National Library of Canada
Acquisitions and Bibliographic Services Branch
395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale du Canada
Direction des acquisitions et des services bibliographiques
395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

PERSONAL "PROGRESS FUNCTIONS" IN THE SOFTWARE PROCESS

by
Khalid Sherdil

School of Computer Science
McGill University, Montreal, CANADA

November 1994

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE

ISBN 0-612-05628-7

Canada

Abstract

Individuals can expect continuous improvement in productivity as a consequence of (i) a growing stock of knowledge and experience gained by repeatedly doing the same task (first-order learning) or (ii) due to technological and training programs injected by the organization (second-order learning). Organizations have used this type of progress behavior in making managerial decisions regarding cost estimating and budgeting, production and labor scheduling, product pricing, etc. This progress was studied in productivity, product-quality and personal skills, in an experiment involving a sample of 12 subjects, who completed one project every week for ten weeks. Second-order training was provided to the subjects through the Personal Software Process, PSP, of Humphrey. A within-subject repeated measure time-series quasi-experimental design was used along with a modified G/Q/M method. It was found that on average, progress takes place at a rate of 20%, with the second-order training adding up to 13% more improvement in addition to the first-order learning. Detailed statistical methods were used to produce linear and log-linear models of high correlations, involving four variables: productivity, defect-rate, complexity and cumulative output. The motivation of the subjects did not change significantly during the experiment. It was also found that the McCabe's and Halstead's complexity metrics had a correlation of 0.80 amongst each other. However, no relationship could be found between the personal capabilities of the individuals and the progress rate.

Résumé

Les gens peuvent s'attendre à une amélioration continue de leur productivité due à (i) l'accumulation de connaissances et d'expérience acquise en répétant les mêmes tâches (apprentissage de premier ordre) ou (ii) due à des programmes technologiques et d'entraînement introduits par les entreprises (apprentissage de second ordre). Les organisations ont déjà utilisé ce type de comportement du progrès dans leurs décisions concernant l'estimation des coûts et du budget, la planification de la production et du travail, la tarification des produits, etc. Le progrès en productivité, qualité de produits, et habileté personnelle a été étudié dans une expérience impliquant un échantillon de 12 personnes qui ont complété un projet par semaine pendant dix semaines. Un entraînement de second ordre leur a été donné à travers le "Personal Software Process", PSP, de Humphrey. Un plan quasi-expérimental avec mesures répétées à l'intérieur du groupe de sujets a été utilisé avec une méthode G/Q/M modifiée. Nous avons découvert que en moyenne, le progrès du à l'apprentissage de premier ordre était de 20%, et que l'entraînement de second ordre améliorait ce pourcentage de 13%. Des méthodes statistiques détaillées ont été utilisées pour produire des modèles linéaires et log-linéaires de haute corrélation utilisant quatre variables: productivité, taux d'erreur, complexité, et temps. La motivation des sujets n'a pas changé de façon significative durant l'expérience. Nous avons aussi découvert que les mesures de complexité de McCabe et Halstead avait une corrélation de 0.80 entre eux. Par contre, aucune relation n'a pu être faite entre les capacités personnelles des individus et leur taux de progrès.

Acknowledgments

I will like to thank Professor Nazim H. Madhavji, my supervisor, for his support and supervision in the process of carrying out this research.

I am also thankful to Khaled El Emam, for his help through out my Masters program. His comments and suggestions have helped me a lot in designing and executing this experimental work.

I am thankful to my parents and family for their moral support. I would also like to thank my research group in particular, and the McGill University in general, for providing an excellent environment to study.

