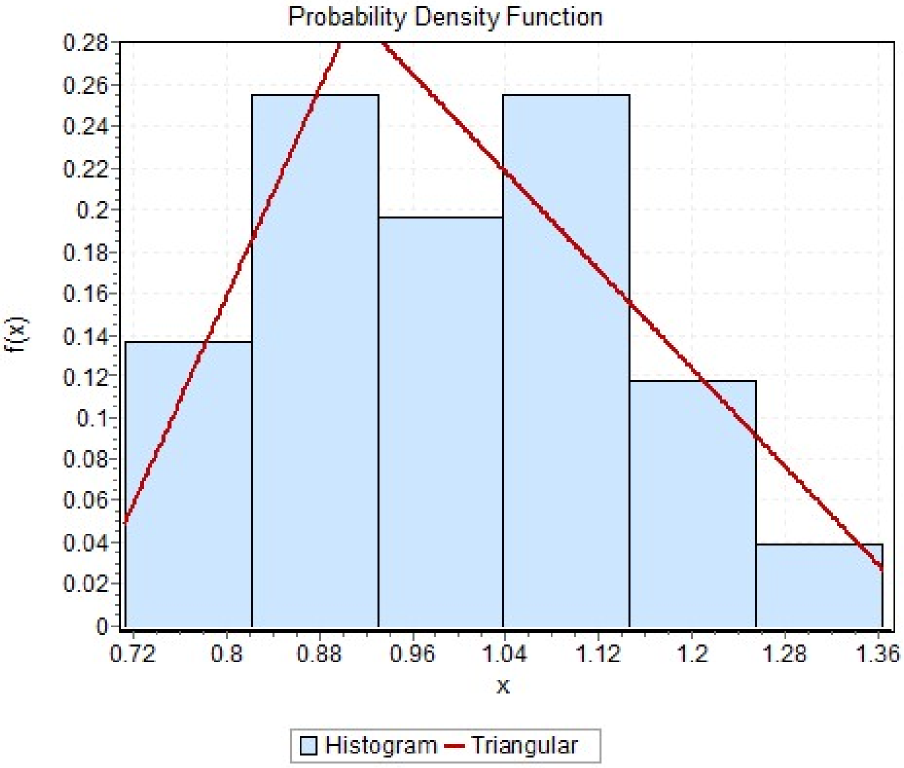


Figure 5.4 Triangular Distribution of Required Software Reliability.

In Figure 5.4, the range of Required Software Reliability values is from 75% to 130% which means the actual value could appear within this range. The other Effort Multipliers have the same issue and the same steps above can be applied to find the distributions of other Effort Multipliers. These distributions are based on the data from COCOMO 81, in this study it assumes that the distribution remains the same until there are new data available. For example, in COCOMO II 2000, value of RELY for very high ratio is 1.26; the possible range will be from 0.95 to 1.64.

When it applies the same steps to Use of Software Tools, the data from COCOMO 81 shows distributions in Table 5.4:

Table 5.4 Adjusted Effort Multipliers of Use of Software Tools

Very high	high	Nominal	Low	Very low
0.83	0.91	1	1.1	1.24
1.373494	1.67033	1.5	1.790909	1.225806
	1.340659	1.22	1.445455	0.895161
	1.274725	1.17	1.436364	0.564516
	1.208791	1.15	1.245455	
	1.131868	1.12	1.145455	
	1.043956	1.1	1.145455	
	1.010989	1.08	1.081818	
	0.989011	1.08	1.045455	
	0.934066	1.05	1.027273	
	0.824176	1.02	0.990909	
	0.802198	1.01	0.954545	
	0.604396	0.99	0.945455	
		0.99	0.918182	
		0.97	0.863636	
		0.9	0.818182	
		0.9	0.736364	
		0.9	0.636364	
		0.89		
		0.88		
		0.86		
		0.85		
		0.85		
		0.83		
		0.8		
		0.76		
		0.73		
		0.67		

Source: Barry W. Boehm, Software Engineering Economics, p463, Prentice-Hall, INC., 1981[25]

Delete the outliers which are smaller than 70% or greater than 150%, then import the data to EasyFit:

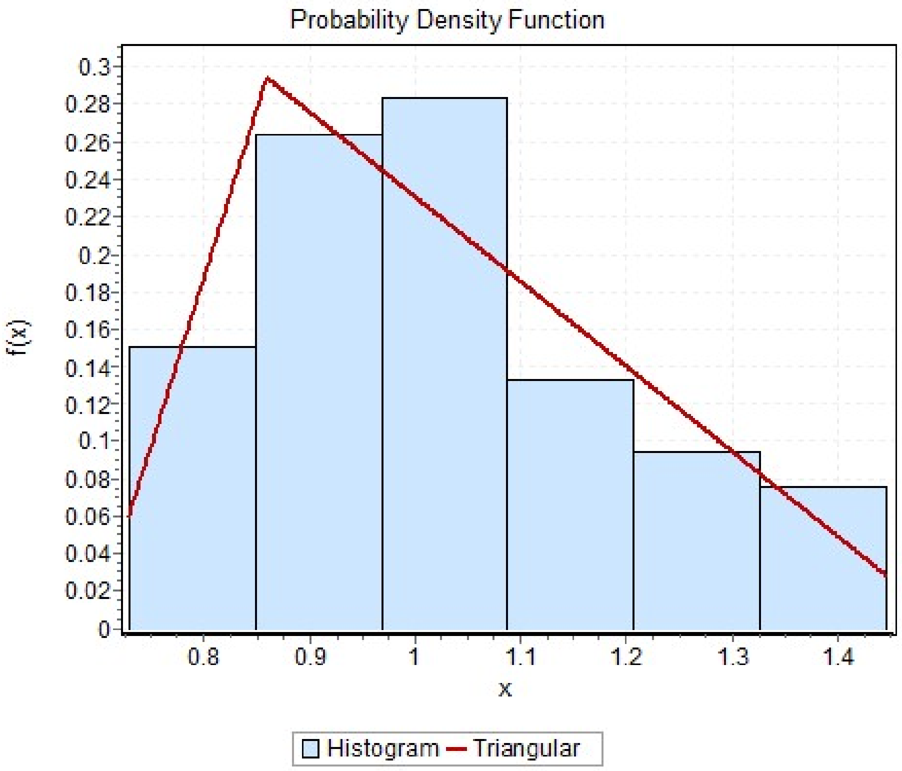


Figure 5.5 Triangular Distribution of Use of Software Tools.

In Figure 5.5, the range of Use of Software Tools values is from 75% to 140%

When it applies the same steps to Schedule Constraint, the data from COCOMO 81 shows distributions in Table 5.5.

Table 5.5 Adjusted Effort Multipliers of Use of Software Tools

Very high	high	Nominal	Low	Very low
1.1	1.04	1	1.08	1.23
	1.653846	2.17	1.703704	1.227642
	1.134615	1.5	1.25	1.227642
	1.076923	1.43	1.212963	1.162602
	1.057692	1.43	1.138889	1.146341
	0.923077	1.39	1.055556	1.04878
	0.836538	1.35	1.027778	1.01626
		1.29	0.981481	0.97561
		1.14	0.953704	0.96748
		1.1	0.842593	
		1.07	0.648148	
		1.07		
		1.05		
		1.05		
		1.03		
		0.99		
		0.99		
		0.95		
		0.94		
		0.93		
		0.93		
		0.9		
		0.9		
		0.9		
		0.89		
		0.85		
		0.85		
		0.85		
		0.84		
		0.84		
		0.82		
		0.8		
		0.75		
		0.72		
		0.6		
		0.51		

Source: Barry W. Boehm, Software Engineering Economics, p469, Prentice-Hall, INC., 1981[26]

Delete the outliers which are smaller than 80% or greater than 140%, then import the data to EasyFit. In Figure 5.6, the range of Schedule Constraint value is from 84% to 130%.

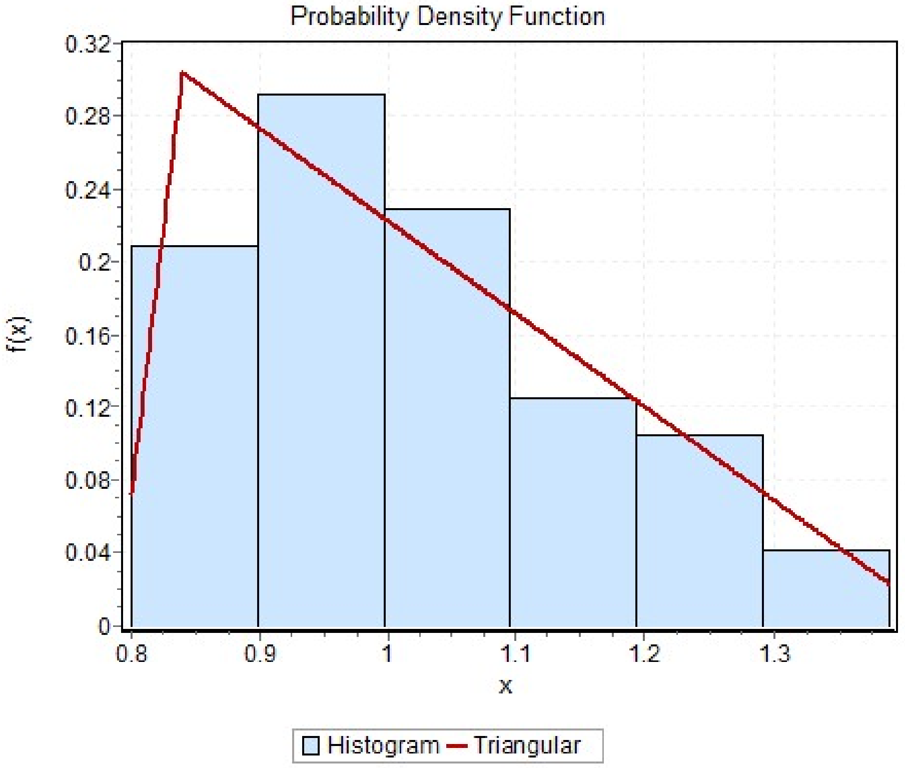


Figure 5.6 Triangular Distribution of Schedule Constraint

5.3 Uncertainty in Scale Factors

Scale Factors are very different from Effort Multipliers because their influence on estimation is affected by the size of the project. In COCOMO II, the productivity range of Effort Multipliers does not involve the size of the project. However, the productivity range of Scale Factors is based on a 100 KLOC project. Therefore, the productivity range of Scale Factors is not normally the same in different projects.

COCOMO II is calibrated from 161 projects[27]. Thus, the values of Scale Factors are also calibrated from the data. For example, the latest value for Process Maturity is 7.80, 6.24, 4.68, 3.12, 1.56, 0.00 mapped to Very Low, Low, Normal, High, Very High, Extra High. However, the numbers also involve risks. The ideal value is possibly within a certain range around this number. PMAT Coefficient Range is shown in Figure 5.5.

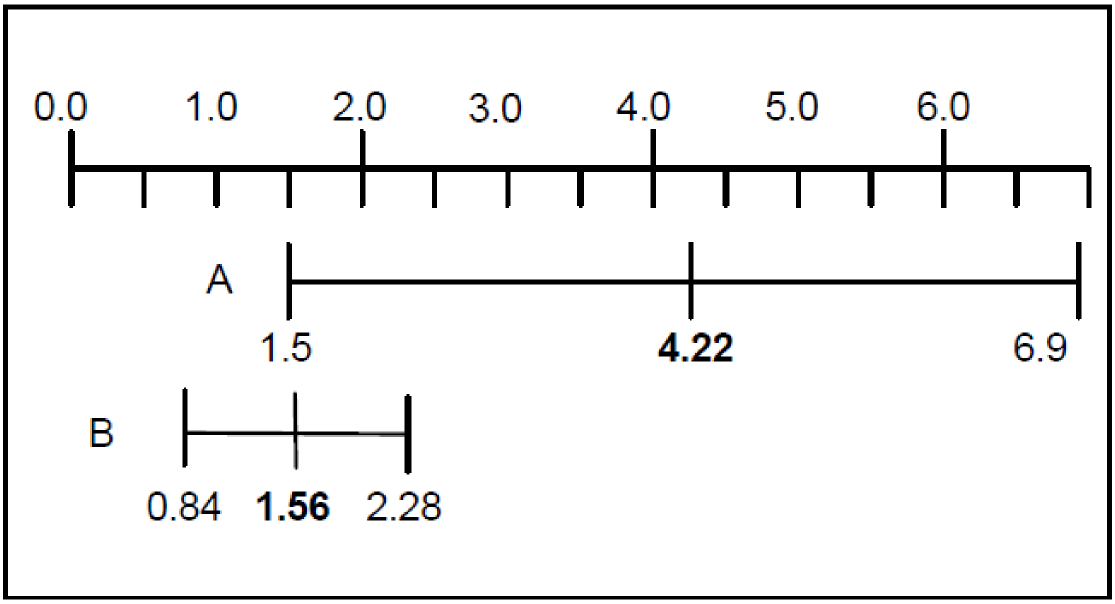
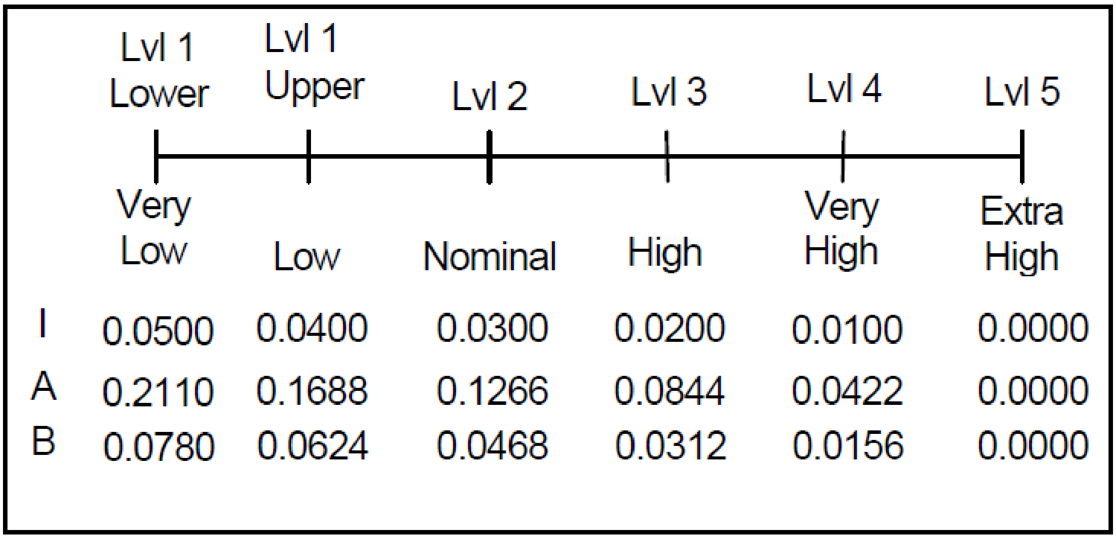


Figure 5.7 PMAT coefficient range. 4.22 is the mean value of sample A and 1.56 is the mean value of Sample B.
Source: Bradford K. Clark, The Effects of Software Process Maturity on Software Development Effort, http://sunset.usc.edu/~bkclark/Research/Dissertation.pdf[26]

In Figure 5.7, A is a sample of data from 112 projects collected in 1997 which is used to initially demonstrate the influence of process maturity on effort. B is another sample of 161 projects collected in 1998. The mean value of B was used to calibrate COCOMO II. The process is shown in Figure 5.8.



	Lvl 1 Lower Very Low	Lvl 1 Upper Low	Lvl 2 Nominal	Lvl 3 High	Lvl 4 Very High	Lvl 5 Extra High
I	0.0500	0.0400	0.0300	0.0200	0.0100	0.0000
A	0.2110	0.1688	0.1266	0.0844	0.0422	0.0000
B	0.0780	0.0624	0.0468	0.0312	0.0156	0.0000

Figure 5.8 PMAT Scale Values. The levels are the rating of Process Maturity, I is initiated number.

Source: Bradford K. Clark, The Effects of Software Process Maturity on Software Development Effort, http://sunset.usc.edu/~bkclark/Research/Dissertation.pdf[26]

Each initial rating is multiplied by sample coefficient (4.22 or 1.56), and the result is called scale value. But the coefficient is a mean value which could introduce uncertainty to the result. For example, sample B shows the coefficient is from 0.84 to 2.28. The distribution of PMAT scale factor is from 54% to 146%. Because the other scale factors are calculated by the same method, the risk in scale value will be predictable if the coefficient range is available. However, due to lack of the data for other four scale factors, the only way is to simulate the coefficient range of other scale factors in this study.

In Figure 5.7, the mean value of PMAT coefficient range is 1.56. According to COCOMO II 2000, the value of scale factors for nominal ratio is 3.72 (PREC), 3.04 (FLEX), 4.24 (RESL), 3.29 (TEAM), 4.68 (PMAT). There is an equation: scale value = scale coefficient * initial rating (4.68 = 1.56 * 3).

According to this equation, it is possible to get the coefficient value of other scale factors. For example, the scale values of PREC are shown in Table 5.6:

Table 5.6 Scale Values of PREC

	Precendentedness					
	Very low	Low	Nominal	High	Very high	Extra high
Initial	0.0500	0.0400	0.0300	0.0200	0.0100	0.0000
B	0.0620	0.0496	0.0372	0.0248	0.0124	0.0000

According to Table 5.6, the coefficient value of PREC is: scale value of Very low (0.0620) / initial rating of Very low (0.0500) = 1.24.

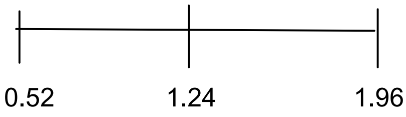
If the equation is applied to the other scale factors, the coefficient values of scale factors would be:

Table 5.7 Coefficient Value of Scale Factors

PREC	FLEX	RESL	TEAM	PMAT
1.24	1.01	1.41	1.10	1.56

However, without accurate data of each scale factor it is difficult to find the variance of each scale factors. Thus the most effective way is to assume the same coefficient range of PMAT will happen to the other scale factors. The variance of PMAT coefficient is from -0.72 to +0.72, and in COCOMO the sum of five scale factors determine the value of exponent. Therefore, the variance of different scale factors is interchangeable.

The lower bound of PREC coefficient range is 1.24-0.72=0.52, and the upper bound of PREC coefficient range is 1.24+0.72=1.96.



Because 0.52/1.24 = 42% and 1.96/1.24 = 158%, the variance of PREC coefficient is from 42% to 158%. If there is a triangular distribution for PREC, it will be:

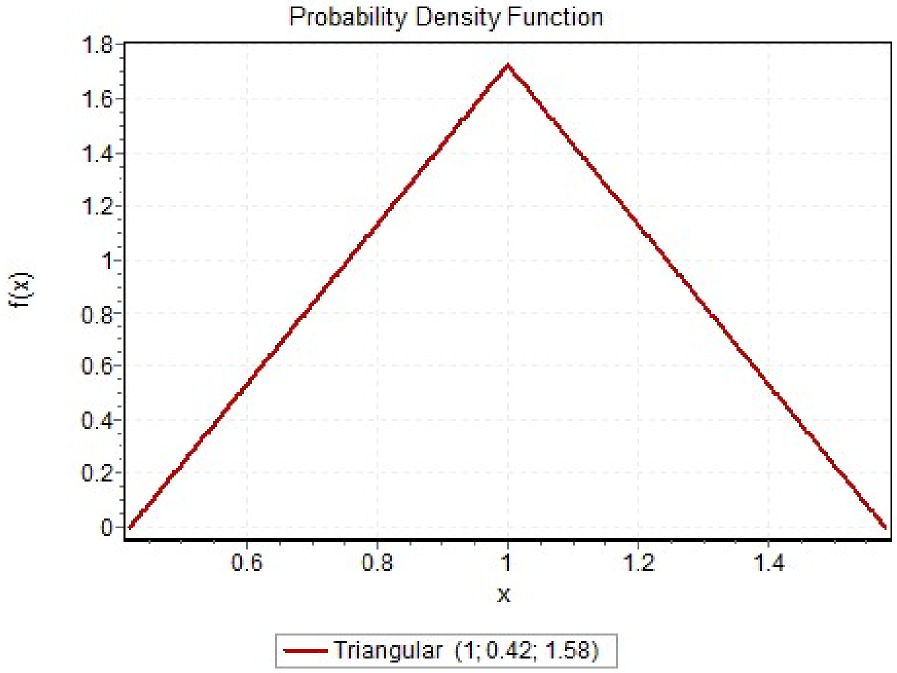


Figure 5.9 Probability density of PREC.