List of Figures

Figure 34: A typical graph of the subjective metrics *109*

List of Tables

List of Boxes

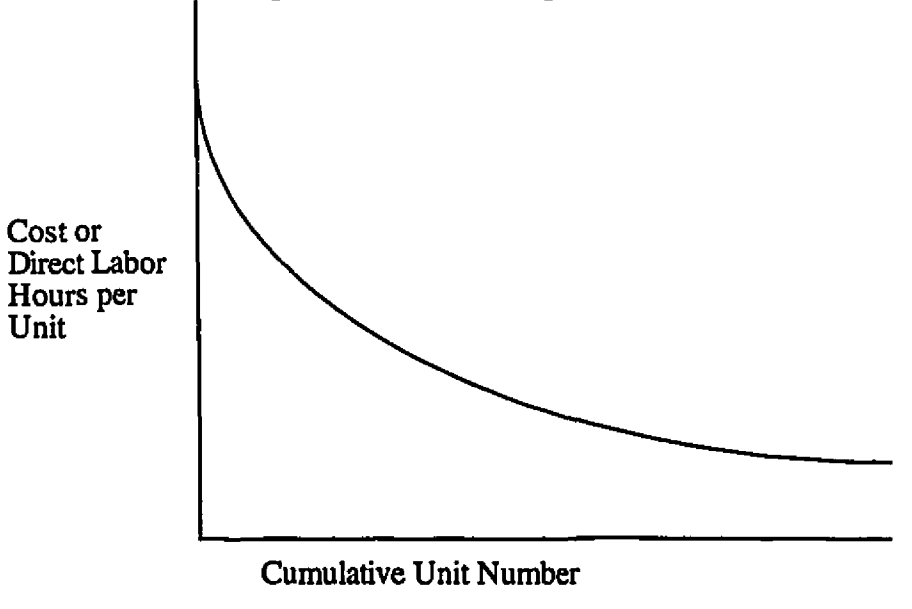
Contents

1.0 Introduction

An individual person can expect continuous improvement in productivity as a consequence of a growing stock of knowledge and experience gained by repeatedly doing the same task [5]. Organizations have used this type of progress behavior in making managerial decisions regarding cost estimating and budgeting, production and labor scheduling, product pricing, etc. [4][29]. While considerable research on this topic has been done in industrial and manufacturing sectors [60], we found little such emphasis in the software process field, which is one of the most emerging and widely used `Industry' of the present times. Furthermore, such research has remained almost exclusively focused on outcomes, such as units of items produced, rather than processes [1], although it is the industrial processes which determine the nature of the product.

Our objective in this research was to study the *progress functions* for an individual in the domain of software process. A progress function is a mathematical form for representing the improvement in performance, typically for some production activity. In simple terms, the progress function represents the percentage decline in cost or labor requirement as the cumulative output increases by one unit [29][35], as shown in Figure 1.

Figure 1: The Learning Curve



Progress functions differ from the widely used term *learning curves* because the former also incorporate a *second-order* learning mechanism [24]. Whereas the *first-order* learning (also referred as *autonomous* or *labor* learning) is the improvement due to the experience which a person gains by repeatedly doing the same task, the second-order learning (also known as *induced* or *organizational* learning) is due to technological and training programs injected by the organization [2][42]. Although the distinction between these two is often blurred [42], we have analyzed them separately in our work. Such a distinction is useful for making managerial decisions regarding initiating formal training programs and making engineering technology changes [2].

In order to study these progress functions, we have carried out an experimental study. An integral part of this experiment is the personal software process, PSP, designed by Humphrey [30]. In this experiment, we deployed 12 software developers working on 10 short programming projects at a rate of one project per week. These projects used the C++ programming language, which none of the subjects was familiar with, and had to learn it during the course of the experiment. For each project, they kept precise track of even the minutest measurements, such as the details of every defect found, every line of code (new, reused, modified), time spent on each development phase, etc. For consistency, all software developers used the same standards throughout the experiment, such as the same coding standards, the same personal process, the same data collection techniques, etc. The subjects underwent an intensive training program through which they were taught various software engineering techniques, which provided them a second-order organizational learning medium. This training program introduced one new concept every week, such as the statistical methods for estimating size, code reviews, design reviews, etc.

Using the data gathered, besides measuring their rate of progress, we attempted to analyze the contributions of the first and second order learnings. This involved, amongst other things, an evaluation of the subjects size-estimation abilities, the advantages of reusing code, and the benefits of using code-reviews. Our study also incorporates various subjective measures, such as the managerial motivation provided to the subjects, the degree of the subjects' personal capabilities, their programming experience, etc. This includes an analysis of how the first-order learning is related to the personal capabilities of the subjects.

Finally, one of the most important aspects of our work emerged from the results of the study of the programmer-productivity and defect-quality learning curves. We found that besides being dependent upon time or cumulative output, the two above mentioned variables also depend upon the complexity of the programming

task as well as upon each other. This lead to an analysis of a model involving the following four variables, using detailed statistical techniques:

- Programmer Productivity
- Defect Quality
- Cumulative Output (or Time)
- Complexity

For measuring complexity, we used the two most popular metrics, namely McCabe's and Halstead's. We also carried out a comparative analysis of these two metrics.

In order to carry out scientifically valid work, we followed Basili's Goal Question Metric (G/Q/M) paradigm [9] for defining our objectives. This measurement paradigm describes how to define the objectives and formalize metrics corresponding to them. However, we made some modifications to this method in order to incorporate some extra precautions against threats to the validity of our experiment.

Any acceptable study needs a scientifically valid experimental design. Our design is a hybrid of the time-series quasi-experimental design and the repeated-measure within-subject design. The details of such experiment designs are described in the body of the thesis.

Our work has several original aspects to it which have not been dealt with before, particularly in software engineering. As mentioned earlier, while considerable research on this topic has been carried out in industrial and manufacturing sectors, little emphasis has been paid in the software process field. Due to the very nature of the software process field, we had to take into consideration several variables which are not needed in the manufacturing field. For example, nearly all the learning curve studies have been carried out with the assumption that the nature and complexity of the task stays constant. However, in software development, such an assumption would be unreasonable since the programming projects generally vary in the level of difficulty. Therefore, we had to treat the Complexity of the work done as a variable in addition to the cumulative output. Furthermore, our work is different from most of the past studies because besides measuring the progress in productivity, it involves measuring the progress in product quality, something not considered in other studies. Similarly, most of the studies on progress functions use a count of tangible objects to measure the rate of progress. Our work not only looks at this productivity-oriented aspect of progress but also at

the progress in the personal skills of the individual.

Our study results are as follows:

- Learning took place in all the three fields, productivity, quality and skills. The average learning rate was about 20%.
- The second order training and technology helps by up to 13% more in progress, as compared to the first order learning.
- Both, the linear and the log-linear models can be used to describe the learning curve. However, the quadratic model does not significantly increase our understanding of the progress as compared to the linear model.
- No relationship can be deduced from our data between the personal capabilities of the subjects and their progress.
- The motivation of the subjects did not change significantly through the course of the experiment
- The Halstead's and the McCabe's complexity metrics have a high correlation amongst each other
- In the 4-variable model, it was found that as the cumulative output increases, the productivity increases while the defect-rate decreases. The latter two variables were found to be inversely proportional to each other.
- It was found that as complexity increases, the productivity decreases. However, no relationship could be deduced from our data between the complexity and the defect-rate

Section 2 discusses related work on the subject of learning curves. Section 3 gives the details of PSP. Section 4 states our objectives and describes our experiment model. A description of the research method including experiment design is given in section 5 while in-depth data analysis is done in section 6. Finally, comparison with related work and the conclusion are in sections 7 and 8, respectively.

2.0 Related Work

This section presents some background work which is related to our study. It describes how the learning curves evolved and how they are being used. This section also gives an introductory mathematical description of the theory behind the progress functions.