Implement the same method on the other scale factors, the result is shown in Table 5.8:

Table 5.8 Coefficient Range of Scale Factors

	PREC	FLEX	RESL	TEAM
Coefficient Range	0.52 – 1.96	0.29 – 1.73	0.69 – 2.13	0.38 – 1.82
Percentage Range	42% - 158%	29% - 171%	50% - 150%	35% - 165%

5.4 Monte Carlo Simulation and Tools

Chapter 4 explains that the uncertainties in COCOMO involves sizing, productivity, effort multipliers and scale factors. Hence, there should be a method to collect all of the data to calculate a value of overall variance. Monte Carlo Simulation is a method to repeat random samples in a certain range to compute their result. Theoretically, there are certain distribution ranges of sizing, productivity, effort multipliers and scale factors. Monte Carlo Simulation could help to calculate the overall distribution. Thus it may improve COCOMO by adding a range to the result to make it not only a point estimate.

There are many kinds of software that can do Monte Carlo Simulation such as Oracle Crystal Ball, ModelRisk Standard 4.0, Palisade @Risk 5.7, and Frontline Risk Solver 11.0. Here is a comparison of these Monte Carlo Simulation tools from Crystal Ball Services<http://www.crystalballservices.com>. The conclusion is

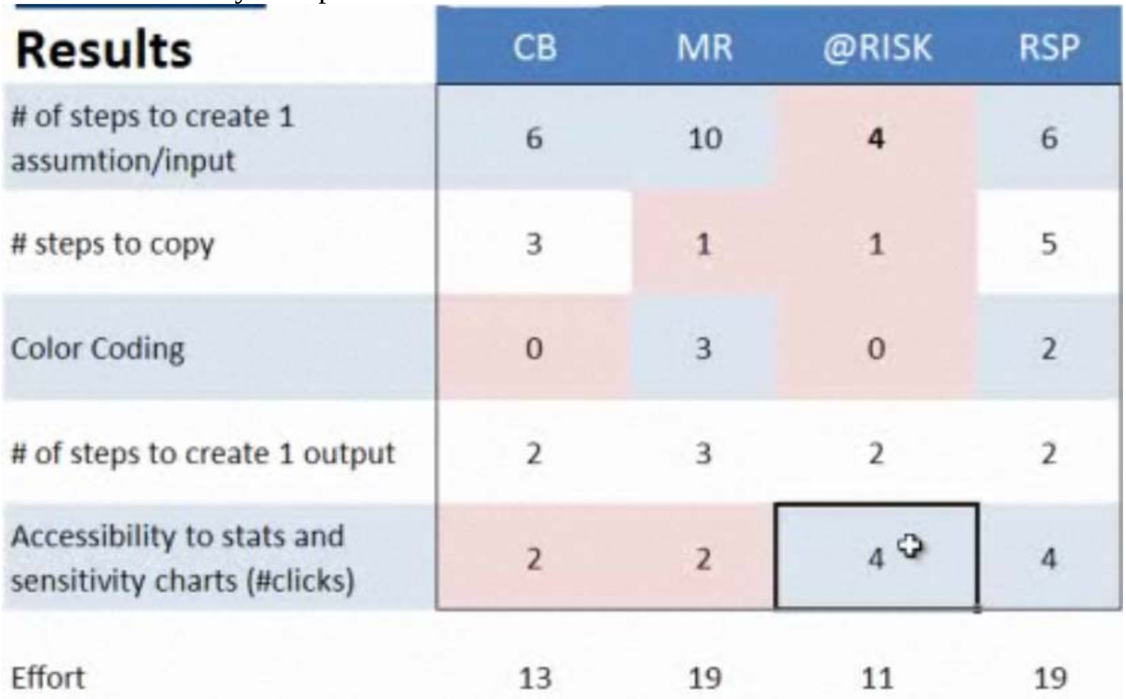
"Crystal Ball is an efficient modeling package that is easy to use. Like the other tools, they have formulas but they cannot be included in the sensitivity analysis. So use these with care.

ModelRisk's approach to building inputs and outputs offers lots of flexibility but can introduce errors into model if the user is not paying attention. Same is to be said about the color coding.

@Risk is a very efficient modeling tool but some of the interface buttons could be clearer and easier to find. This makes rummaging around the interface more likely for inexperienced users.

Risk Solver is almost there but have some bugs in their input definition process that result in lots of extra work. Otherwise a very interesting solution." [28]

Table 5.9 Usability Comparison Between Four Monte Carlo Tools

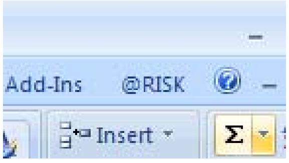


Results	CB	MR	@RISK	RSP
# of steps to create 1 assumtion/input	6	10	4	6
# steps to copy	3	1	1	5
Color Coding	0	3	0	2
# of steps to create 1 output	2	3	2	2
Accessibility to stats and sensitivity charts (#clicks)	2	2	4	4
Effort	13	19	11	19

Source: http://www.crystalballservices.com/Resources/ConsultantsCornerBlog/EntryId/71/Excel-Simulation-Show-Down-Comparing-the-top-Monte-Carlo-Simulation-Tools.aspx[27]

The chart shows how many steps a tool requires for implementing a simulation, the smaller numbers mean better efficiency. In this experiment, the author chooses to use Palisade @Risk which can be downloaded from website of Palisade <http://www.palisade.com/risk/>.

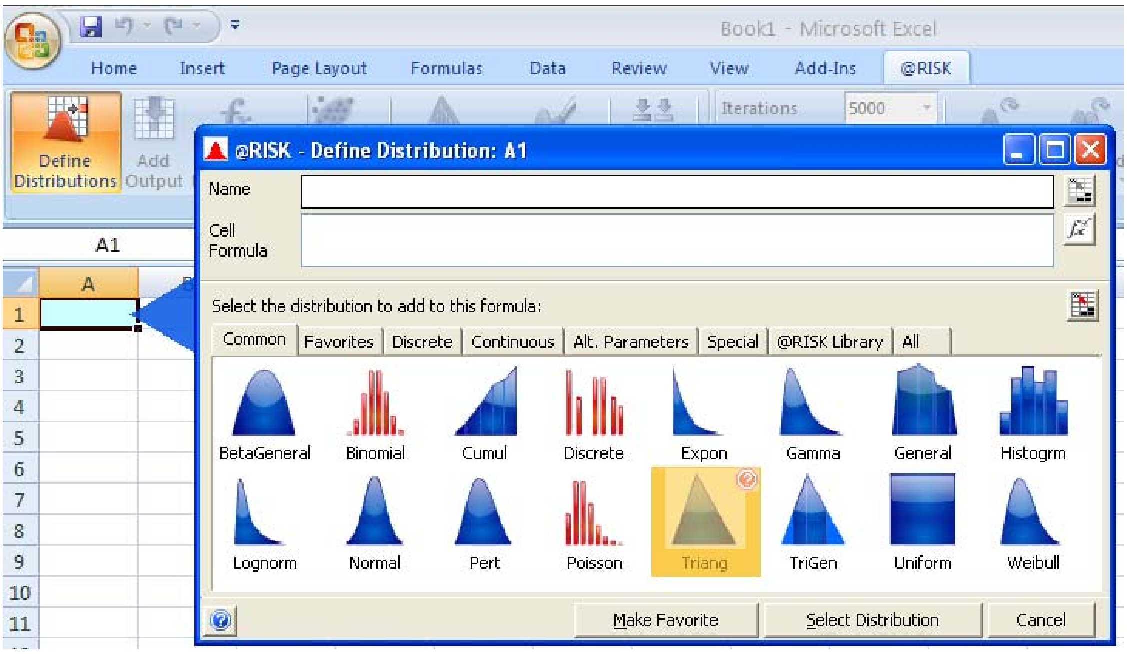
@Risk is a kind of add-in to Microsoft Excel, thus it is required to install Microsoft Excel before implementing @Risk. The author is using Microsoft Excel 2007 on his computer and it is perfectly compatible with @Risk 5.7. After installing @Risk 5.7 to the machine, run @Risk 5.7.exe and it will startup Microsoft Excel 2007 which is a little different from original version.



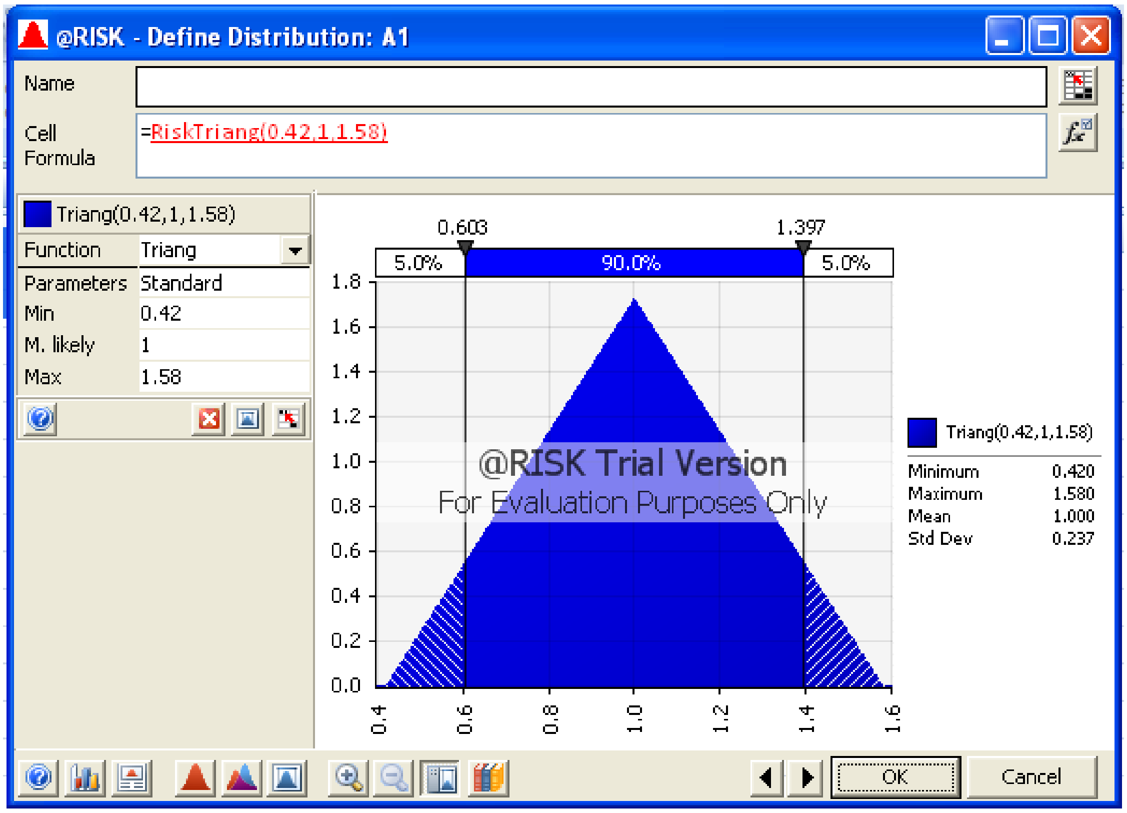
Click the @Risk at the Toolbar, and then start defining the distributions.

Step 1: Define Distribution

Choose a cell and Click button “Define Distribution”, you will see



Select Triangular Distribution and define the range. This example is using the percentage range from Table 5.8.



Name

Cell Formula =RiskTriang(0.42,1,1.58)

Triang(0.42,1,1.58)

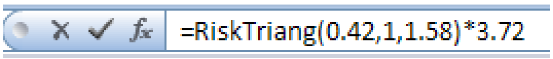
Function	Triang
Parameters	Standard
Min	0.42
M. likely	1
Max	1.58

Triang(0.42,1,1.58)

Minimum	0.420
Maximum	1.580
Mean	1.000
Std Dev	0.237

Step 2: Input the value and calculate

After determining the distribution range, there should be the nominal value for this cell. For example, the nominal value for PREC is 3.72, and then input it to formula line:

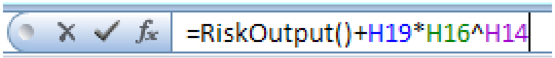


=RiskTriang(0.42,1,1.58)*3.72

As for now, the cell for PREC factor is set. The same method is applicable to set the other factors such as effort multipliers or scale factors.

Step 3: Add Output

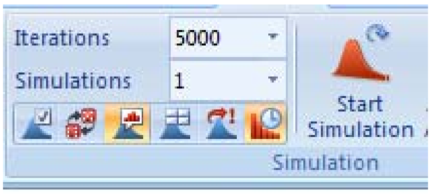
When all the needed factors are set, an output cell is required to run the calculations. For example, let cell H21 be the output cell and it should contain H19*H16^H14. Cell H19 represents the coefficient A in COCOMO, cell H16 represents size and H14 represents the exponent E. Finally, select cell H21 and click button “Add Output”. The formula line of H21 will be like this:



```
=RiskOutput()+H19*H16^H14
```

Step 4: Start Simulation

Input Iterations and click button “Start Simulation” to display the result. Normally 5000 iterations should be enough.



Iterations 5000

Simulations 1

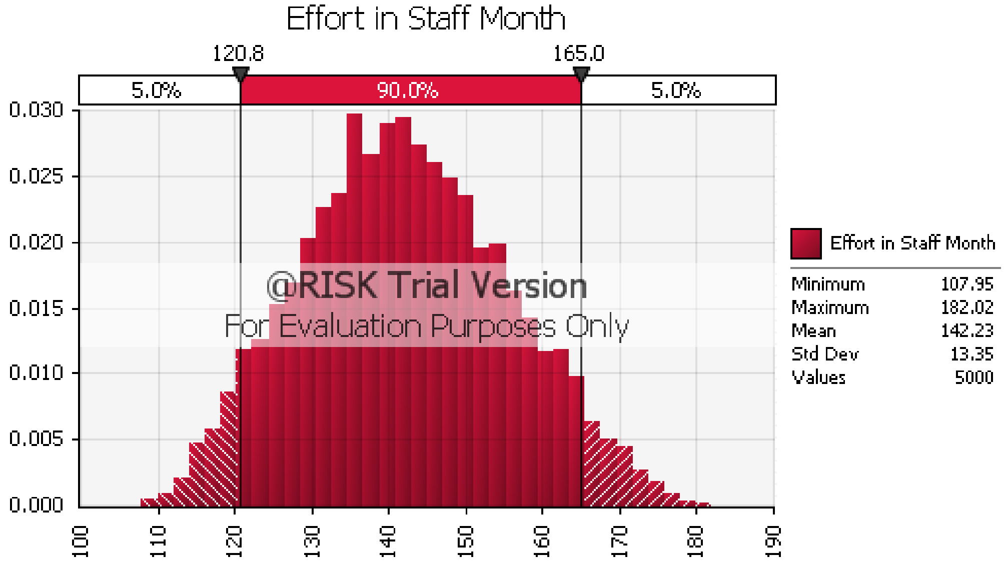


Figure 5.10 Example of Monte Carlo simulation.

Figure 5.10 shows the overall distribution of an ordinary COCOMO estimation. It contains the minimum, maximum, mean and standard deviation of effort in staff month. Therefore, Monte Carlo Simulation is an applicable method to calibrate risks in COCOMO and @Risk is also an appropriate tool in this field.

CHAPTER 6

ANALYSIS AND SOLUTION

6.1 Analysis on Historical Data

In this chapter, there is an example to analyze the variance in COCOMO based on some real project data. The data is from Boehm's paper Prototyping and Specifying: A Multi-Project Experiment. In his experiment, there are seven groups of people developed a same small size software product. Four groups are using Specifying method and 3 groups are using Prototype method. The size is between 2 KLOC and 4KLOC. According to the result, "The COCOMO model strongly overestimated the amount of effort required to develop the experimental products. The overestimates were typically by a factor of about 2.5, much larger than could be explained by not counting the typical 30–40% of the workday devoted to non-project activities." [29]

However, the productivity of the seven groups is very close to each other. The size and effort are shown in Table 6.1.

Table 6.1 Summary of Projects

	Specifying				Prototype		
	Group1	Group2	Group3	Group4	Group5	Group6	Group7
Size (SLOC)	2985	3164	4606	2809	1952	2726	1514
Effort (Staff Hour)	589	498	459	789	323	422	232
Productivity (SLOC/SH)	5.1	6.4	10	3.6	6	6.5	6.5
Productivity (SLOC/SM)	897.6	1126.4	1760	663.6	1056	1144	1144

Source: Barry W. Boehm*, Terence E. Gray, and Thomas Seewaldt, PROTOTYPING VS. SPECIFYING: A MULTI-PROJECT EXPERIMENT University of California, Los Angeles Computer Science Department[30]

The lowest productivity of Specifying teams is 663.6 SLOC/Staff Month. The highest is 1760 SLOC/Staff Month. And the average value is 1111.7 SLOC/Staff Month. Because the nominal productivity of COCOMO II is only 340 SLOC/Staff Month, it explains why COCOMO overestimated the amount of required effort so much. Then it can build a triangular distribution of Specifying:

Lower bound = 663.6/1111.7 = 59.7%

Upper bound = 1760/1111.7 = 158.3%

Therefore, the productivity range of Specifying is from 59.7% to 158.3%.

The lowest productivity of Specifying teams is 1056 SLOC/Staff Month. The highest is 1144 SLOC/Staff Month. And the average value is 1114.7 SLOC/Staff Month. Because the nominal productivity of COCOMO II is only 340 SLOC/Staff Month, it explains why COCOMO overestimated the amount of required effort so much. Then it can build a triangular distribution of Specifying:

Lower bound = 1056/1114.7 = 94.7%

Upper bound = 1144/1111.7 = 102.9%

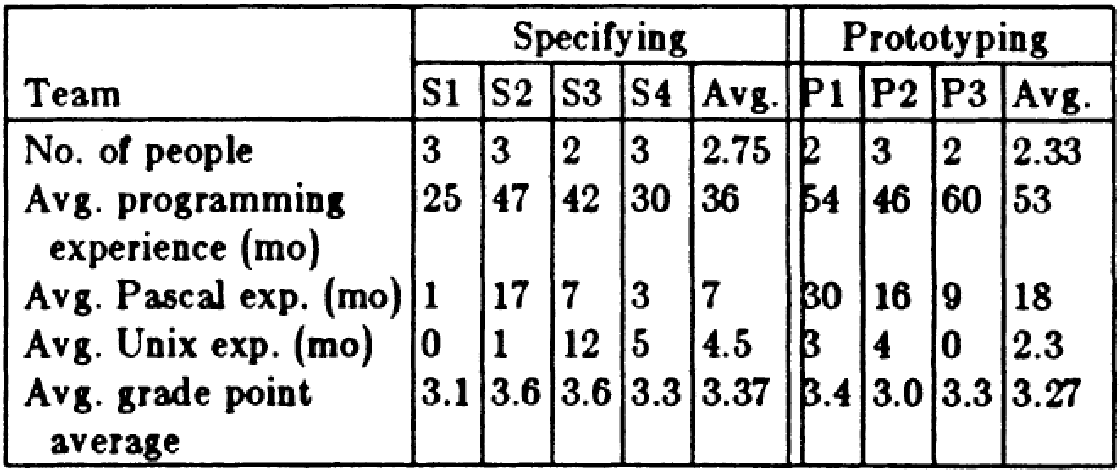
Therefore, the productivity range of Specifying is from 94.7% to 102.9%.

According to the data, both methods have the same average productivity. But Specifying has much more variance in productivity. The reason is possibly that the prototyping teams' products were 40% smaller on average, and required 45% less effort to develop. Admittedly, there might be some other factors that could affect the productivity.

Because the product in the experiment is so small that the scale factors will almost not affect the result, the main reason that causes the variance in productivity attributes in Effort Multipliers.

Normally, it needs to consider that how much the Personnel Factors may affect the experiment. The experiment team characteristics are shown in Table 6.2.

Table 6.2 Experimental Team Characteristics



Team	Specifying					Prototyping			
	S1	S2	S3	S4	Avg.	P1	P2	P3	Avg.
No. of people	3	3	2	3	2.75	2	3	2	2.33
Avg. programming experience (mo)	25	47	42	30	36	54	46	60	53
Avg. Pascal exp. (mo)	1	17	7	3	7	30	16	9	18
Avg. Unix exp. (mo)	0	1	12	5	4.5	3	4	0	2.3
Avg. grade point average	3.1	3.6	3.6	3.3	3.37	3.4	3.0	3.3	3.27

Source: Barry W. Boehm*, Terence E. Gray, and Thomas Seewaldt, PROTOTYPING VS. SPECIFYING: A MULTI-PROJECT EXPERIMENT University of California, Los Angeles Computer Science Department[30].

There are two Personnel Factors that could be responsible for the variance in the experiment. They are Programmer Capability and Language and Tool Experience. To rate the factors it needs to set a nominal value. In Specifying groups, a GPA of 3.37 and programming experience of 36 months are average numbers. In Prototyping teams, a GPA of 3.27 and programming experience of 53 months are average numbers.

According to the team characteristics, the Personnel Factor ratings for the seven teams are shown in Table 6.3.

Table 6.3 Experimental Team Ratings

	Team1	Team2	Team3	Team4	Team5	Team6	Team7
PCAP	Low	High	High	Nominal	High	Low	Nominal
LEXP	Low	High	High	Nominal	High	Nominal	Nominal

If the above factors are applied to COCOMO estimation, after eliminating the possible effect of personnel factors which means to make all ratings to nominal, the productivity of the seven teams is shown in Table 6.4.

Table 6.4 Adjusted Productivity and Effort

	Group1	Group2	Group3	Group4	Group5	Group6	Group7
Productivity (SLOC/SM)	1125.1	902	1409.4	663.6	845.6	1315.6	1144
Adjusted Effort(SM)	2.65	3.50	3.27	4.23	2.31	2.07	1.32

Then the productivity range of the two methods is:

Specifying:

Average productivity = 1025 SLOC/Staff Month

Lower bound = 663.6/1025 = 64.7%

Upper bound = 1409.4/1025 = 137.5%

In summary, the productivity range of Specifying is from 64.7% to 137.5%. The lower bound of Effort is 2.65 Staff Month and the upper bound of effort is 4.23 Staff Month. Average effort is 3.41 Staff Month.

Prototype:

Average productivity = 1101.7 SLOC/Staff Month

Lower bound = 845.6/1101.7 = 76.8%

Upper bound = 1315.6/1101.7 = 119.4%

In summary, the productivity range of Prototype is from 76.8% to 119.4%. The lower bound of Effort is 1.32 Staff Month and the upper bound of effort is 2.31 Staff Month. Average effort is 1.9 Staff Month.

6.2 Assumption and Improvements

The above calculations are based on that all the teams which have the same capability at the normal level. If there are better people in the teams, as somel of them can be rated as high in PCAP and LEXP.

For example, Team 1 is rated Low in PCAP and Low in LEXP. Because the ratings of PCAP and LTEX are:

	Low	High
PCAP	1.15	0.88
LTEX	1.09	0.91

The updated productivity of Team 1 will be 897.6*(1.15/0.88)*(1.09/0.91) = 1405 SLOC/SM

In summary, the productivity of the seven teams will be:

Table 6.5 Adjusted Productivity and Effort Under High Ratings

	Group 1	Group 2	Group 3	Group 4	Group 5	Group 6	Group 7
Productivity (SLOC/SM)	1405	1126.4	1760	828.7	1056	1642.9	1428.6
Adjusted Effort (SM)	2.12	2.81	2.62	3.39	1.85	1.66	1.06

With updated data, the new productivity range of Specifying and Prototype is calculated:

Specifying:

Average productivity = 1280 SLOC/Staff Month

Lower bound = 828.7/1280 = 64.7%

Upper bound = 1760.4/1280 = 137.5%

In summary, the productivity range of Specifying is from 64.7% to 137.5%. The lower bound of Effort is 2.12 Staff Month and the upper bound of effort is 3.39 Staff Month. Average effort is 2.74 Staff Month.

Prototype:

Average productivity = 1375.8 SLOC/Staff Month

Lower bound = 1056/1375.8 = 76.8%

Upper bound = 1642.9/1375.8 = 119.4%

In summary, the productivity range of Specifying is from 76.8% to 119.4%. The productivity range of Specifying is from 64.7% to 137.5%. The lower bound of Effort is 1.06 Staff Month and the upper bound of effort is 1.85 Staff Month. Average effort is 1.52 Staff Month.

Table 6.6 Effort (Staff Month) of Specifying Method

Capability of PCAP, LTEX	Total Numbers	Mean(Average)	Standard Deviation
Normal	4	3.41	0.653
High	4	2.735	0.525

Table 6.7 Effort (Staff Month) of Prototype Method

Capability of PCAP, LTEX	Total Numbers	Mean(Average)	Standard Deviation
Normal	3	1.9	0.516
High	3	1.52	0.412

6.3 Conclusion

The distribution of Effort doesn't change much with higher capability of programmer but the average effort and standard deviation reduces greatly. Because the project in this example is too small, the scale factors are not included in the experiment.

However, the assumption in Section 6.2 proves the possibilities that changing factors in COCOMO may cause reduction of variance. Traditionally, there are several reasons that are believed to be responsible for the deviation between estimated and actual effort occurred. The reasons include but not limited to:

"Requirement changes

Unclear requirements

Additional requirements

Delay of decisions concerning requirements due to

Team members' lack of responsibility and motivation

Internal differences (due to political decisions)

Technical problems

Use of unknown technology

New and inexperienced team members

Change of technology

Occurrence of risks

Lack of sufficient customer communication

Unforeseen problems due to high complexity

Lack of qualified consultants"[30]

Among these reasons, some can be attributed to personal problems such as new or inexperienced team members. Most of the reasons can be reflected to the factors in COCOMO. If there is a “What-if” study based on the factors of COCOMO, it is possible to find the effect of target factor to help reducing the risk.

CHAPTER 7

MONTE CARLO PROBABILITY DISTRIBUTIONS

7.1 Simulation Design

Chapter 6 shows an example to reduce variance by higher capability of the programmer. However, better programmer capability is not the only factor to reduce variance. The scale factors can also have similar effect on variance. They are not discussed in Chapter 6 due to the size of the experiment project. In fact, all the scale factors and other effort multipliers can be considered as candidates to reduce variance. The problem is that how to find out the exact influence of each factor to help make a better solution and to reduce risk.

The reason why better individuals produce lower risk is that they are more productive, and higher productivity leads to smaller effort and variance. Therefore many managers prefer to have smarter people in their groups. But hiring better people is not always the best solution because it involves higher cost which should be also taken into consideration. Thus people are seeking alternatives such as improving process maturity to improve productivity and making such decisions has become a new challenge to the managers. If the effect of each method can be predicted and compared, the managers may have more helpful information to make a decision to reduce risk. And as it is explained in Chapter 6, Monte Carlo simulation is an ideal method to estimate the risks. In this chapter, the data from Figure 3.5 in Boehm's book[31] is used as an example to show how to use Monte Carlo simulation to calculate the risk.

The first step is to elicit project information:

Project Size: 44700 lines of code

Optimistic Productivity: 604.9 SLOC/Staff Month

Most Likely Productivity: 484 SLOC/Staff Month

Pessimistic Productivity: 387.2 SLOC/Staff Month

The range of productivity has included the uncertainties introduced by the effort multipliers, thus there is no need to involve effort multipliers in this simulation. The only thing should be cared about are the scale factors:

Precedentedness: 3.72

Development Flexibility: 3.04

Architecture/Risk resolution: 4.24

Team Cohesion: 3.29

Process Maturity: 4.68

All of above scale factors are rated as nominal value, and the distribution of PMAT is set from 54% to 146% based on the data in chapter 5.

The second step is to determine the formulas:

Process Maturity = RiskTriang(0.54,1,1.46) * 4.68

Exponent E = 0.91 + 0.01 * (SF1+SF2+SF3+SF4+SF5)

Exponent F = 0.28 + 0.2 * (E - 0.91)

Productivity A = RiskTriang(1000/604.9,1000/484,1000/387.2)

Effort = RiskOutput(“Effort in Staff Month”) + A * 44.7^E

Schedule = RiskOutput(“Schedule in Month”) + 3.67 * Effort^F

Finally, input above information to @Risk and get ready to run Monte Carlo simulation.

7.2 Implementation

With the information and formulas discussed in Section 7.1, @Risk can perform a complete Monte Carlo simulation and the iterations are set to 5000 times. The original result is shown below:

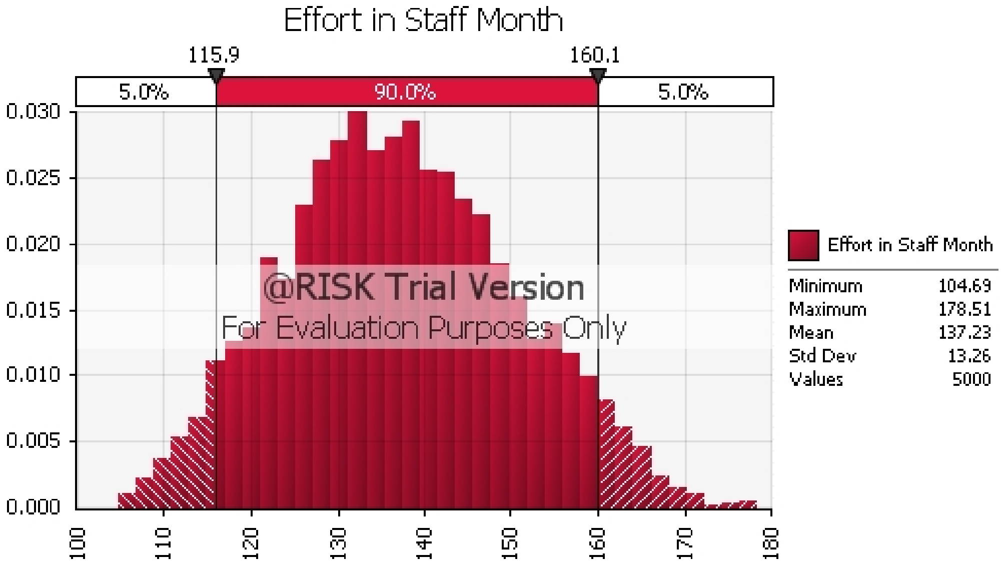


Figure 7.1 Original effort distribution. The iteration time of this simulation is 5000. Minimum effort is 104.69. Maximum effort is 178.51. Mean effort is 137.23. Standard deviation is 13.26.

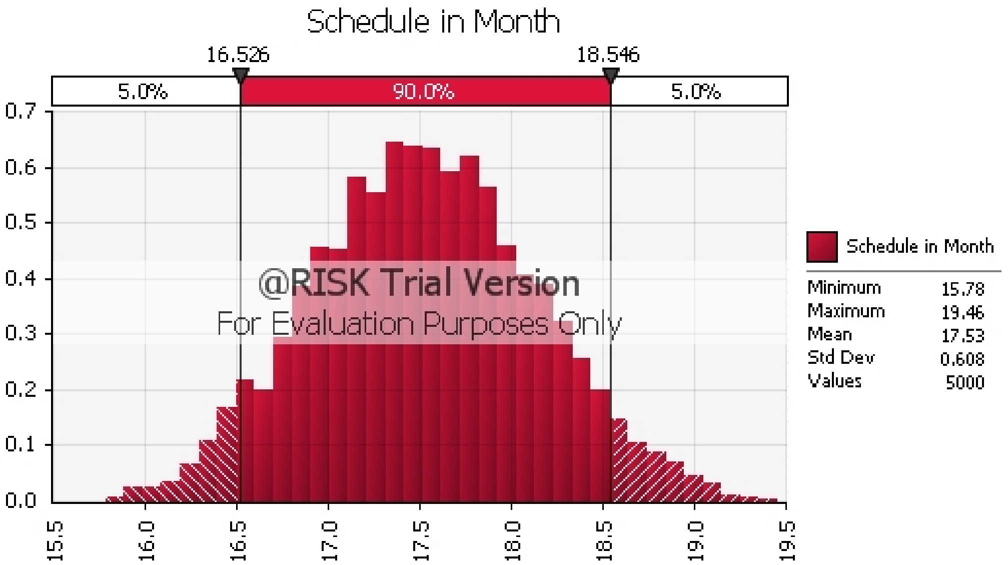


Figure 7.2 Original schedule distribution. The iteration time of this simulation is 5000. Minimum schedule is 15.78 months. Maximum schedule is 19.46 months. Mean schedule is 17.53. Standard deviation is 0.608.

Because traditional COCOMO estimation is point estimate, it won't show such distribution but the result is very close to the mean value in Figure 7.1 and Figure 7.2. Here is the comparison between Boehm's estimation and Monte Carlo simulation:

Table 7.1 Comparison Between Boehm's Data and Monte Carlo Simulation

	Effort		Schedule	
	Boehm	Monte Carlo	Boehm	Monte Carlo
Optimistic	73.9	104.69	13.7	15.78
Most Likely	92.4	137.23	14.7	17.53
Pessimistic	115.5	178.51	15.7	19.46

Source: Barry W. Boehm, Software Cost Estimation with COCOMO II, pp.15, Figure 3.5, Prentice-Hall, INC., 2000[30]

In Boehm's data, his estimation has a smaller value because he doesn't include the scale factors in the calculation. Maybe Boehm has already included the effect of scale factors in productivity range. But in this experiment it is assumed that the scale factors still have influence on the estimation.

The standard deviation of effort and schedule are 13.26 and 0.608 and the goal is to find methods to reduce standard deviation. In Chapter 6, it is proved that programmer capability can reduce variance by producing less effort. Compared to programmer capability, how will Process Maturity help reduce variance? Thus it is required to observe the change of standard deviation when different productivity or Process Maturity is applied to Monte Carlo simulation. For example, it can use different productivity in simulation without changing the distribution of the process maturity. Here is an experiment that shows the change of standard deviation when productivity is increased from 100% to 140%.

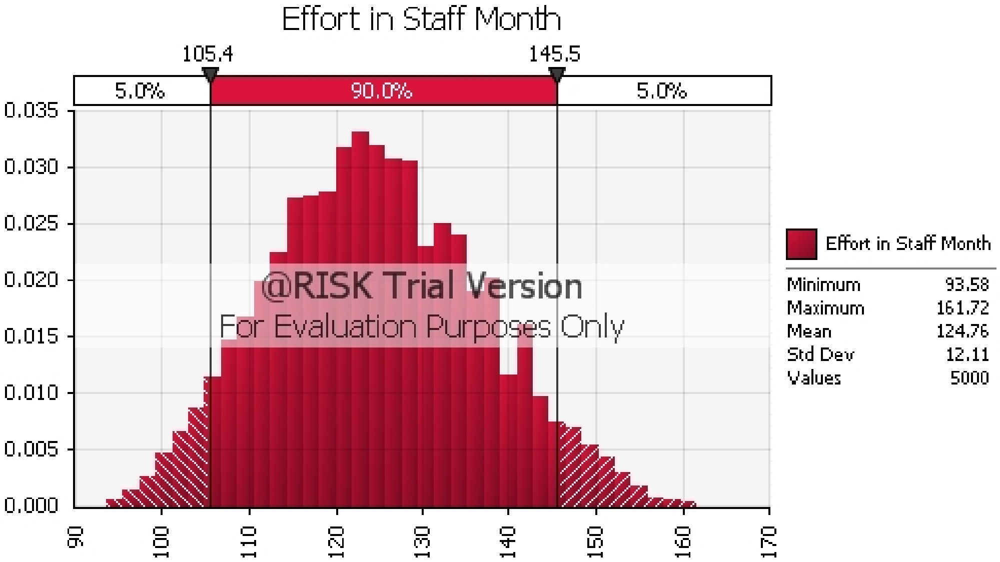


Figure 7.3 Effort distribution when productivity is increased by 10%.

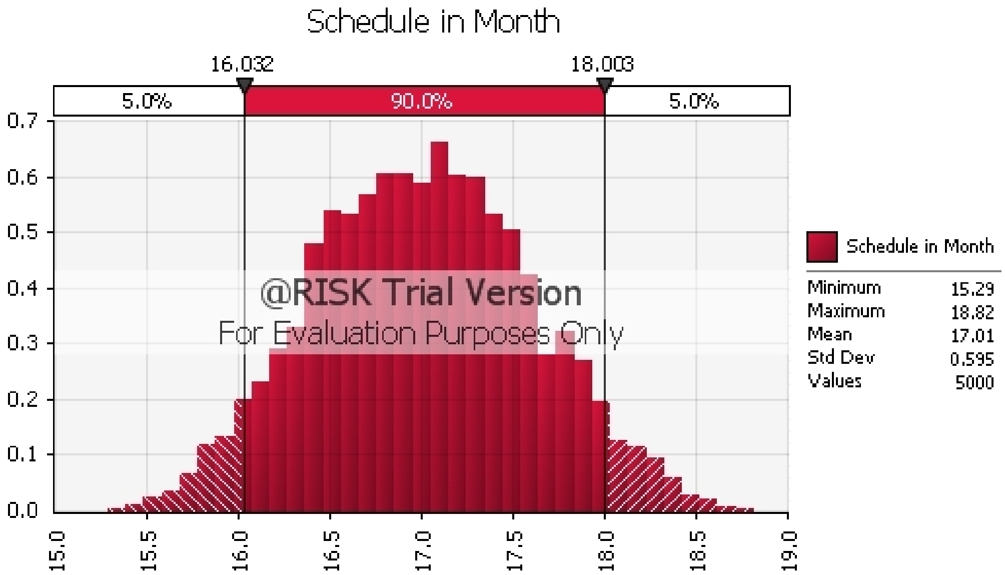


Figure 7.4 Schedule distribution when productivity is increased by 10%.

When productivity is increased by 10% to 532 SLOC/PM, mean effort is 124.76 staff months which is reduced by 10% respectively. And mean schedule is 17.01 months which is reduced by 3%. These numbers are very close to the normal calculation of COCOMO (without variance). The standard deviation of effort and schedule is 12.11 staff months and 0.595 months.

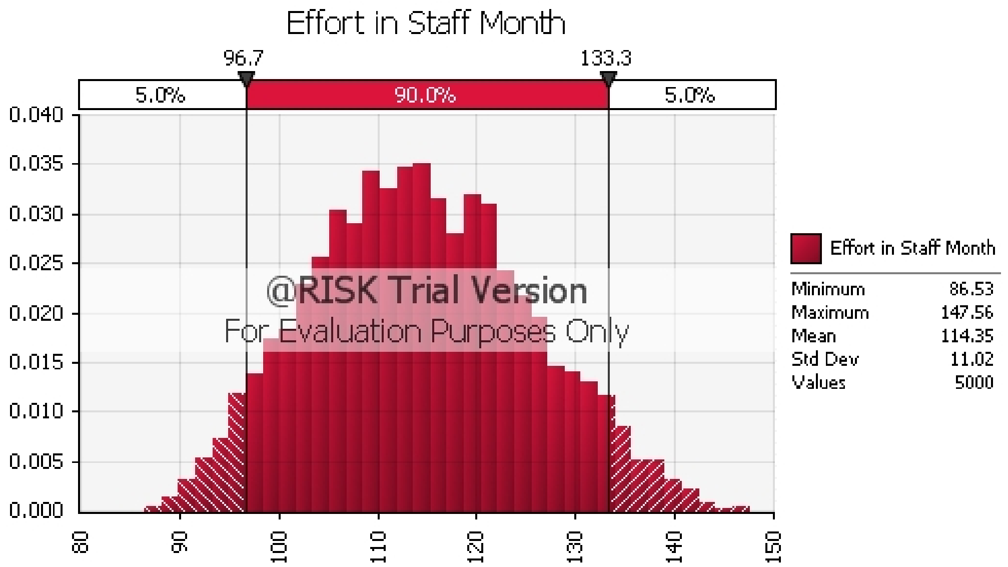


Figure 7.5 Effort distribution when productivity is increased by 20%.

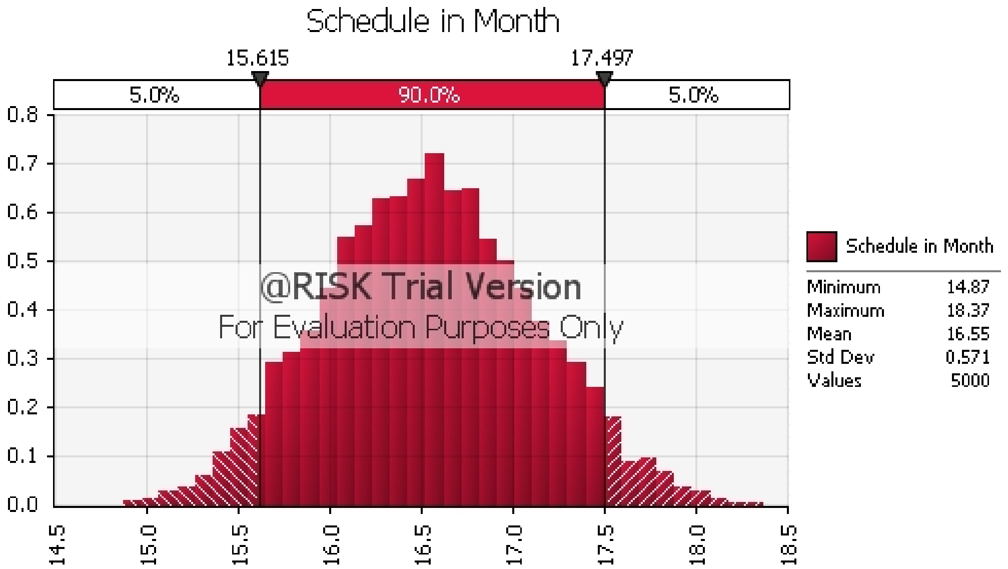


Figure 7.6 Schedule distribution when productivity is increased by 20%.