2.1 History of Progress Functions

The history of the learning curves can be traced back to the early twentieth century. In the late nineteenth century, industrial expansion in the USA was accompanied by growing efforts to control management processes using empirical methods [24]. At that time, there were two prevailing theories: economic and managerial. The economic theory paid more attention to the equipment and other capital goods for achieving a greater firm productivity. The managerial theory also focused on static cost functions that were insensitive to time and experience. The progress function was thus a major discovery because it suggested that the efficiency of industrial process was dynamic and changing. [24]

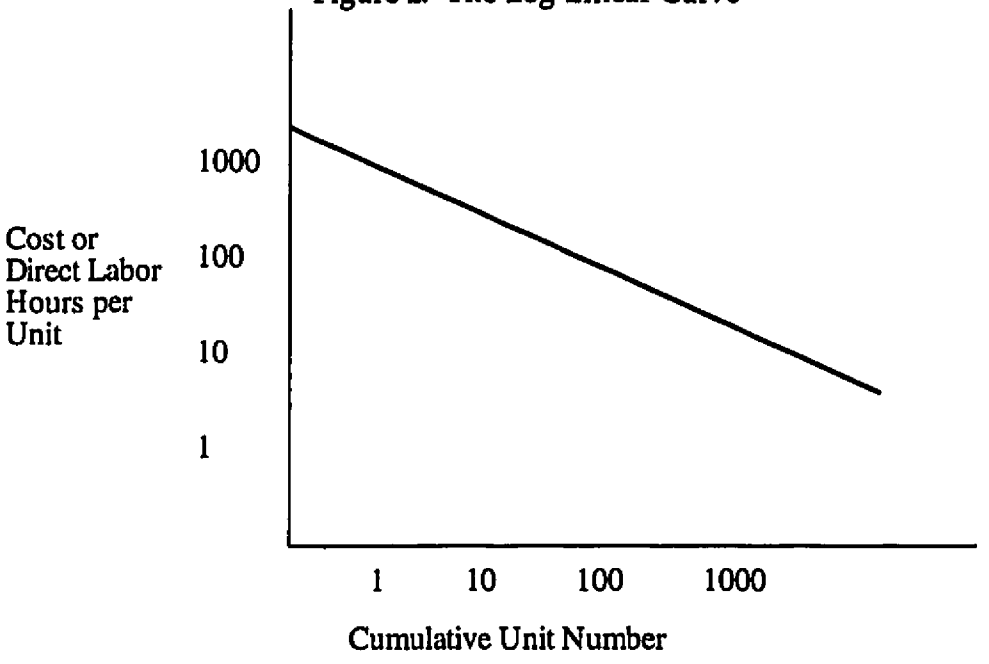
Between 1900 and 1930, the use of progress functions was applied to the domain of aircraft manufacturing while during the second world war, it was applied to the domain of ship-building [46][24]. Contractors searched for ways to predict cost and time requirements for construction of ships and aircraft to conduct the war [60]. Hence, the importance of progress functions grew and by now they have been applied to nearly all the economic and managerial fields, and there exists over 60-years of literature on this topic. These fields include electronics, machine tools, paper-making, steel, apparel, automobiles and others [24]. However, little emphasis has been paid to this topic in the software process field.

T. P. Wright, an engineer and an administrator, was the first to report the phenomenon of the learning curves in 1936 [24][60]. He observed that as the quantity of units manufactured doubles, the direct labor time taken to produce each unit decreases at a uniform rate depending on the manufacturing process being observed. Subsequent studies often substituted direct cost or cumulative output for direct labor time [29].

2.2 Mathematical Models of Progress Functions

Wright used a log-linear model, i.e., when the direct cost (or labor), Y, is plotted against the cumulative output, X, on logarithmic scales, it gives a straight line, as shown in figure 2.

Figure 2: The Log-Linear Curve



However, this model is not suitable for all the situations, and several firms and researchers use different models, such as the plateau model (see Figure 3) or the Stanford-B model (see Figure 4) [60].