When productivity is increased by 20% to 580 SLOC/PM, mean effort is 114.35 staff months which is reduced by 20% respectively, and mean schedule is 16.55 months which is reduced by 5.6%. The standard deviation of effort and schedule is 11.02 staff months and 0.571 months.

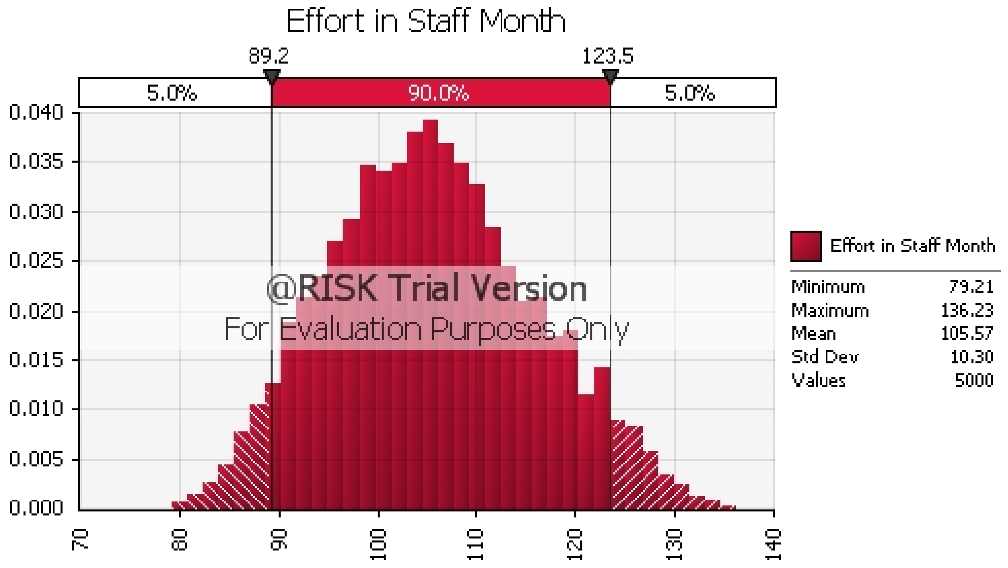


Figure 7.7 Effort distribution when productivity is increased by 30%.

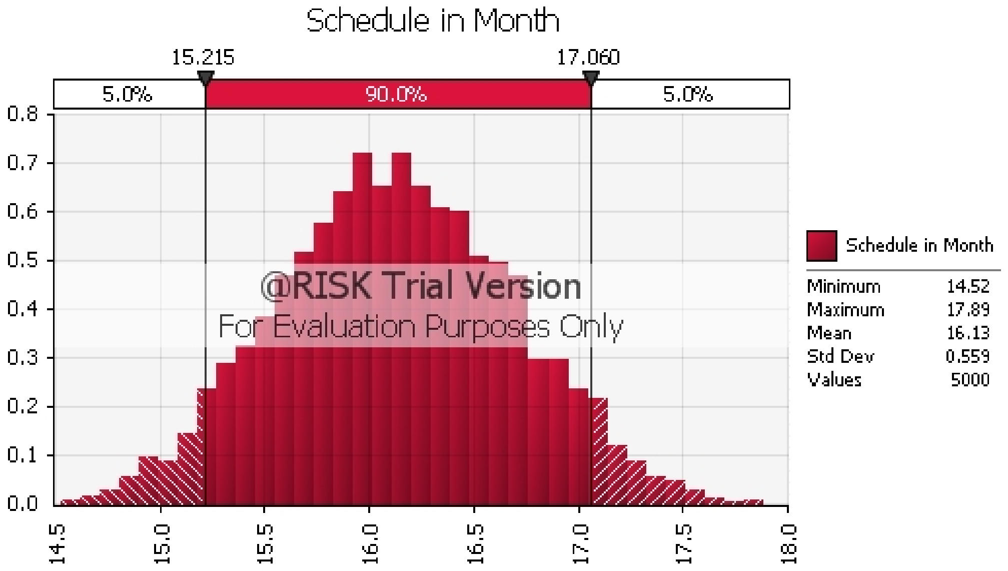


Figure 7.8 Schedule distribution when productivity is increased by 30%.

When productivity is increased by 30% to 629 SLOC/PM, mean effort is 105.57 staff months which is reduced by 30% respectively. And mean schedule is 16.13 months which is reduced by 8%. The standard deviation of effort and schedule is 10.30 staff months and 0.559 months.

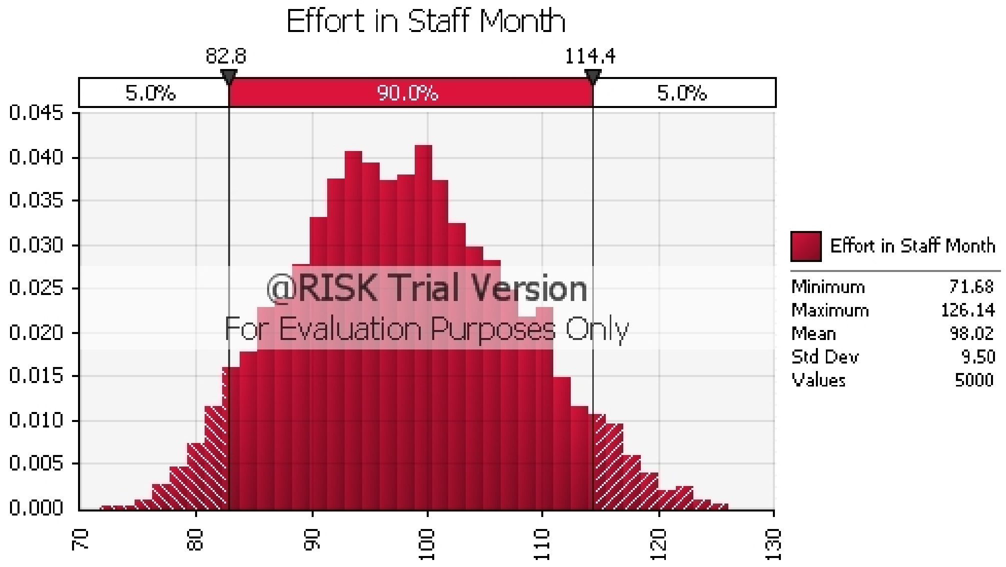


Figure 7.9 Effort distribution when productivity is increased by 40%.

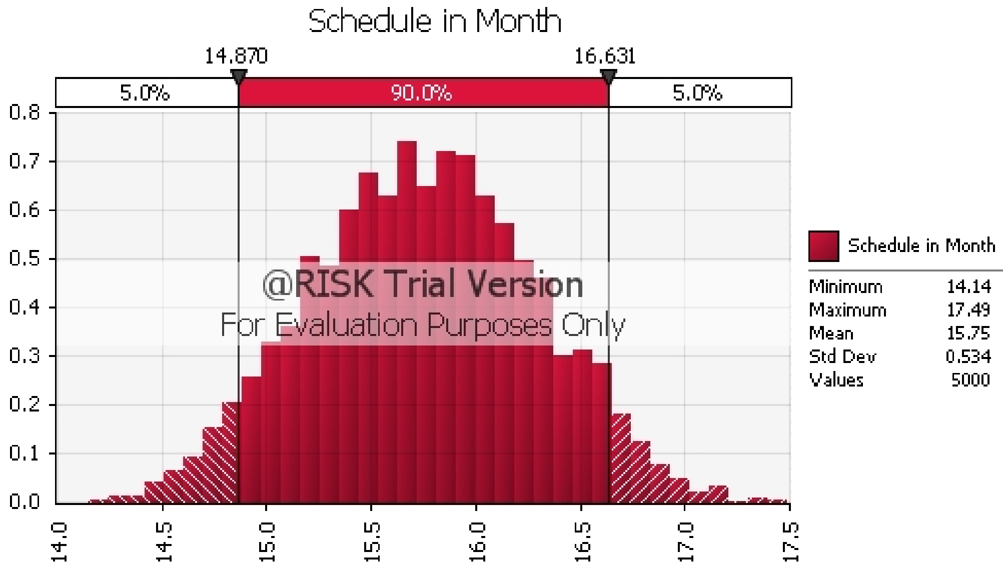


Figure 7.10 Schedule distribution when productivity is increased by 40%.

When productivity is increased by 40% to 678 SLOC/PM, mean effort is 98.02 staff months which is reduced by 40% respectively. And mean schedule is 15.75 months which is reduced by 10% and standard deviation is reduced by 12%. The standard deviation of effort and schedule is 9.5 staff months and 0.534 months. Therefore, the effort is disproportionately affected by increasing productivity. But schedule shows an apparent diseconomy of scale when productivity increases.

Next there is another experiment on how the Process Maturity can impact the variance of effort and schedule. In this study, the standard deviation is traced when Process Maturity is improved from 4 to 1.

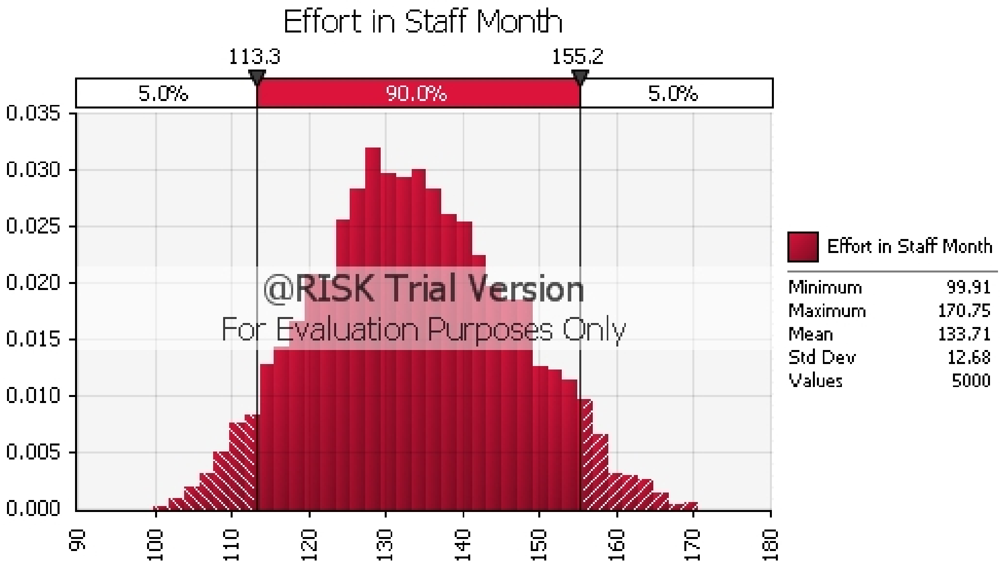


Figure 7.11 Effort distribution when PMAT is improved to 4.

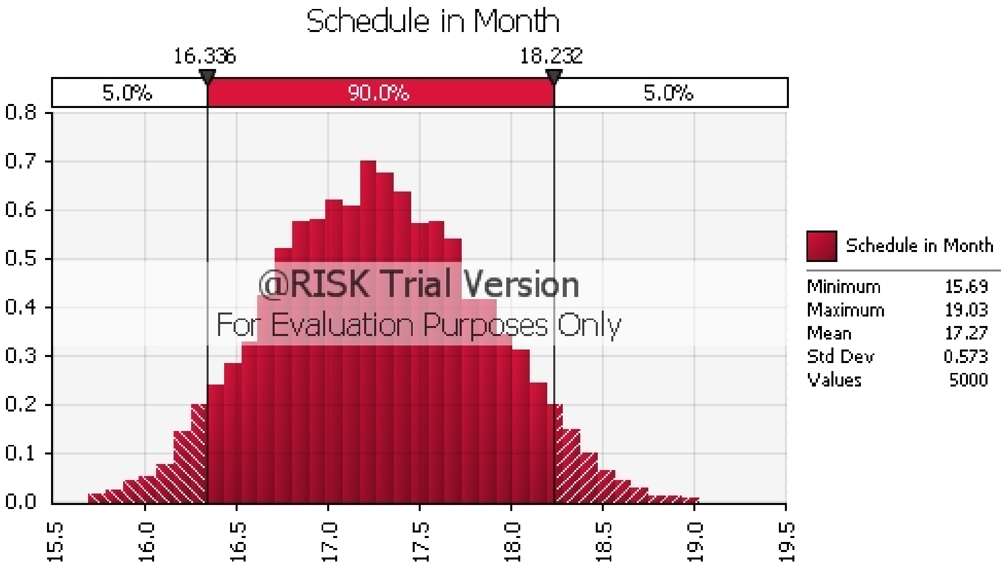


Figure 7.12 Schedule distribution when PMAT is improved to 4.

When Process Maturity is 4, mean effort is 133.71 Staff Month and standard deviation is 12.68. Mean schedule is 17.27 and standard deviation is 0.573.

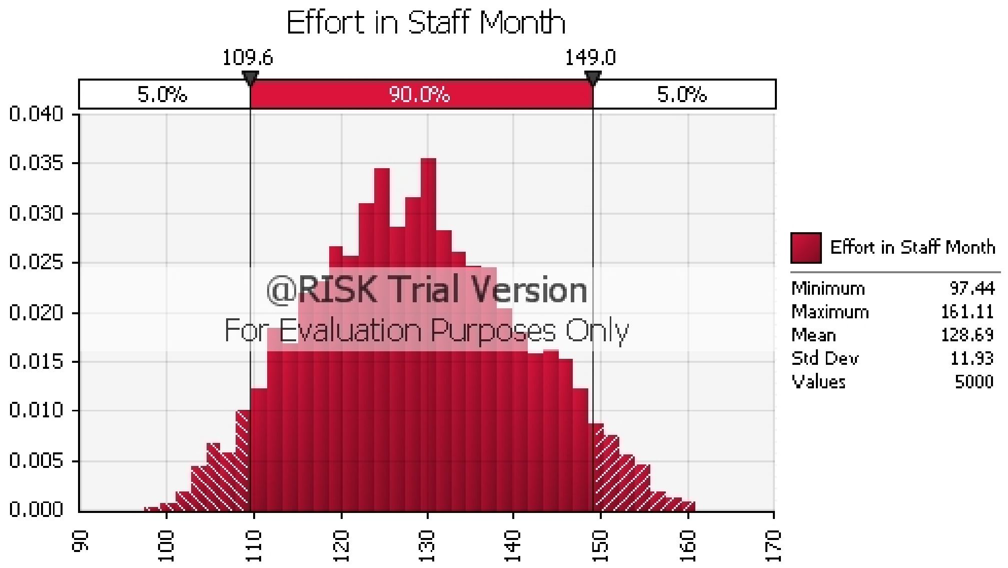


Figure 7.13 Effort distribution when PMAT is improved to 3.

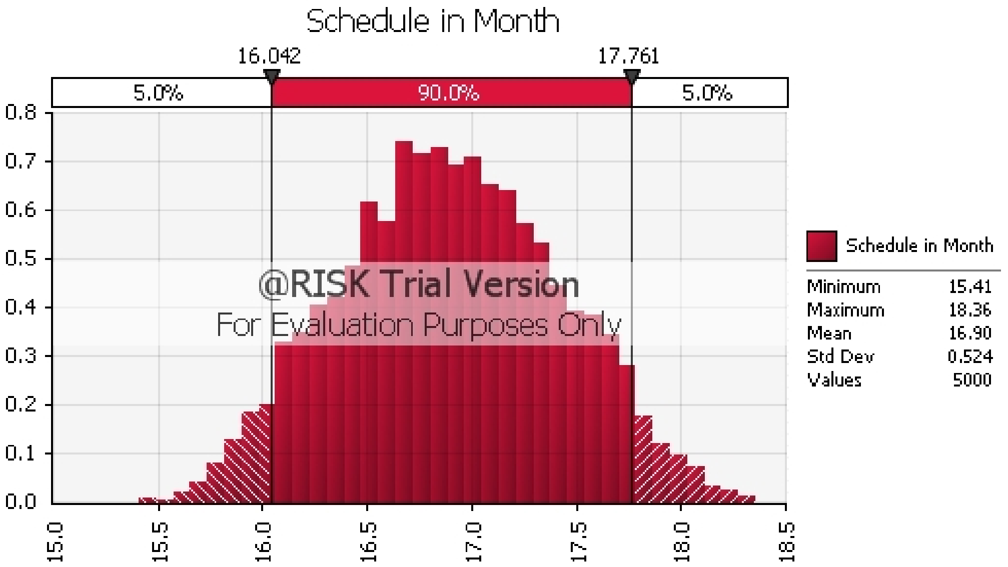


Figure 7.14 Schedule distribution when PMAT is improved to 3.

When Process Maturity is 3, mean effort is 128.69 Staff Month and standard deviation is 11.93. Mean schedule is 16.90 and standard deviation is 0.524.

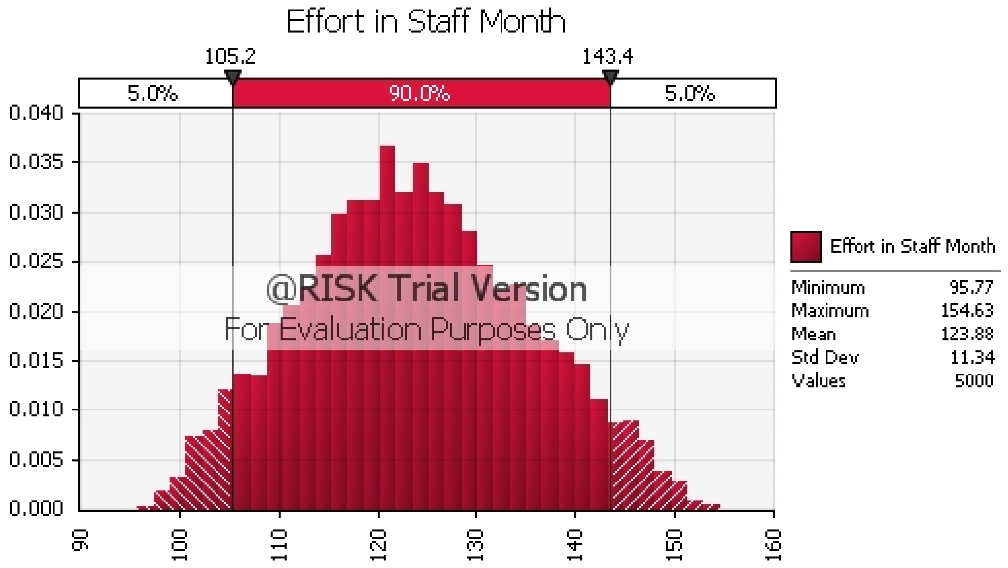


Figure 7.15 Effort distribution when PMAT is improved to 2.

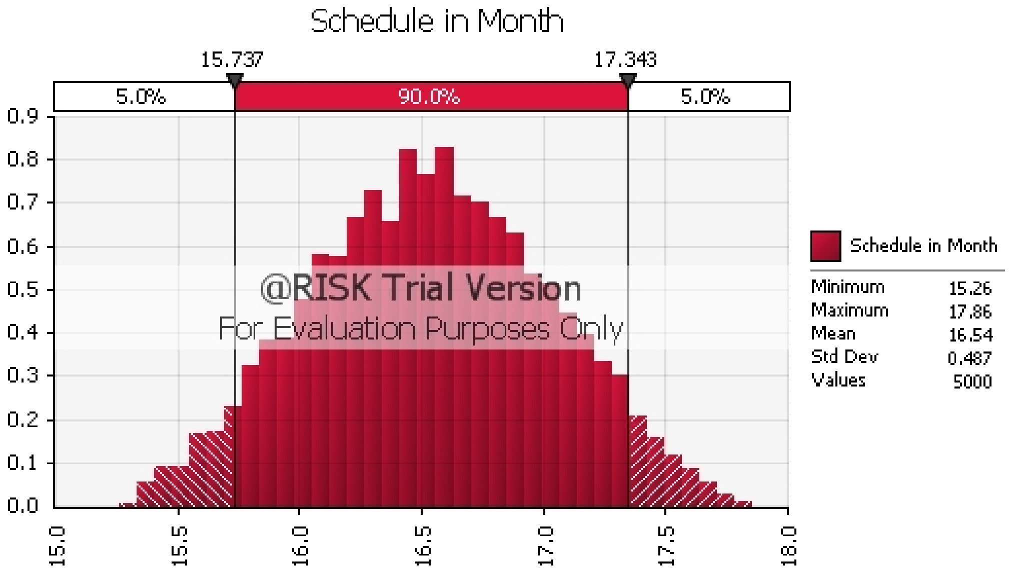


Figure 7.16 Schedule distribution when PMAT is improved to 2.

When Process Maturity is 2, mean effort is 123.88 Staff Month and standard deviation is 11.34. Mean schedule is 16.54 and standard deviation is 0.487.

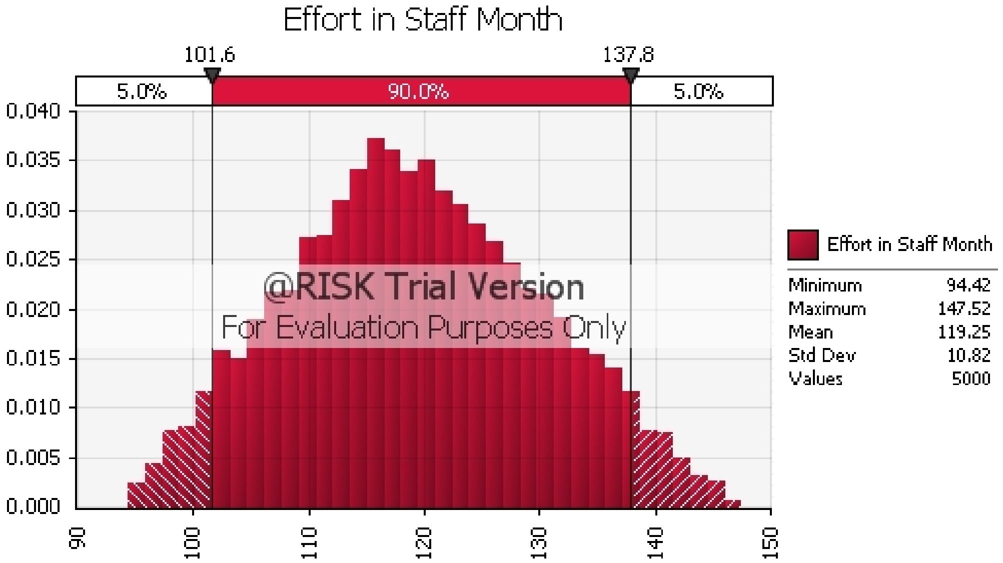


Figure 7.17 Effort distribution when PMAT is improved to 1.

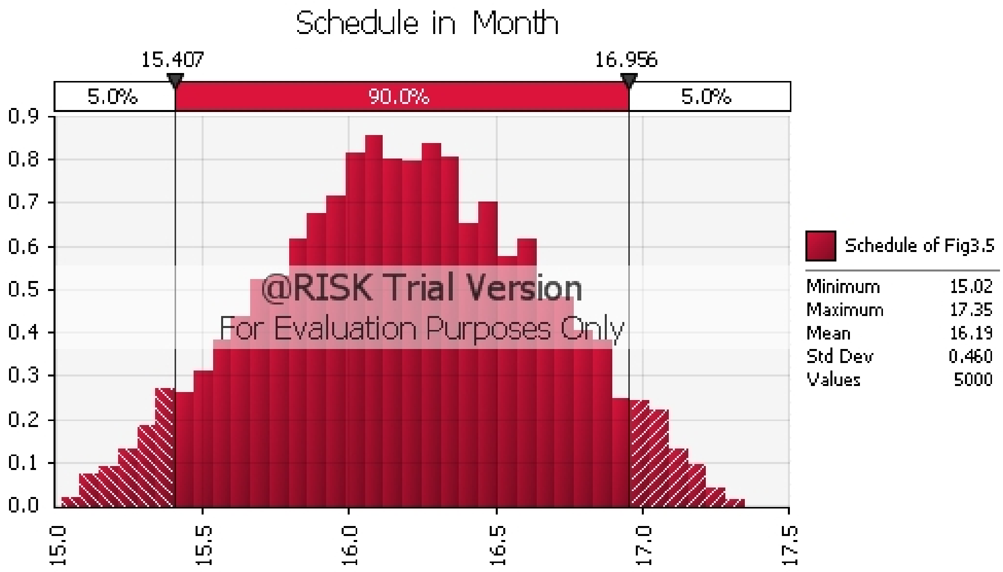


Figure 7.18 Schedule distribution when PMAT is improved to 1.

When Process Maturity is 1, mean effort is 119.25 Staff Month and standard deviation is 10.82. Mean schedule is 16.19 and standard deviation is 0.460. Mean schedule is reduced by 7.6% and standard deviation is reduced by 24%.

Compared to productivity, Process Maturity has smaller influence on effort but it is more likely to reduce standard deviation.

7.3 Results and Conclusion

Due to the result of Monte Carlo simulation, both productivity and PMAT can reduce risk of estimation. But they have different effects on effort and schedule. Here is the summary:

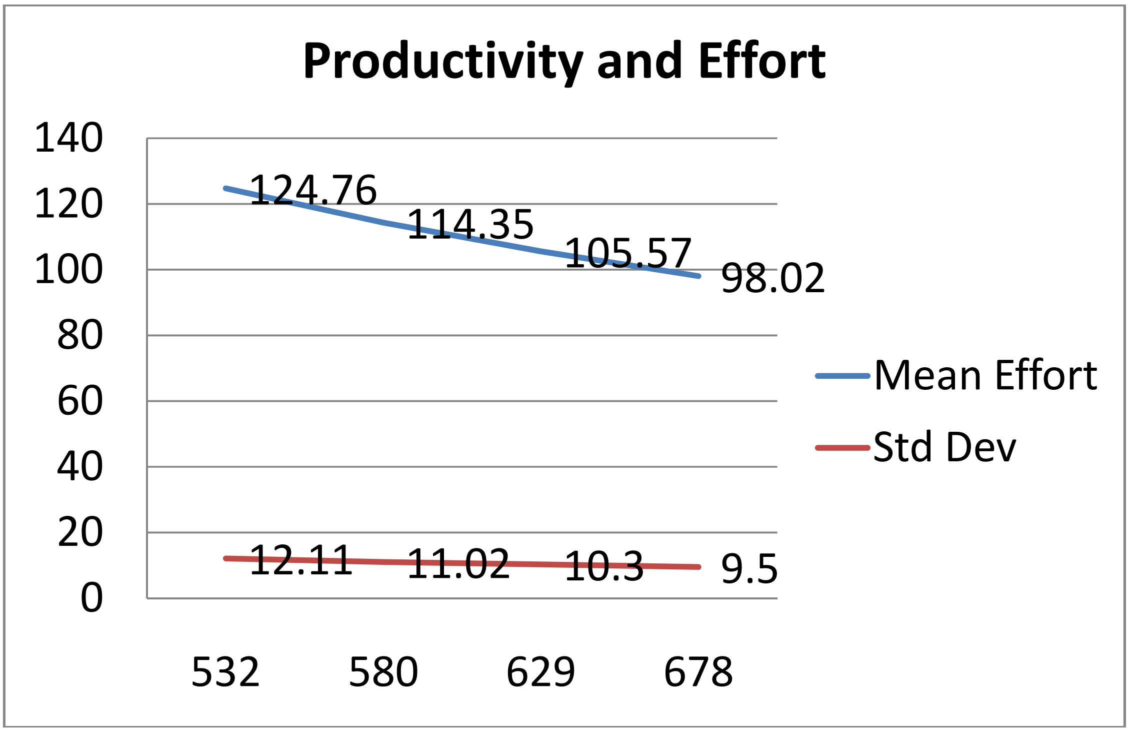


Figure 7.19 Relationship between productivity and effort. When productivity is increased from 532 SLOC/Staff Month to 678 SLOC/Staff Month, Effort is reduced from 124.76 staff month to 98.02 staff month.

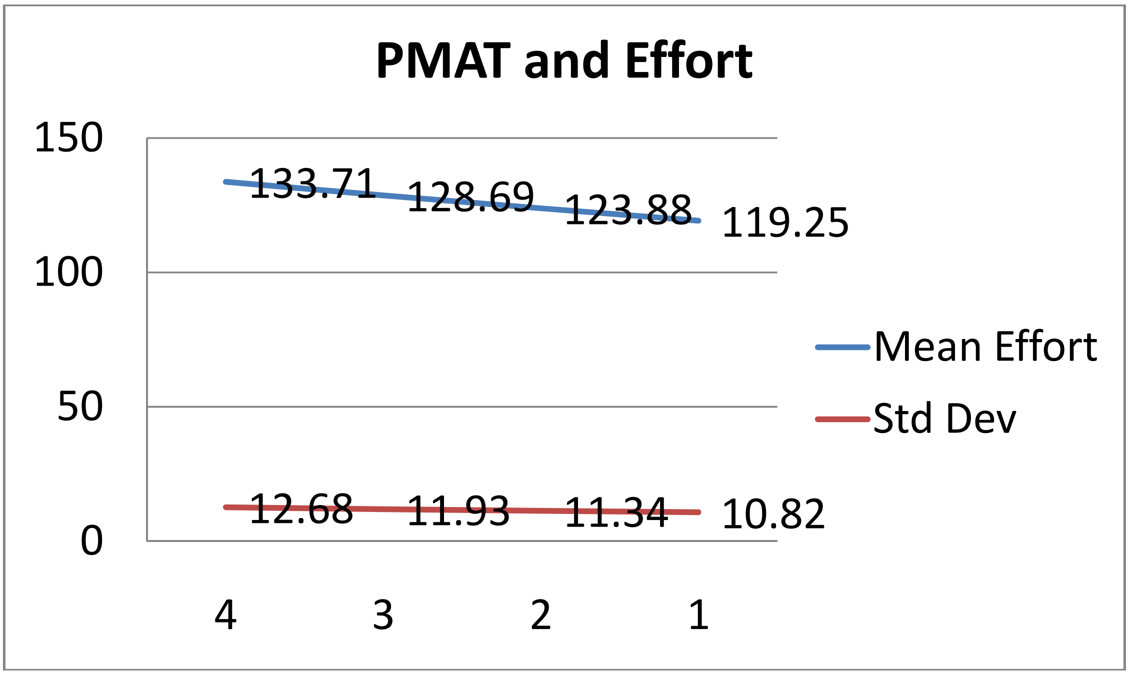


Figure 7.20 Relationship between PMAT and effort. When PMAT is improved from 4 to 1, Effort is reduced from 133.71 staff month to 119.25 staff month.

According to Figure 7.19 and Figure 7.20, when productivity reduces effort by 21%, it can reduce same proportion to standard deviation. When PMAT reduces effort by 11%, it can reduce standard deviation by 15%.

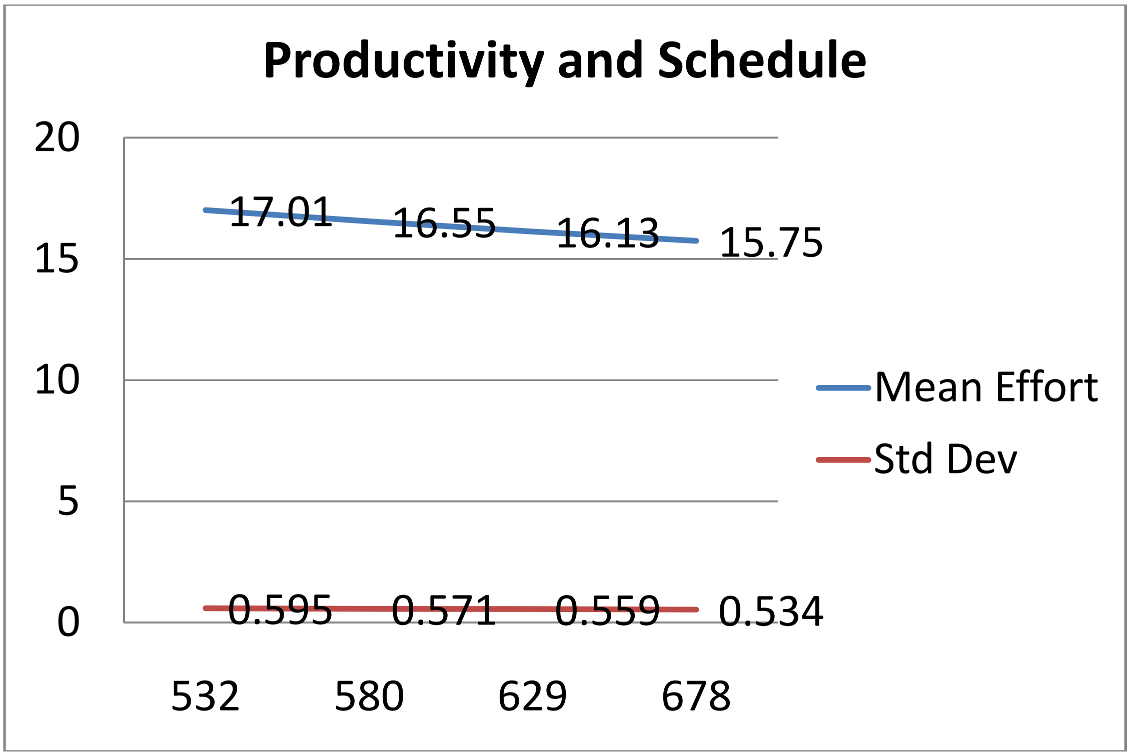


Figure 7.21 Relationship between productivity and schedule. When productivity is increased from 532 SLOC/Staff Month to 678 SLOC/Staff Month, schedule is reduced from 17.01 month to 15.75 month.

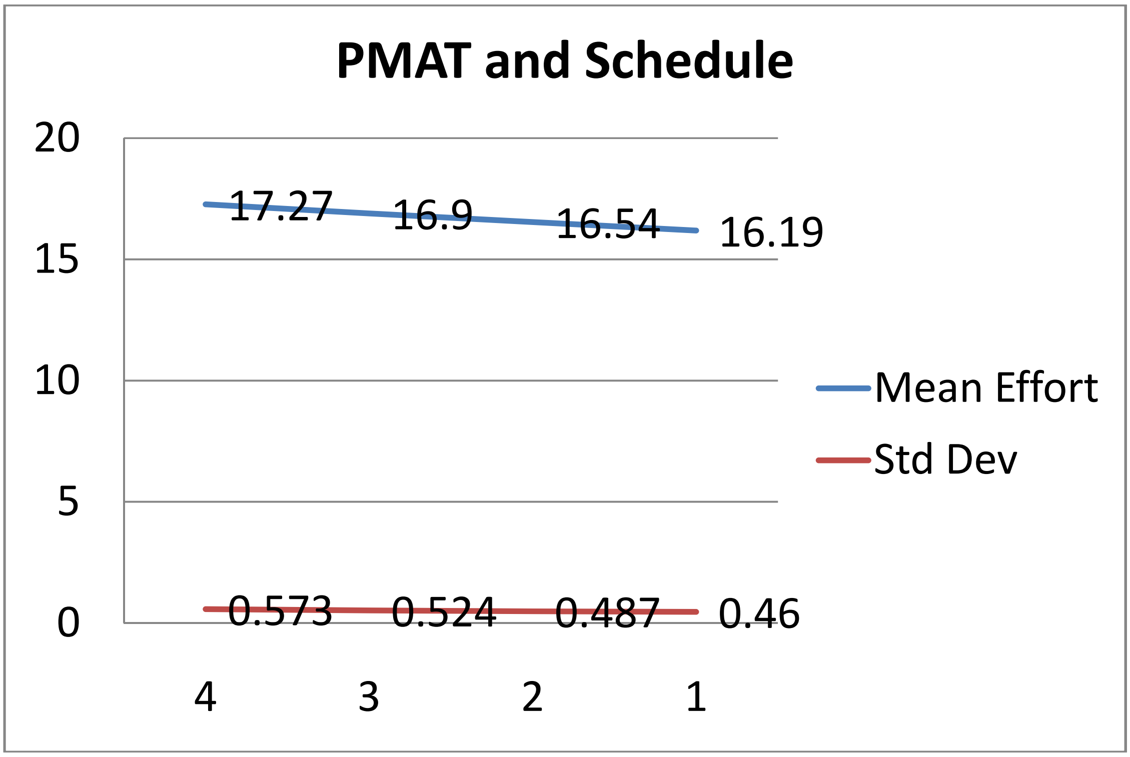


Figure 7.22 Relationship between PMAT and Schedule. When PMAT is improved from 4 to 1, schedule is reduced from 17.27 month to 16.19 month.

According to Figure 7.21 and Figure 7.22, when productivity reduces schedule by 7%, it can reduce standard deviation by 10%. When PMAT reduces schedule by 6%, it can reduce standard deviation by 20%.

In summary, productivity has more impact on reducing mean value of effort and schedule while better PMAT is more effective to reduce standard deviation.

CHAPTER 8

EPILOGUE

> "Inaccuracy of cost estimation is well known. Appendix B is a recent paper by the SEI proposing the use of Bayesian Belief Network (BBN) models and Monte Carlo simulation that quantifies Uncertainty. This is similar to Yang's thesis except that the SEI approach concentrates on early life cycle cost estimates and not on periodic measures of estimation variance throughout the development. The variance is used as the measure of risk and the model is used to study the effects of cost driver changes on the variance calculations. The SEI technical report was issued in December 2011, Yang became aware of it when it was published in the Journal of Software Technology in Feb. 2012. Yang started work on his thesis in July 2011. While the processes are surprising similar the goals are different and the approaches complement each other."

by Larry Bernstein, NJIT Software Engineering Adjunct Professor

The SEI technical report[32] shows a similar idea that people are aware about the risks in estimation and Monte Carlo simulation could be an ideal tool to help people find the risks. It introduced the QUELCE (Quantifying Uncertainty in Early Cost Estimation) method and it contains the following steps:

"Step 1: Identify Program Change Drivers

Step 2: Identify States of Program Change Drivers

Step 3: Identify Cause-and-Effect Relationships for Dependency Matrix

Step 4: Reduce the Dependency Matrix Using a Design Structure Matrix

Step 5: Construct a BBN Using the Reduced Matrix

Step 6: Populate BBN Nodes with Conditional Probabilities

Step 7: Define Scenarios by Altering Program Change Driver Probabilities

Step 8: Select Cost Estimating Relationships or Tools to Generate an Estimate

Step 9: Obtain Program Estimates Not Computed by the BBN

Step 10: For Each Scenario, Run a Monte Carlo Simulation

Step 11: Report Each Scenario Result Independently"[33]

The SEI's technical report includes scenario building and Bayesian Belief Network (BBN) modeling. Bayesian Belief Network is a very useful modeling method to calibrate risk and the author was thinking about adding it to his thesis at beginning but it was removed because the thesis was focused on an engineering phase. The goal of this thesis is to find a practical way to calibrate the risk in COCOMO and an engineering solution to reduce the risk. The solution is aimed to be as simple as possible, such as making a choice between hiring better developers and improving developing process. A practitioner never wants the things to become complicated. That's why COCOMO was chosen to be the basic cost estimate model in this thesis and spend more effort on introducing the tool of Monte Carlo simulation rather than the estimation and statistics theories.

QUELCE is a very detailed method to quantify uncertainty in early lifecycle cost estimation. However, no one can predict the change in a project in the later phases. That's why "Management processes need to account for moving targets" and "Effort needs to be re-estimated when change requests occur"[30]. Therefore, to monitor the status of the project, the method mentioned in this thesis should be repeated when any change occurs. If the result shows any reduction of the risk, it means the accuracy of estimation has been improved. Admittedly, QUELCE is a more complex method that may provide more explicit, quantified result of uncertainty. The method in this thesis may take advantage of

QUELCE in the further research. Combining other probabilistic methods such as Bayesian Belief Network and Monte Carlo simulation, traditional cost estimates will gain more improvements in accuracy and risk control.

APENDIX A

SAMPLES OF DISTRIBUTION OF EFFORT MULTIPLIERS

Figure A.1 to A.13 show distribution of Effort Multipliers in COCOMO, which is based on the data of COCOMO 81.