Figure 3: The Plateau Model

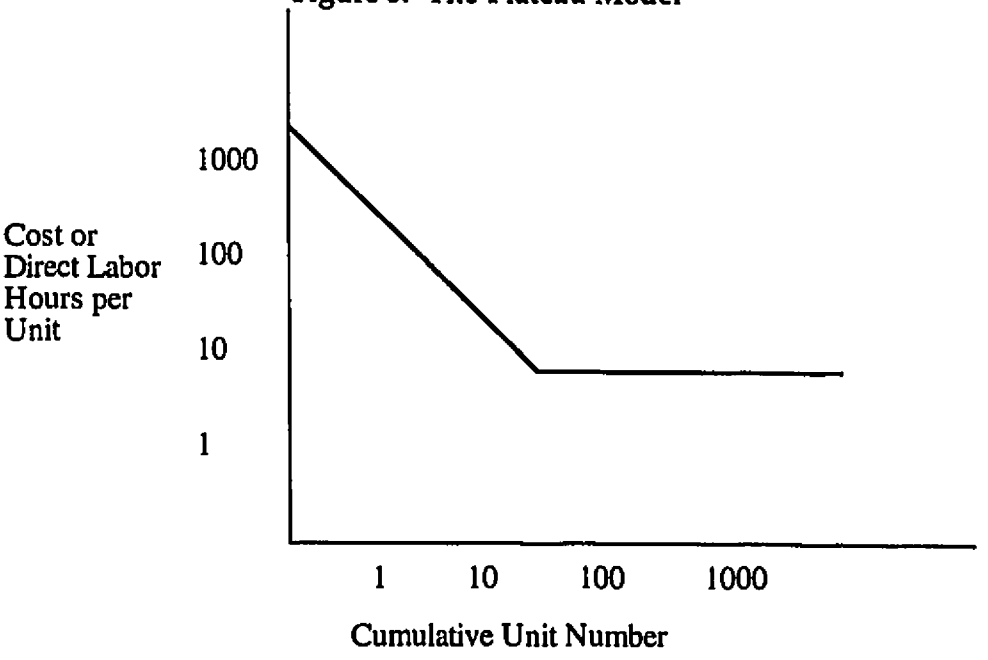
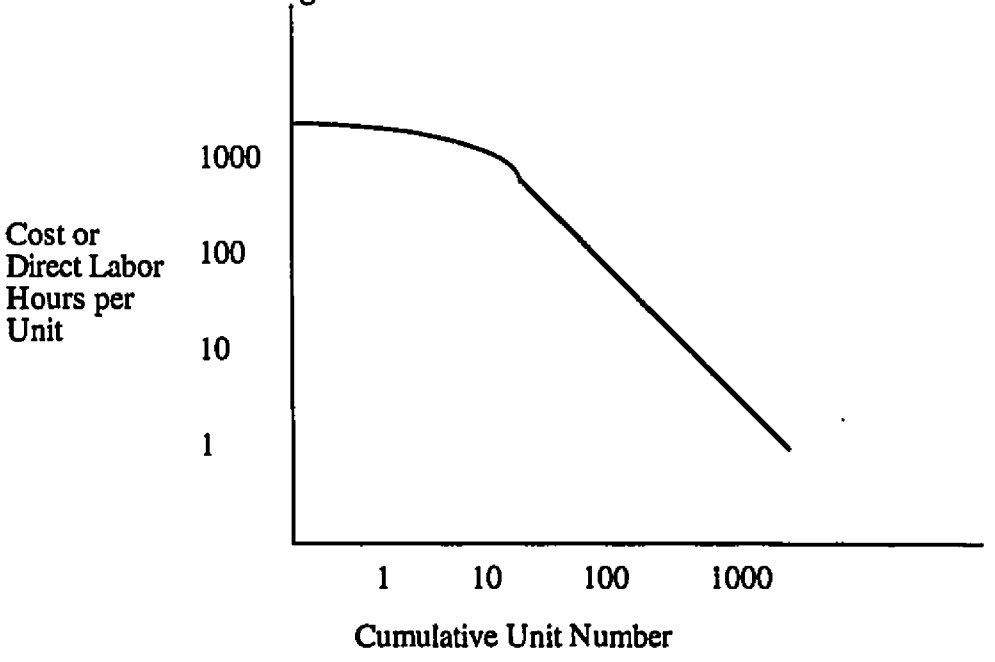