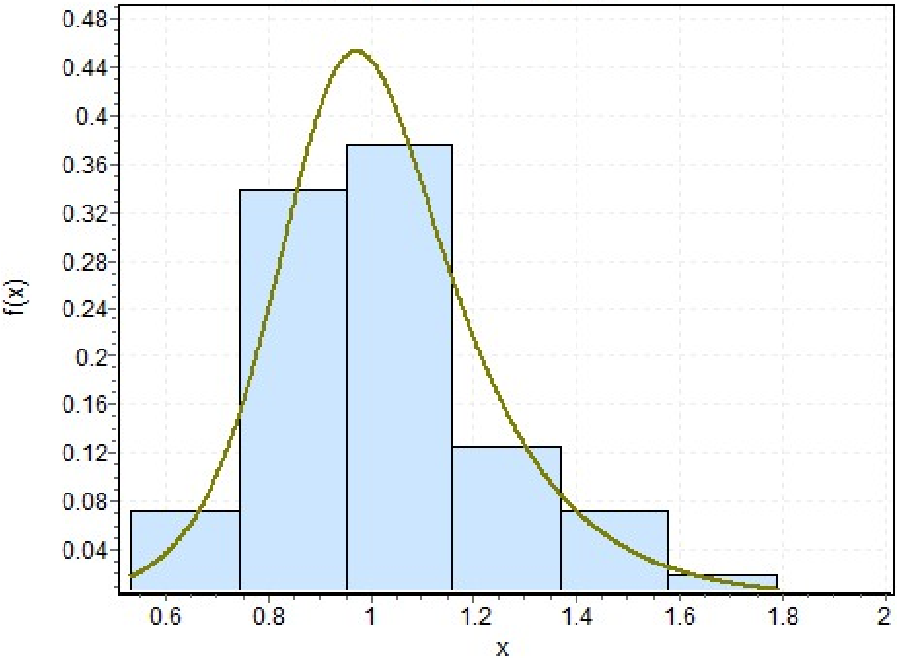


Figure A.1 Distribution of database size

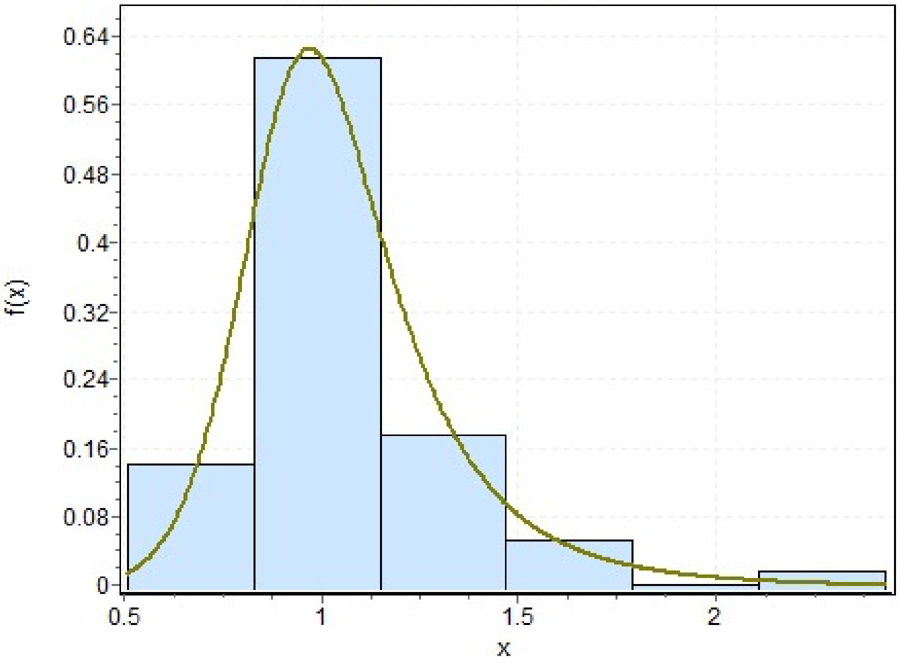


Figure A.2 Distribution of product complexity

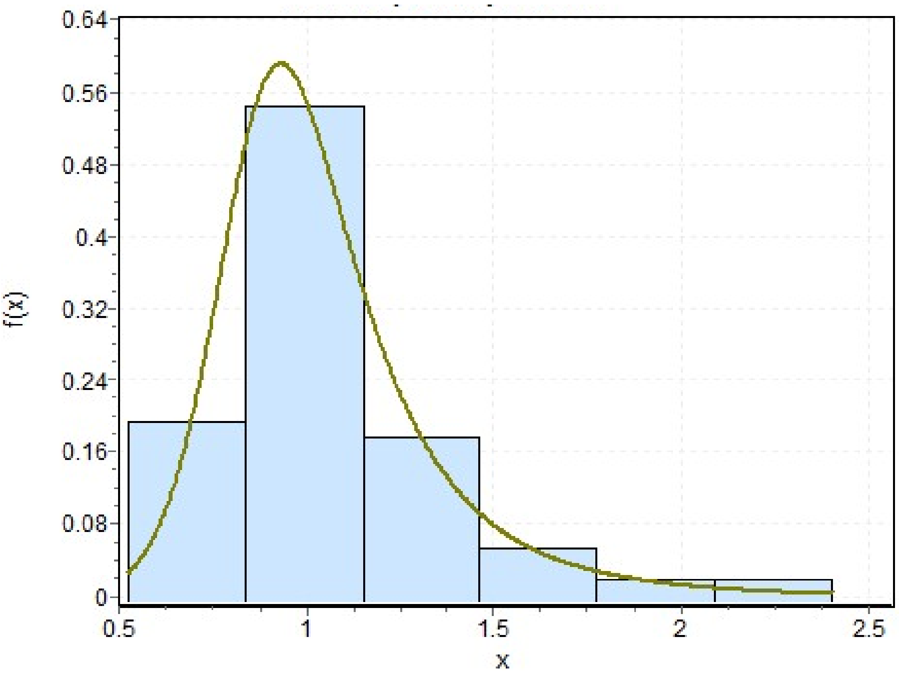


Figure A.3 Distribution of execution time constraint

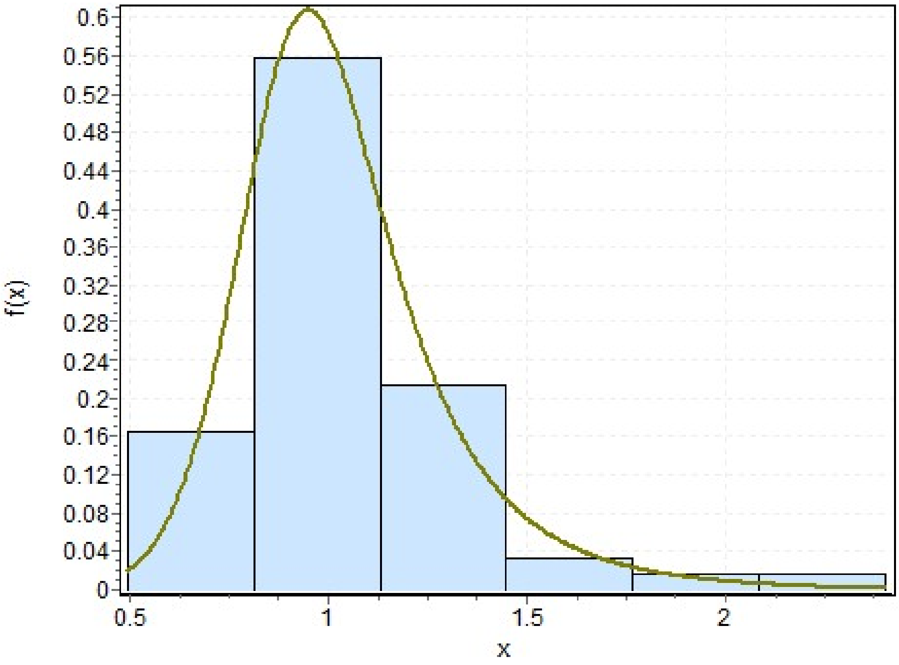


Figure A.4 Distribution of main storage constraint

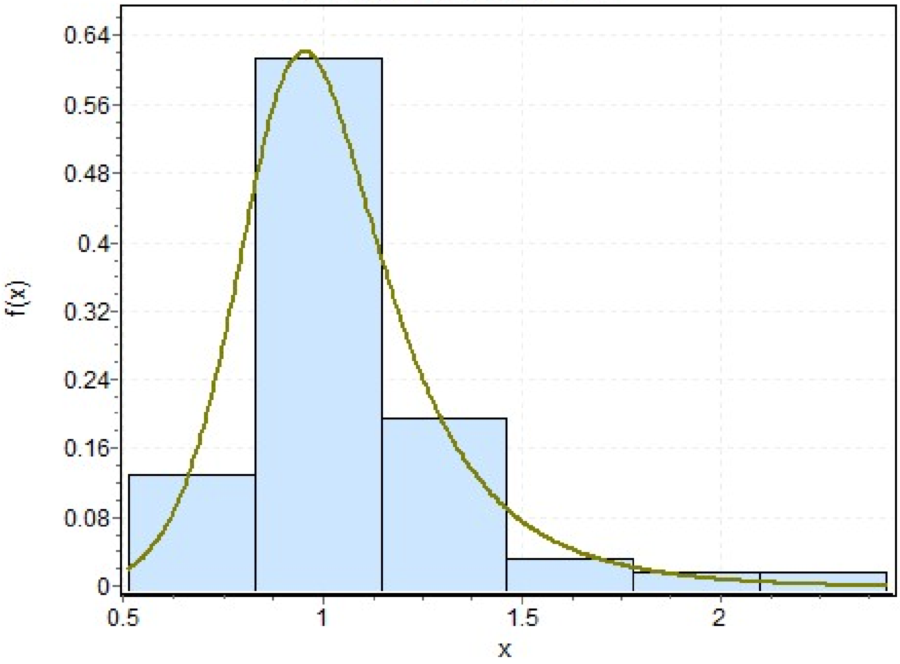


Figure A.5 Distribution of virtual machine volatility

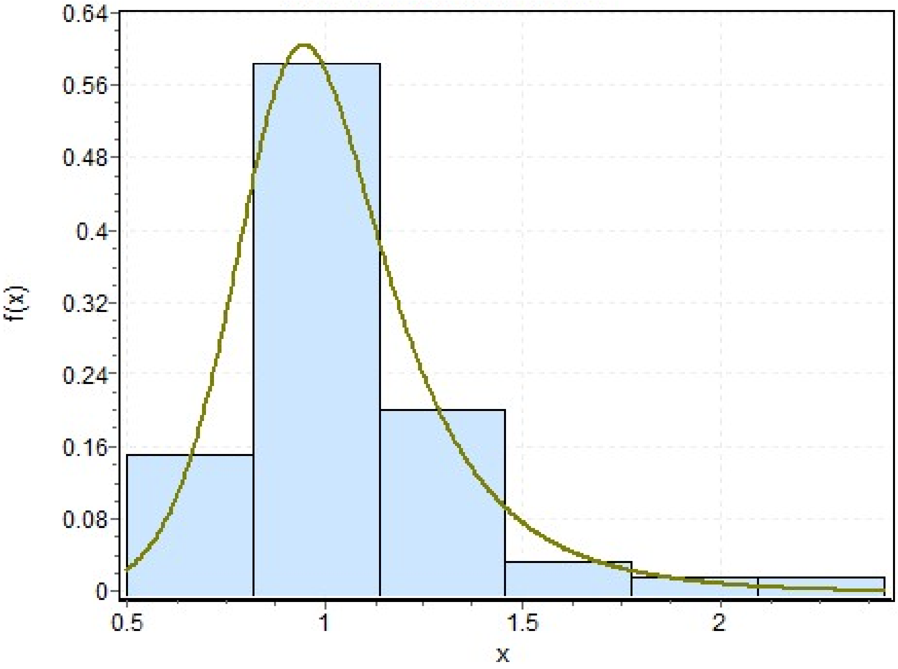


Figure A.6 Distribution of computer turnaround time

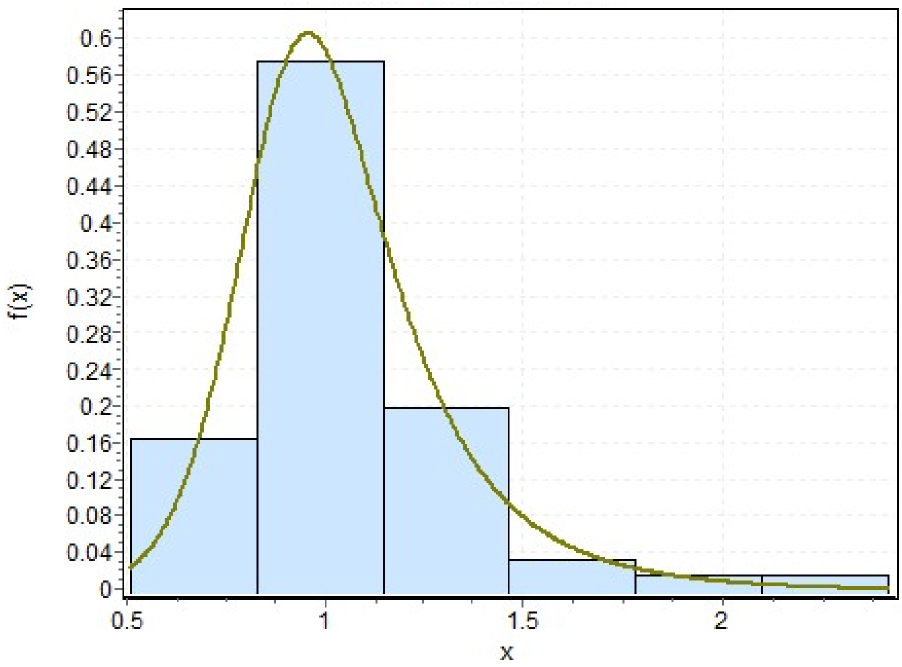


Figure A.7 Distribution of analyst capability

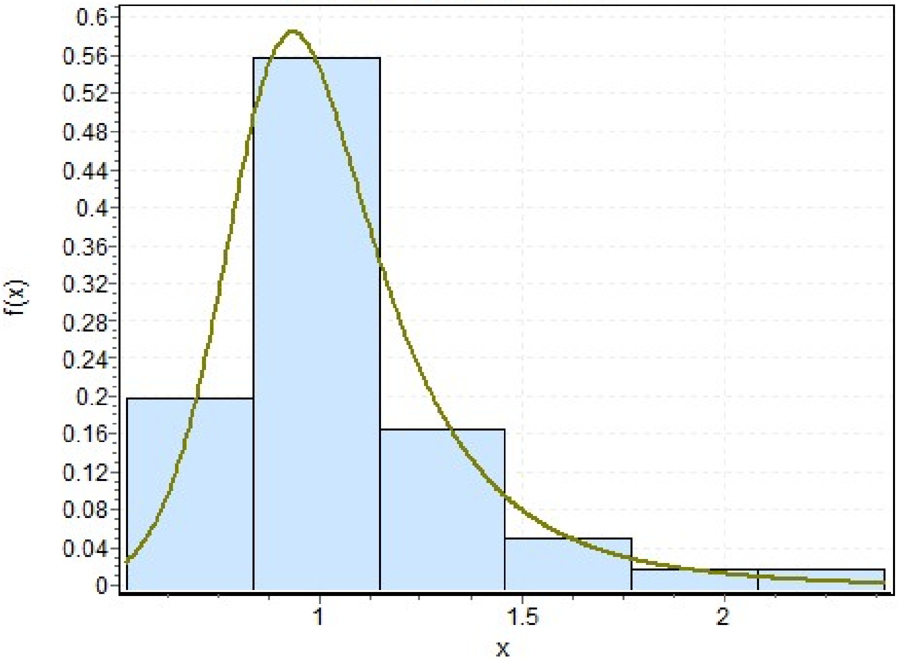


Figure A.8 Distribution of application experience

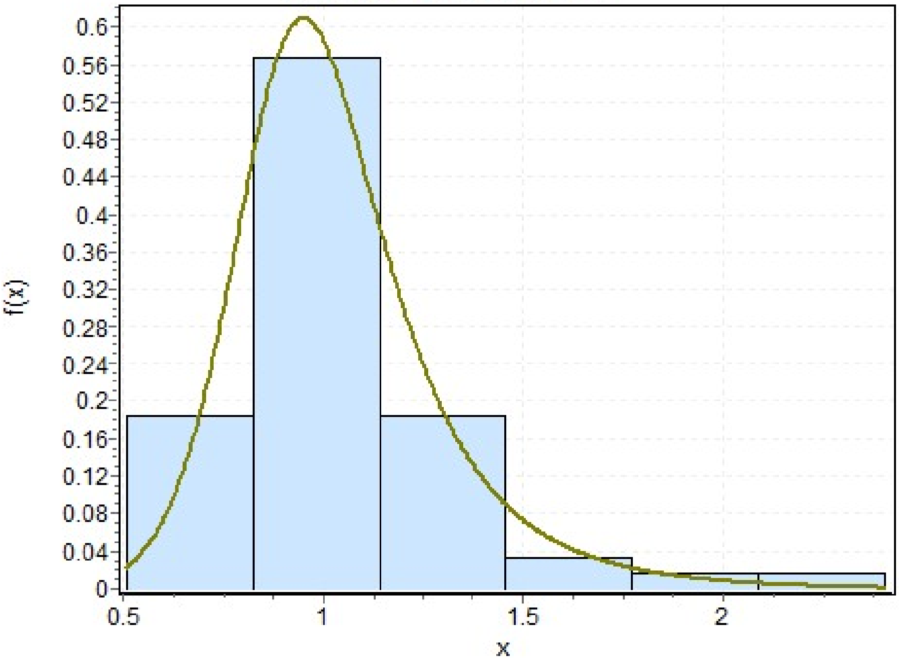


Figure A.9 Distribution of programmer capability

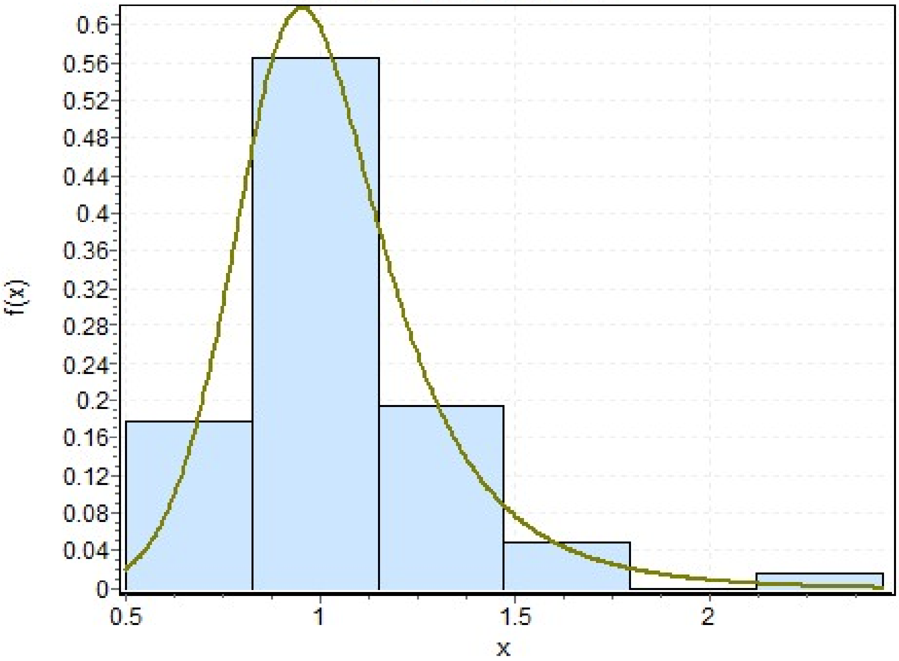


Figure A.10 Distribution of virtual machine experience

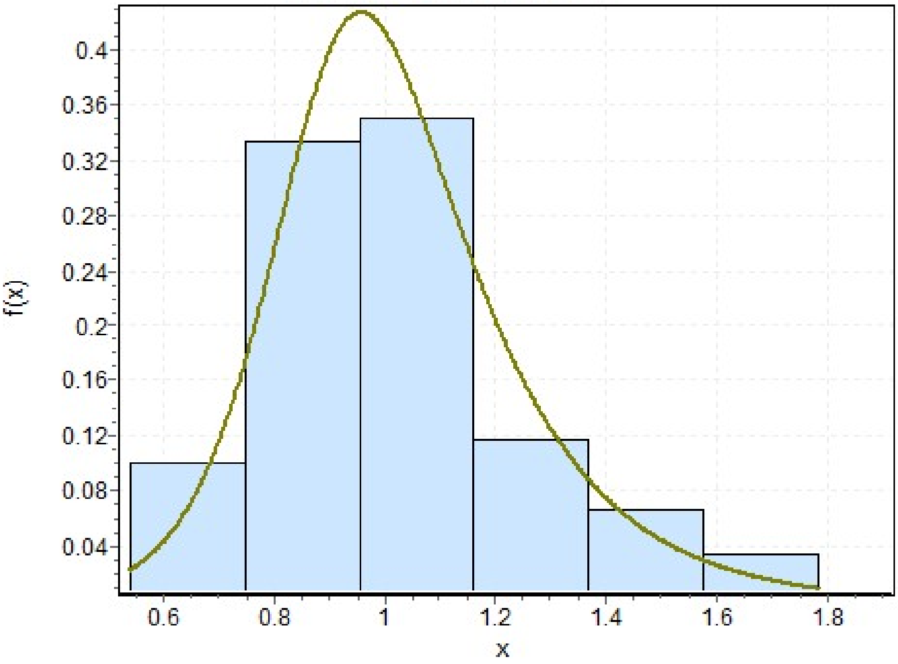


Figure A.11 Distribution of programming language experience

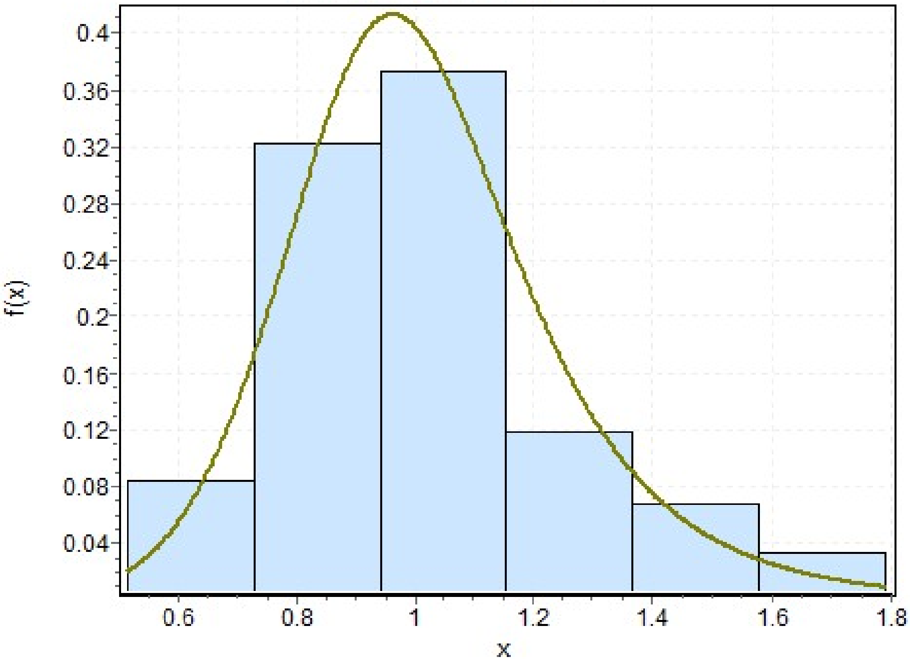


Figure A.12 Distribution of modern programming practices

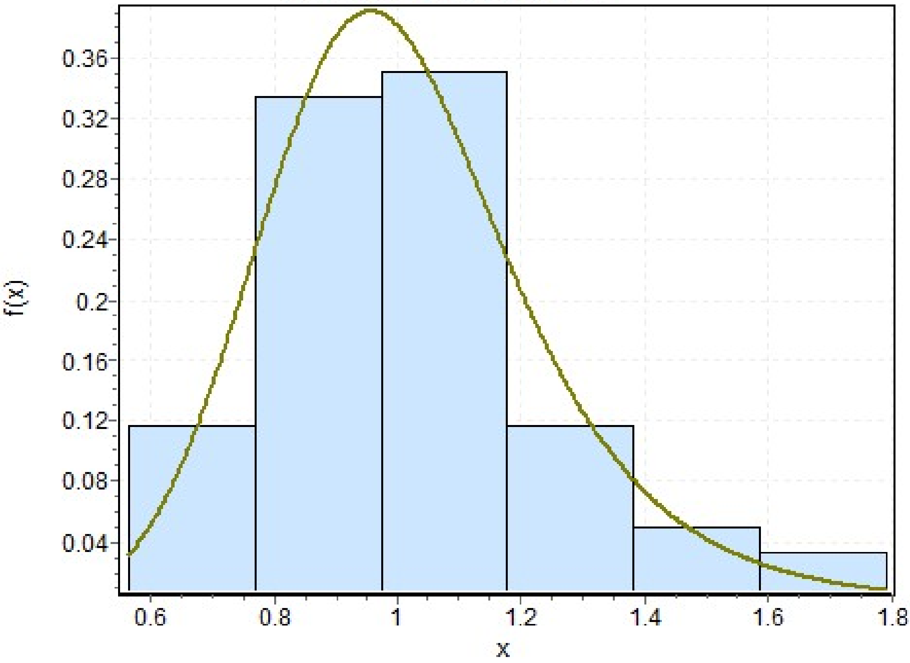


Figure A.13 Distribution of use of software Tools

APPENDIX B

AN INNOVATIVE APPROACH TO QUANTIFYING UNCERTAINTY IN EARLY LIFECYCLE COST ESTIMATION

By SEI Cost Estimation Research Group: Robert Ferguson, Dennis Goldenson, James McCurley, Robert Stoddard, and David Zubrow. Reproduced from reference[33].

The inaccuracy of cost estimates for developing major Department of Defense (DoD) systems is well documented, and cost overruns have been a common problem that continues to worsen. Because estimates are now prepared much earlier in the acquisition lifecycle, well before concrete technical information is available, they are subject to greater uncertainty than they have been in the past. Early lifecycle cost estimates are often based on a desired capability rather than a concrete solution. Faced with investment decisions based primarily on capability, several problems are encountered when creating estimates at this early stage:

- *Limited Input Data* -The required system performance, the maturity of the technology for the solution, and the capability of the vendors are not fully understood.

- *Uncertainties in Analogy-Based Estimates* - Most early estimates are based on analogies to existing products. While many factors may be similar, the execution of the program and the technology used as part of the system or to develop it are often different. For example, software product size depends heavily on the implementation technology, and the technology heavily influences development productivity. Size and productivity are key parameters for cost estimation.

• *Challenges in Expert Judgment* - Wide variation in judgment can exist between experts and the confidence in the input that they provide is generally not quantified and unknown.

• *Unknown Technology Readiness* – Technology readiness may not be well-understood, and is likely to be over or under estimated.

An Improved Method for Early Cost Estimation

In 2011 the SEI introduced the QUELCE (Quantifying Uncertainty in Early Cost Estimation) method, an integrative approach for pre-Milestone A cost estimation. The method aims to provide credible and accurate program cost estimates within clearly defined, statistically valid confidence intervals. QUELCE produces intuitive visual representations of the data that explicitly model influential relationships and interdependencies among the drivers on which the estimates depend. Assumptions and constraints underlying the estimates are well documented, which contributes to better management of cost, schedule, and adjustments to program scope as more is learned and conditions change. Documenting the basis of an estimate facilitates updating the estimate during program execution and helps others make informed judgments about estimation accuracy.

The QUELCE method differs from existing methods because it

• uses available information not normally employed for program cost estimation

• provides an explicit, quantified consideration of the uncertainty of the program change drivers

• enables calculation (and re-calculation) of the cost impacts caused by changes that may occur during the program lifecycle

- enhances decision-making through the transparency of the assumptions going into the cost estimate

Figure B.1 shows the flow of information in a typical MDAP Acquisition, with blue boxes added to represent the contributions from the QUELCE method.

How QUELCE Works

QUELCE synthesizes scenario building, Bayesian Belief Network (BBN) modeling, and Monte Carlo simulation into an estimation method that quantifies uncertainties, allows subjective inputs, visually depicts influential relationships among change drivers and outputs, and assists with the explicit description and documentation underlying an estimate. It uses scenario analysis and design structure matrix (DSM) techniques to limit the combinatorial effects of multiple interacting program change drivers to make modeling and analysis more tractable. Representing scenarios as BBNs enables sensitivity analysis, exploration of alternatives, and quantification of uncertainty.

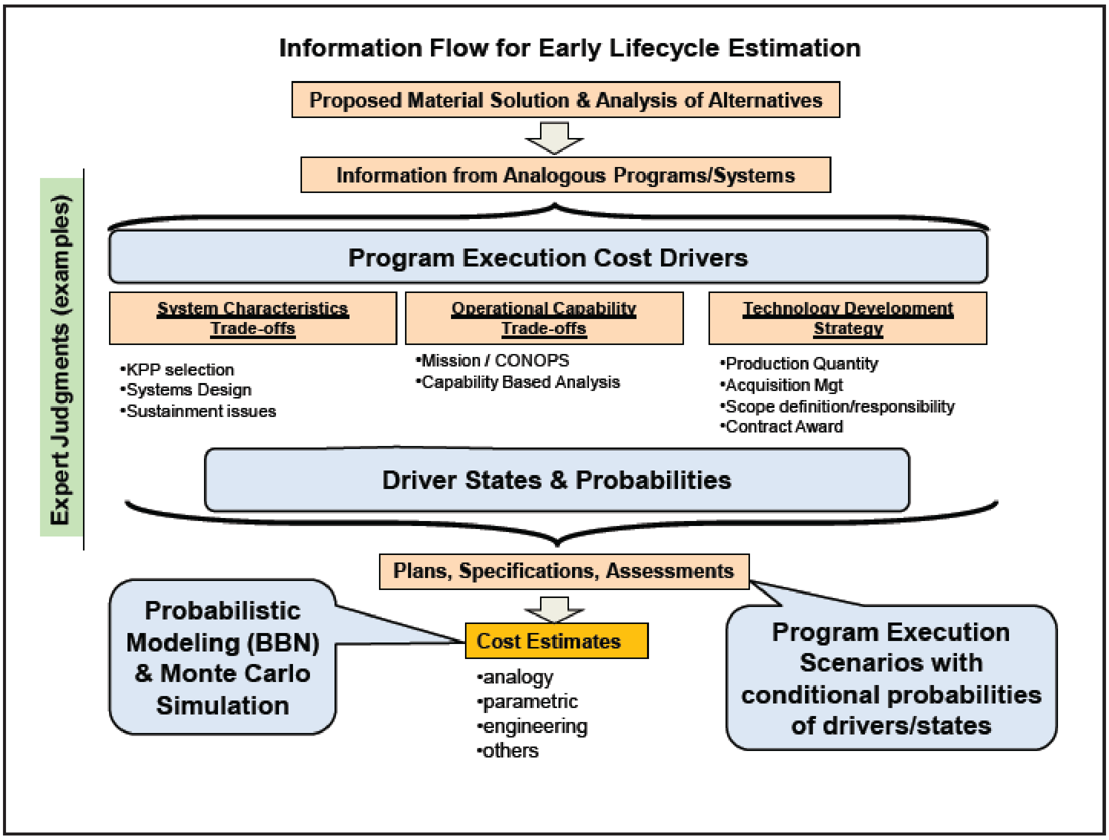

Information Flow for Early Lifecycle Estimation
Proposed Material Solution & Analysis of Alternatives
Information from Analogous Programs/Systems
Expert Judgments (examples)
Program Execution Cost Drivers
System Characteristics Trade-offs
Operational Capability Trade-offs
Technology Development Strategy
•KPP selection
•Systems Design
•Sustainment issues
•Mission / CONOPS
•Capability Based Analysis
•Production Quantity
•Acquisition Mgt
•Scope definition/responsibility
•Contract Award
Driver States & Probabilities
Plans, Specifications, Assessments
Probabilistic Modeling (BBN) & Monte Carlo Simulation
Cost Estimates
•analogy
•parametric
•engineering
•others
Program Execution Scenarios with conditional probabilities of drivers/states

Figure B.1 Information Flow for Early Lifecycle Estimation, Including the QUELCE Method

The BBNs and Monte Carlo simulation are then used to predict variability of what become the inputs to existing, commercially available cost estimation methods and tools. As a result, interim and final cost estimates are embedded within clearly defined confidence intervals. The method can be described as a series of eleven steps, summarized in the following sections. Our recent SEI technical report, CMU/SEI-2011-TR-026, elaborates further on the method and its application.

Step 1: Identify Program Change Drivers

The identification of program change drivers is best accomplished by the experts who provide programs with information for consideration in cost estimation. Workshops with DoD contractors, domain experts, and former DoD program managers are used to identify drivers that could affect program costs. These experts should consider all aspects of a program that might change (and affect cost) during the program's lifecycle—particularly given the new information developed during the Technology Development Phase in preparation of Milestone B. The Probability of Program Success (POPS) factors used by the Navy and Air Force can be used to start the discussion.

Step 2: Identify States of Program Change Drivers

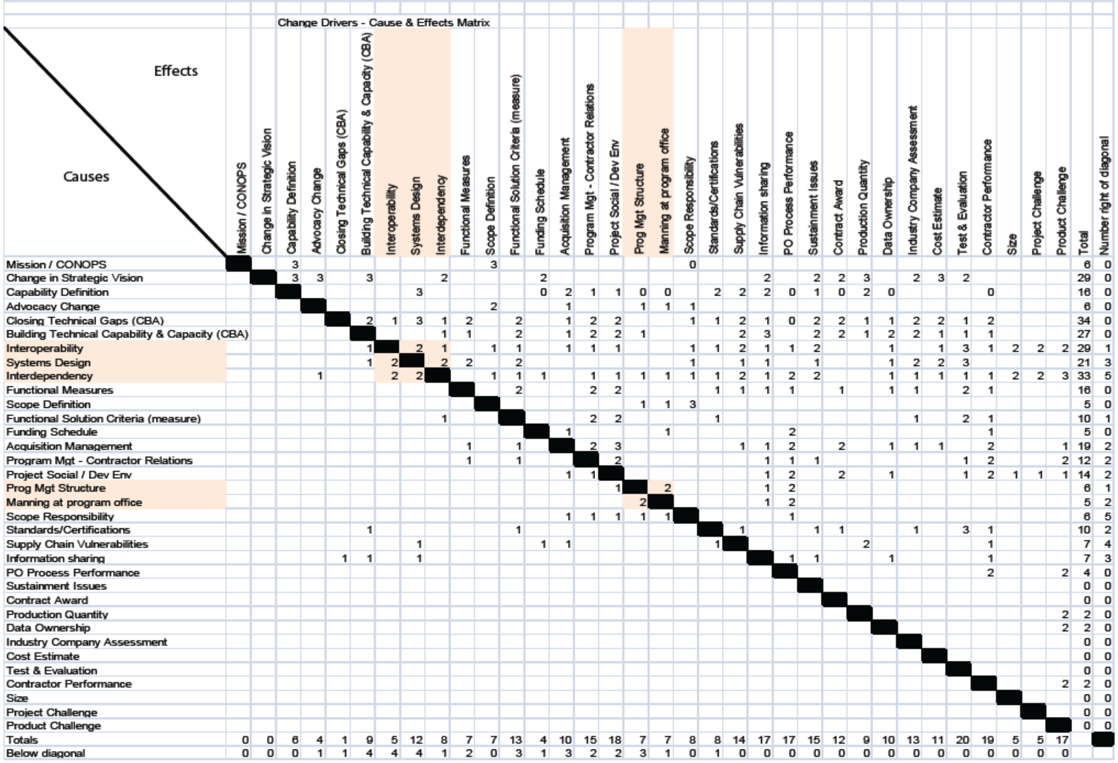
In the workshops, experts are asked to brainstorm ideas about the status of each program change driver. The specific, assumed state as proposed by the Materiel Solution is identified and labeled as the nominal state. Experts then brainstorm about possible changes in the condition of each driver that may occur during the program lifecycle. The experts identify possible changes that might occur to the nominal state and use their best judgment for the probability that the nominal state will change.

Step 3: Identify Cause-and-Effect Relationships for Dependency Matrix

Once the changed condition— referred to as potential driver states—are fully identified, participants subjectively evaluate the cause and effect relationships among the drivers. Expert judgment is applied to rank the causal effects. A matrix is developed that provides the relationship between nominal and dependent states and contains the conditional probability that one will affect the other, but not the impact of the change. This exercise can result in a very large number of program change drivers and states identified for an MDAP.

Step 4: Reduce the Dependency Matrix Using a Design Structure Matrix

Using the Design Structure Matrix (DSM) technique the change drivers can be reduced to an efficient set that has the most potential impact to cost. The DSM technique is a well-established method to reduce complicated dependency structures to a manageable size. An example of a dependency matrix after DSM transformation created during an SEI pilot workshop is provided in Figure B.2.



Change Drivers - Cause & Effects Matrix

Effects Causes	Mission / CONOPS	Change in Strategic Vision	Capability Definition	Advocacy Change	Closing Technical Gaps (CBA)	Building Technical Capability & Capacity (CBA)	Interoperability	Systems Design	Interdependency	Functional Measures	Scope Definition	Functional Solution Criteria (measure)	Funding Schedule	Acquisition Management	Program Mgt - Contractor Relations	Project Social / Dev Env	Prog Mgt Structure	Manning at program office	Scope Responsibility	Standards/Certifications	Supply Chain Vulnerabilities	Information sharing	PO Process Performance	Sustainment Issues	Contract Award	Production Quantity	Data Ownership	Industry Company Assessment	Cost Estimate	Test & Evaluation	Contractor Performance	Size	Project Challenge	Product Challenge	Total	Number right of diagonal
Mission / CONOPS	■		3								3								0																6	0
Change in Strategic Vision		■	3	3		3			2				2									2		2	2	3		2	3	2					29	0
Capability Definition			■					3					0	2	1	1	0	0		2	2	2	0	1	0	2	0				0				16	0
Advocacy Change				■							2			1			1	1	1																6	0
Closing Technical Gaps (CBA)					■	2	1	3	1	2		2		1	2	2			1	1	2	1	0	2	2	1	1	2	2	1	2				34	0
Building Technical Capability & Capacity (CBA)						■			1	1		2		1	2	2	1				2	3		2	2	1	2	2	1	1	1				27	0
Interoperability						1	■	2	1		1	1		1	1	1			1	1	2	1	1	2			1		1	3	1	2	2	2	29	1
Systems Design						1	2	■	2	2		2							1		1	1		1			1	2	2	3					21	3
Interdependency				1			2	2	■		1	1	1		1	1	1	1	1	1	2	1	2	2			1	1	1	1	1	2	2	3	33	5
Functional Measures										■		2			2	2				1	1	1	1		1		1	1		2	1				16	0
Scope Definition											■						1	1	3																5	0
Functional Solution Criteria (measure)									1			■			2	2				1								1		2	1				10	1
Funding Schedule													■	1				1					2								1				5	0
Acquisition Management										1		1		■	2	3					1	1	2		2		1	1	1		2			1	19	2
Program Mgt - Contractor Relations										1		1			■	2						1	1	1						1	2			2	12	2
Project Social / Dev Env														1	1	■						1	2		2		1			1	2	1	1	1	14	2
Prog Mgt Structure																1	■	2				1	2												6	1
Manning at program office																	2	■				1	2												5	2
Scope Responsibility														1	1	1	1	1	■				1												6	5
Standards/Certifications						1						1								■	1			1	1			1		3	1				10	2
Supply Chain Vulnerabilities								1					1	1						1	■					2					1				7	4
Information sharing					1	1		1														■	1	1			1				1				7	3
PO Process Performance																							■								2			2	4	0
Sustainment Issues																								■											0	0
Contract Award																									■										0	0
Production Quantity																										■								2	2	0
Data Ownership																											■							2	2	0
Industry Company Assessment																												■							0	0
Cost Estimate																													■						0	0
Test & Evaluation																														■					0	0
Contractor Performance																															■			2	2	0
Size																																■			0	0
Project Challenge																																	■		0	0
Product Challenge																																		■	0	0
Totals	0	0	6	4	1	9	5	12	8	7	7	13	4	10	15	18	7	7	8	8	14	17	17	15	12	9	10	13	11	20	19	5	5	17		■
Below diagonal	0	0	0	1	1	4	4	4	1	2	0	3	1	3	2	2	3	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0		

Figure B.2 Example Dependency Matrix After DSM Transformation

Step 5: Construct a BBN Using the Reduced Matrix

Using the program change drivers derived from Step four and their cause and effect relationships established in Step 3, a BBN is constructed. This BBN is a probabilistic model that dynamically represents the drivers and their relationships, as envisioned by the program domain experts. Figure B.3 depicts an abbreviated visualization of a BBN, in which the circled nodes represent program change drivers and the arrows represent either cause and effect relationships or leading indicator relationships. This example shows that a change in the Mission and CONOPS driver most likely will cause a change to the Capability Analysis driver, which in turn will likely effect a change in the Key Performance Parameter (KPP) driver and subsequently the Technical Challenge outcome factor. The three outcome factors (Product Challenge, Project Challenge, and Size Growth) are then used to predict some of the input values for traditional cost estimation models.