Figure 4: The Stanford-B Model



The former *plateaus* down, implying that there is an upper ceiling as to how much a person can improve, while the latter shows that initially the learning rate might be a bit slow. A linear model can be expressed in the form

$$y = ax + b$$

We can also apply a quadratic model to the learning curve for more accuracy. The equation for a quadratic model

$$y = ax^2 + bx + c$$

Most studies use the non-linear model,

$$y = ax^{-b}$$

By simply changing the notation, a = K and n = -b, we can write this equation as

$$y = Kx^n$$

where K is the input cost for the first unit.

The progress ratio, p, is defined by

$$p = 1 - 2^n$$

A good measure of the learning is the percentage improvement, L, which is given by

$$L = 2^n \times 100$$

2.3 Parameter Estimation

In order to mathematically describe the above equations, we need the exact values of the constant parameters, such as p and n (given above). Several studies have been carried out to estimate these parameters. The most common value reported in literature for the progress ratio, p, is around 0.20 (or an 80% learning curve percentage, L) [29][60], but there is a huge variance in these results [24]. Besides finding these parameters, another important aspect is to analyze the degree to which any data fits these models, as done by Hirsch [29]. The well known *pearsons' correlation coefficient, R,* can be used for this purpose. The absolute value of R represents the degree of fit on a scale of 0 to 1. Whereas it is easy to find R in linear equations, it can be cumbersome to do so for the equations of the form

$$y = Kx^n$$

However, a simple transformation of this equation can be made by taking the logarithms of both the sides, which gives us a linear equation, as shown below.

$$\log y = \log Kx^n$$
$$\log y = \log K + \log x^n$$

Let

$$\log y = Y$$
$$\log x = X$$
$$\log K = c$$

where c is a constant, then

$$Y = c + nX$$

where the slope of a plot of Y against X is n. We thus need to analyze how well the data fits this equation.

2.4 Labor vs. Organizational Learning

The improvement due to on-the-job, first order, learning is termed as labor learning. Determinants of this might include the person's general experience, specific experience on jobs of a given type, education, gender, age, etc. [42]. We can group all these factors under the category *personal capabilities* and study whether or not there is any relationship between the personal capabilities and the progress in performance. However, besides labor learning, there can be other types of learning also. According to Arrow [5],

> *"... learning takes place only as a by-product of ordinary production. In fact, society has created institutions, education and research, whose purpose is to enable learning to take place more rapidly."*

Here, Arrow is referring to the second-order learning, and says that even field theorists like Gestalt who stress the role of insight in the solution of problems have to assign a significant role to previous experiences. In 1965, Levy [42] proposed that to improve the planning process, the learning behavior of the firm (besides that of the individuals) also needs to be understood. Earlier, in 1952,

Hirsch [29] had found that about 87% of the changes in direct labor requirements were associated with the changes in technical knowledge, which is a form of Organizational Learning. More recently, in 1991, Adler and Clark [2] formed a Learning Process Model, in which the Organizational Learning is further attributed to (i) *engineering/technology* changes and (ii) to the labor *training*. A third factor which contributes to organizational learning is *management motivation*, which is discussed next.

2.5 Management Motivation

Motivation and incentives are more important in labor-intensive manufacturing than in machine-intensive manufacturing [60]. It is found that in the latter, the phenomenon of 'plateauing' is much more likely to occur [7]. Several researchers have worked on the effect of incentives and wages on performance, and have found that an incentive during learning period leads to laborers learning faster [58]. Considering its importance, no study on progress functions can be complete if the management aspect is ignored.