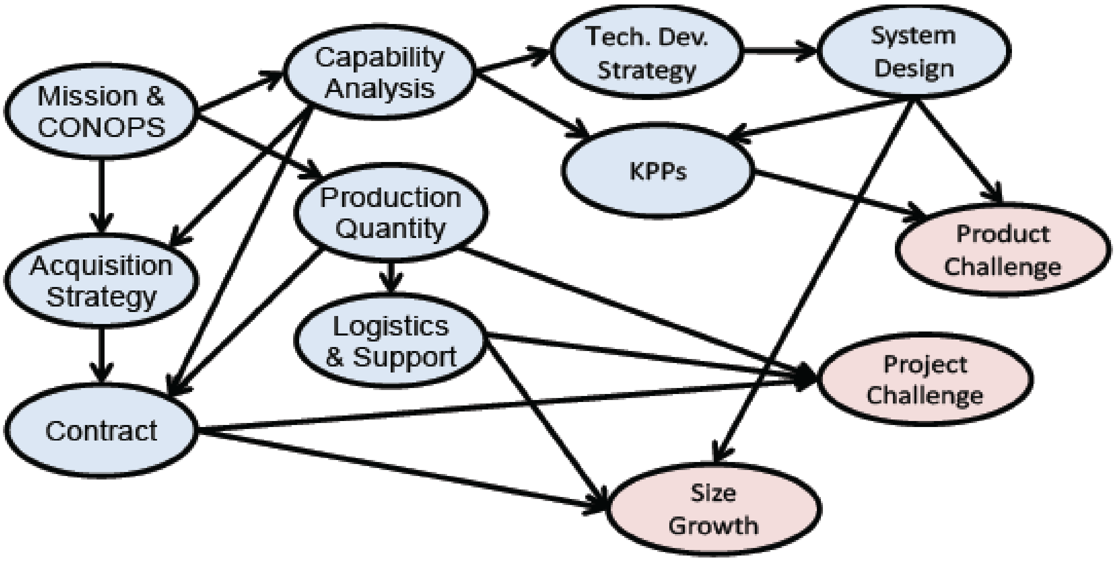


Figure B.3 Example BBN

Step 6: Populate BBN Nodes with Conditional Probabilities

Conditional probabilities are assigned to the nodes (drivers) in the BBN. Each node can assume a variety of states, each of which has an associated likelihood identified by the domain experts. This allows the calculation of outcome distributions on the variables.

Step 7: Define Scenarios by Altering Program Change Driver Probabilities

Domain experts use the BBN to define scenarios. The realization of a potential state in a particular node was specified in Step 6, and the cascading impacts to other nodes and the resulting change in the outcome variables were recalculated. Any change in one or more nodes (drivers) constitutes a scenario. Once the experts are satisfied that a sufficient number of scenarios are specified, they use their judgment to rank them for likely impacts to cost. An example scenario created during an SEI pilot workshop is provided in Figure B.4.

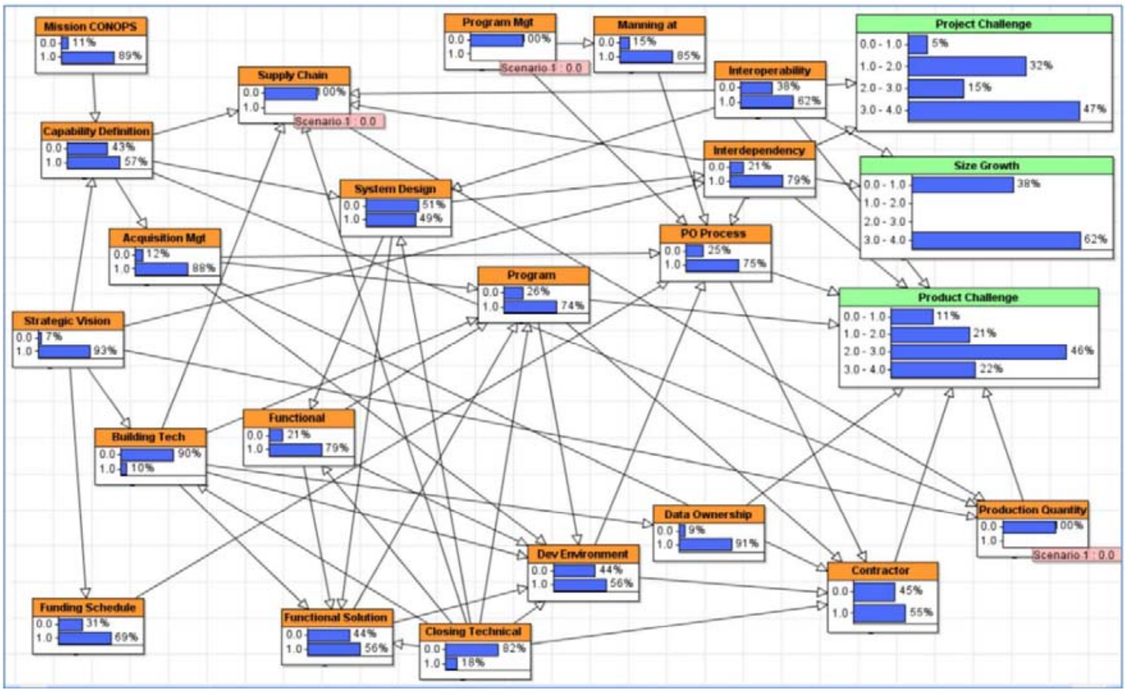
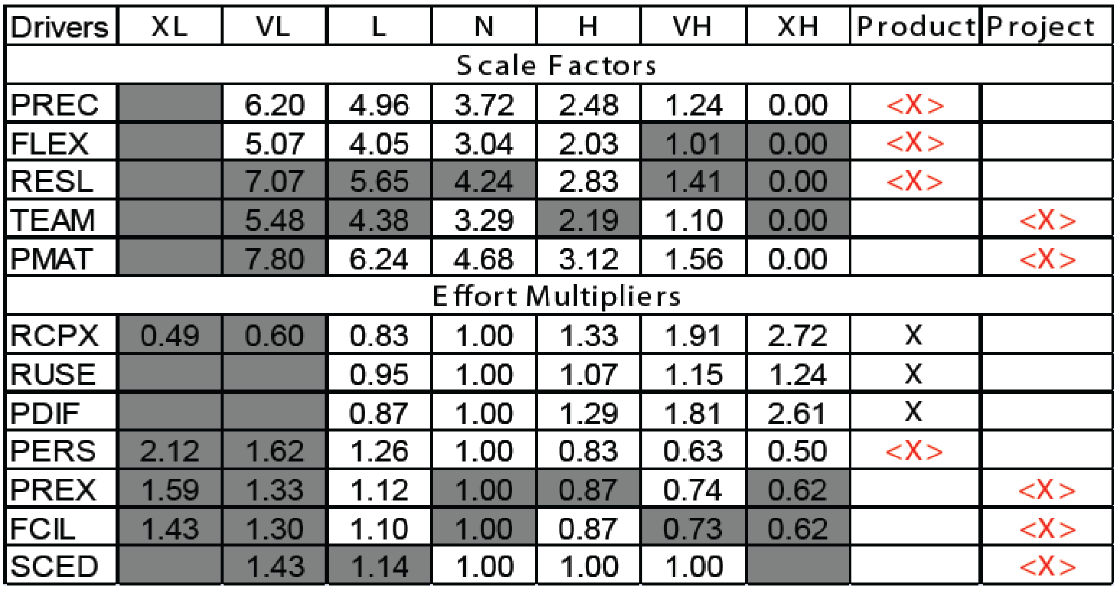


Figure B.4 Example of a Scenario With Two Driver Nodes In A Nominal State
Data & Analysis Center for Software (DACS)

Step 8: Select Cost Estimating Relationships or Tools to Generate an Estimate

Parametric cost estimation models for software use a mathematical equation to calculate effort and schedule from estimates of size and a number of parameters. A decision is made as to which cost estimating tool or tools, CERs, or other methods will be used to form the cost estimate. COCOMO II is a well-known estimation tool and is open source. The SEI has so far developed the relationships between BBN-modeled program change drivers and COCOMO, shown in Figure B.5. The use of the commercial SEER cost estimating tool is being explored.



Drivers	XL	VL	L	N	H	VH	XH	Product	Project
Scale Factors									
PREC		6.20	4.96	3.72	2.48	1.24	0.00	<X>	
FLEX		5.07	4.05	3.04	2.03	1.01	0.00	<X>	
RESL		7.07	5.65	4.24	2.83	1.41	0.00	<X>	
TEAM		5.48	4.38	3.29	2.19	1.10	0.00		<X>
PMAT		7.80	6.24	4.68	3.12	1.56	0.00		<X>
Effort Multipliers									
RCPX	0.49	0.60	0.83	1.00	1.33	1.91	2.72	X	
RUSE			0.95	1.00	1.07	1.15	1.24	X	
PDIF			0.87	1.00	1.29	1.81	2.61	X	
PERS	2.12	1.62	1.26	1.00	0.83	0.63	0.50	<X>	
PREX	1.59	1.33	1.12	1.00	0.87	0.74	0.62		<X>
FCIL	1.43	1.30	1.10	1.00	0.87	0.73	0.62		<X>
SCED		1.43	1.14	1.00	1.00	1.00			<X>

Figure B.5 Mapping BBN Outputs to COCOMO Inputs

Step 9: Obtain Program Estimates Not Computed by the BBN

The Program Office estimates of size and/or other cost model inputs such as productivity are used as the starting point in this step. Often these values are estimated by analogy and aggregation. They are adjusted by applying the distributions calculated by the BBN.

Step 10: For Each Scenario, Run a Monte Carlo Simulation

From each selected scenario in Step 7, use the outcome to parameterize a Monte Carlo simulation. Along with the information from Step 9, this provides probability distributions for adjusting the input factors to the cost estimating models. This also provides explicit confidence levels for the results. Figure B.6 shows the simulation results the SEI obtained when modeling a factor (person-months) in three different scenarios.

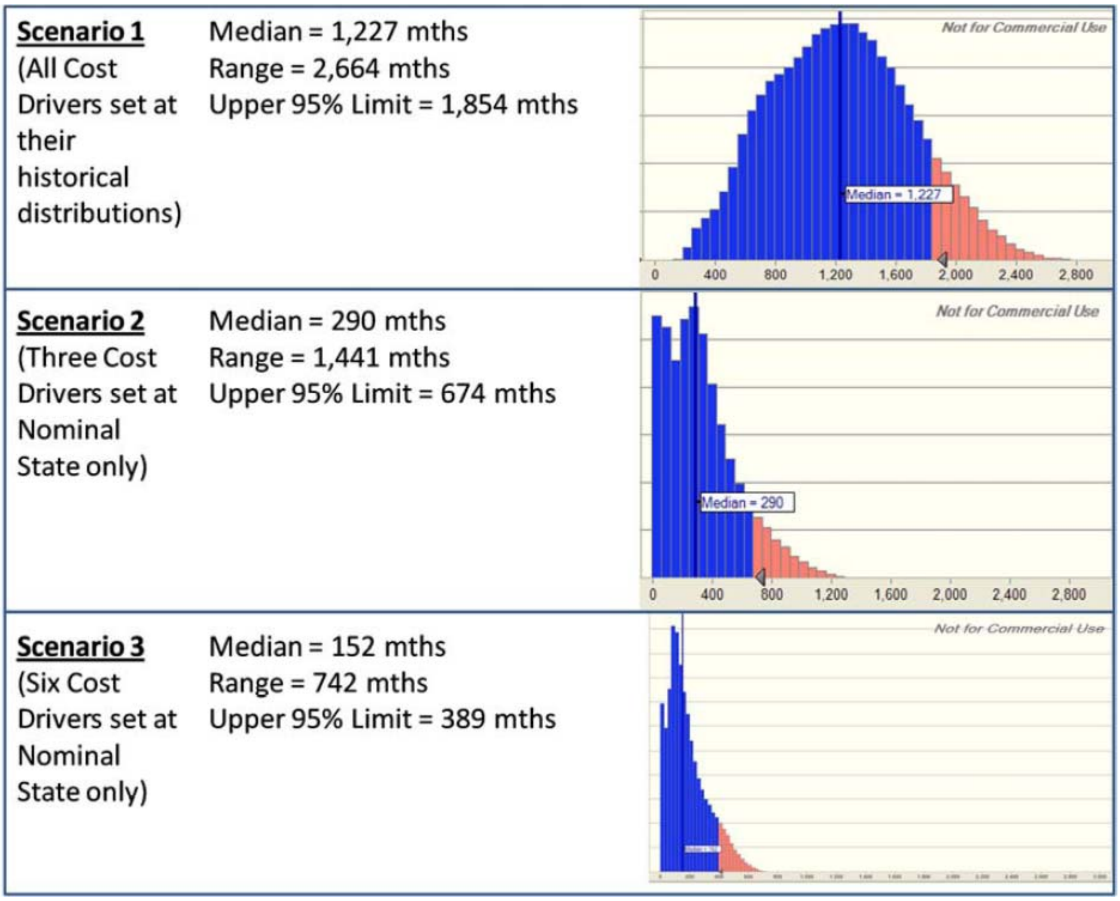
Step 11: Report Each Scenario Result Independently

Report the final cost estimates for each scenario, including the nominal program plan. The explicit confidence levels and the visibility of all considered program change drivers allows for quick comparisons and future re-calculations. The transparency afforded by the consideration of alternative scenarios enables improved decision making and contingency planning.

Results and Future Research

QUELCE as an approach to early cost estimation is unprecedented in many ways. The SEI spent much of the past year developing and refining the analytical methods used. So far, trials of the earlier steps of the method have been conducted in workshops, and post hoc reviews of previous estimation artifacts were used for later steps. The SEI's experience and the results achieved thus far suggest that the approach has considerable merit. Feedback about the value of the approach from the participants in workshops and from leaders in estimation research has been very positive.

Empirical validation of QUELCE is ongoing, and the results of this evaluative research will be used to refine the approach and demonstrate its value. Future efforts will benefit from the participation of programs that are willing to provide access to the artifacts developed prior to Milestone A or to use the QUELCE method in upcoming Milestone A estimates. Through these joint efforts, the SEI will evaluate the extent to which the probabilistic methods proposed improve the accuracy and precision of cost estimates for DoD programs.



Scenario 1 (All Cost Drivers set at their historical distributions)	Median = 1,227 mths
	Range = 2,664 mths
	Upper 95% Limit = 1,854 mths

Scenario 2 (Three Cost Drivers set at Nominal State only)	Median = 290 mths
	Range = 1,441 mths
	Upper 95% Limit = 674 mths

Scenario 3 (Six Cost Drivers set at Nominal State only)	Median = 152 mths
	Range = 742 mths
	Upper 95% Limit = 389 mths

Figure B.6 Simulation Results for Three Scenarios

Conclusion

Extensive cost overruns have been endemic in defense programs for many years. A significant part of the problem is that cost estimates for unprecedented systems must rely heavily on expert judgments made under uncertain conditions. QUELCE aims to reduce the adverse effects of that uncertainty. Important program change drivers and the dependencies among them that may not otherwise be considered are made explicit to improve the realism and likely accuracy of the estimates. The basis of an estimate is documented explicitly, which facilitates updating the estimate during program execution and helps others to make informed judgments about their accuracy. Variations in the range of possible states of the program change drivers that may occur under different likely scenarios are explicitly considered. The use of probabilistic methods combining Bayesian belief systems and Monte Carlo simulation will ultimately place the cost estimates within a more defensible range of uncertainty.

REFERENCES

[1] Neil Fox, The Two Metrics that Matter, http://agile.techwell.com accessed on Aug 15, 2011

[2] IT Cotex , Failure Rate, http://www.it-cortex.com/Stat_Failure_Rate.htm accessed on July 30, 2011

[3] Robert N. Charette, Why Software Fails, http://spectrum.ieee.org/computing/software/why-software-fails, accessed on Aug 08, 2011

[4] James McDonald, Managing the Development of Software-Intensive Systems, pp.3 WILEY, Hoboken, NJ, 2009

[5] James McDonald, Managing the Development of Software-Intensive Systems, pp.141,WILEY, Hoboken, NJ, 2009

[6] Barry W. Boehm, Software Engineering Economics, pp.282-283, Prentice-Hall, INC., Englewood Cliffs, NJ, 1981

[7] Barry W. Boehm, Software Engineering Economics, pp.75, Prentice-Hall, INC., Englewood Cliffs, NJ, 1981

[8] Barry W. Boehm, Software Cost Estimation with COCOMO II, pp.13, Prentice-Hall, INC., Upper Saddle River, NJ, 2000

[9] Barry W. Boehm, Software Cost Estimation with COCOMO II, pp.15, Prentice-Hall, INC., Upper Saddle River, NJ, 2000

[10] Barry W. Boehm, Software Cost Estimation with COCOMO II, pp.19, Prentice-Hall, INC., Upper Saddle River, NJ, 2000

[11] http://www.qsm.com/resources/function-point-languages-table accessed on July 25, 2011

[12] http://www.qsm.com/resources/function-point-languages-table accessed on July 25, 2011

[13] DCG, Software Sizing with Function Points, http://www.davidconsultinggroup.com accessed on Oct 02, 2011

[14] Hao Wang, Software Productivity Analysis with CSBSG Data Set, CSSE, vol. 2, pp.587-593, 2008 International Conference on Computer Science and Software Engineering, 2008

[15] Dan Galorath, Software Staff Size Still Impacts Productivity: Brooks Law Lives!, http://www.galorath.com/wp/software-staff-size-still-impacts-productivity-brooks-law-lives.php accessed on Sep 11, 2011

[16] Barry W. Boehm, Software Engineering Economics, pp.372, Prentice-Hall, INC., Englewood Cliffs, NJ, 1981

[17] Larry Bernstein, 8-Predicting and Improving the Reliability of Software Intensive Systems, http://paris.utdallas.edu/IEEE-RS-ATR/document/2010/8-Predicting%20and%20Improving%20the%20Reliability%20of%20Software%20Intensive%20Systems_Bernstein.pdf accessed on July 19, 2011

[18] Barry W. Boehm, Software Engineering Economics, pp.374, Prentice-Hall, INC., Englewood Cliffs, NJ, 1981

[19] Barry W. Boehm, Safe and Simple Software Cost Analysis, http://csse.usc.edu/csse/TECHRPTS/2000/usccse2000-529/usccse2000-529.pdf, accessed on Jan 07, 2012

[20] http://sunset.usc.edu/research/COCOMOII/Docs/modelman.pdf, accessed on Aug 26, 2011

[21] Barry W. Boehm, COCOMO/SCM Forum #13, page 6, USC, http://csse.usc.edu/csse/event/1998/COCOMO/3_Boehm%20COCOMO.pdf, accessed on Feb 4, 2012

[22] Barry W. Boehm, TRW, Improving Software Productivity, http://csse.usc.edu/csse/TECHRPTS/1987/usccse87-502/usccse87-502.pdf accessed on Nov 7, 2012

[23] http://www.mathwave.com/products/easyfit.html accessed on Dec 11, 2011

[24] Barry W. Boehm, Software Engineering Economics, pp.379, Prentice-Hall, INC., Englewood Cliffs, NJ, 1981

[25] Barry W. Boehm, Software Engineering Economics, pp.463, Prentice-Hall, INC., Englewood Cliffs, NJ, 1981

[26] Barry W. Boehm, Software Engineering Economics, pp.469, Prentice-Hall, INC., Englewood Cliffs, NJ, 1981

[27] Bradford K. Clark, The Effects of Software Process Maturity on Software Development Effort, A Dissertation Presented to the FACULTY OF THE GRADUATE SCHOOLUNIVERSITY OF SOUTHERNCALIFORNIA http://sunset.usc.edu/~bkclark/Research/Dissertation.pdf accessed on Feb 5, 2012

[28] Eric Torkia, Excel Simulation Show-Down: Comparing the top Monte-Carlo Simulation Tools, http://www.crystalballservices.com/Resources/ConsultantsCornerBlog/EntryId/71/Excel-Simulation-Show-Down-Comparing-the-top-Monte-Carlo-Simulation-Tools.aspx, accessed on Sep 20, 2011

[29] Barry W. Boehm, Terence E. Gray, and Thomas Seewaldt, PROTOTYPING VS. SPECIFYING: A MULTI-PROJECT EXPERIMENT, University of California, Los Angeles Computer Science Department, 1984 http://csse.usc.edu/csse/TECHRPTS/1982/usccse82-500/usccse82-500.pdf accessed on Jan 24, 2012

[30] Dirk Basten, Werner Mellis, A Current Assessment of Software Development Effort Estimation, ESEM, pp.235-244, 2011 International Symposium on Empirical Software Engineering and Measurement, 2011

[31] Barry W. Boehm, Software Cost Estimation with COCOMO II, pp.15, Figure 3.5, Prentice-Hall, INC., Upper Saddle River, NJ, 2000

[32] Software Engineering Measurement and Analysis (SEMA) Cost Estimation Research Group: Robert Ferguson, Dennis Goldenson, James McCurley, Robert Stoddard, David Zubrow, Debra Anderson, Quantifying Uncertainty in Early Lifecycle Cost Estimation (QUELCE), (CMU/SEI-2011-TR-026), Software Engineering Institute, Carnegie Mellon University, 2011. http://www.sei.cmu.edu/reports/11tr026.pdf accessed on March 15, 2012

[33] SEI Cost Estimation Research Group: Robert Ferguson, Dennis Goldenson, James McCurley, Robert Stoddard, and David Zubrow, An Innovative Approach to Quantifying Uncertainty in Early Lifecycle Cost Estimation, Journal of Software Technology, March 2012 Vol. 15, Number 1, http://journal.thedacs.com/issue/64/207 accessed on April 1, 2012