3.0 The Personal Software Process

An integral part of our experiment is the personal software process, PSP, designed by Humphrey of Software Engineering Institute [30]. This process provides an individual with an in depth second order learning program. The principles of the personal software process are to help the individuals to [30]:

- know their own performance: to measure their work, to recognize what works best, and to learn how to repeat it and improve upon it;
- understand variation: what is repeatable and what they can learn from the occasional extremes;
- incorporate these lessons in a growing body of documented personal practices.

In this experiment we had deployed 12 software developers working on 10 short programming projects at a rate of one project per week. These projects were in C++, a language which they all were not familiar with and which they had to learn with the passage of time. For each project, they kept precise track of even the minutest measurements, such as:

- Every Line of Code, LOC, (new, reused or modified)
- Every Defect found (from 190 different defect types)
- Phase of Injection and Removal of Defects (from 8 development phases)
- Time spent on fixing each defect
- Time spent on each activity and phase of the project
- Estimated and actual values of project size and time

The above measurements seem to be ample for individual progress rate studies for each subject. However, for making comparisons amongst the subjects, we needed some consistency in their measures. Therefore, all the software developers had to use the same standards throughout the course of the experiment, such as the same:

- System Environment
- Physical Environment
- Programming Language
- Defect Type Standards

- Logical LOC Coding Standards
- Physical LOC Coding Standards
- Data Collection Techniques
- Data Base Package
- Personal Process

As required by PSP, complete details regarding the background of each subject were taken from them. For this purpose, our measurement instrument included a 6 page form, given in Appendix A. Later the subjects had to appear in a 30 minute interview to verify and validate their personal data. This data included:

- Level of Education
- Type of Education (Majors, Minors)
- Total Job Experience (Full-time and Part-time)
- Total Programming Experience
- Languages used (total LOC and months for each)
- Database related packages used
- Spreadsheet and Graphing packages used
- Statistical Tools used
- Experience with Software Process Engineering related tools and courses
- Object Oriented Design experience
- Design methods and Formal methods used

The subjects were given an intensive training program through which they were taught various software engineering techniques, hence providing them a second-order organizational training. This training program introduced one new concept every week, which helped them in improving their software development process. These concepts included:

- Measuring and Tracking the project
- Software Project Planning
- Statistical Methods for Estimating Size and Time
- Schedule and Resource Planning
- Code Reviews & Defect Prevention Strategies

- Structured Design Methods
- Cyclic Personal Process

During this program, the subjects attended three hours of interactive lectures every week with an opportunity to discuss, amongst other things, the measurement techniques and goals. These techniques form a part of the measurement instrument, which is described in the next section.

4.0 Problem Definition and Our Model

An experiment process provides a basis for the needed advancement in knowledge and understanding [15]. We build models of the software process, test the hypotheses about these models through experimentation, and then use this information to refine old hypotheses or develop new ones [15]. Also, for research results to be meaningful, software measurement must be well grounded in theory and empirical results must be obtained through well designed experimental work [6].

Figure 5 represents our model, which is an extension of the models developed by Levy in 1965 [42] and Adler and Clark in 1991 [2]. It shows that any continuous development activity takes place concurrently with first-order and second-order learning, which may lead to improvement. We have developed instruments through which we attempt to measure and test our hypotheses for this model. As is often the case, our instrument may not be perfect, since it is difficult to draw a distinction between the two types of learning. However, what we hope is that our results would help us to refine our model further as more and more replications of our work are carried out. This is analogous to Basili's *quality improvement paradigm, QIP*. This approach of iterations of hypothesizing and testing takes special importance in the adolescent field of software engineering because we need to improve significantly our knowledge of how software is developed and the effect of various technologies on it [15].

24

Figure 5: Progress Functions in the Software Process

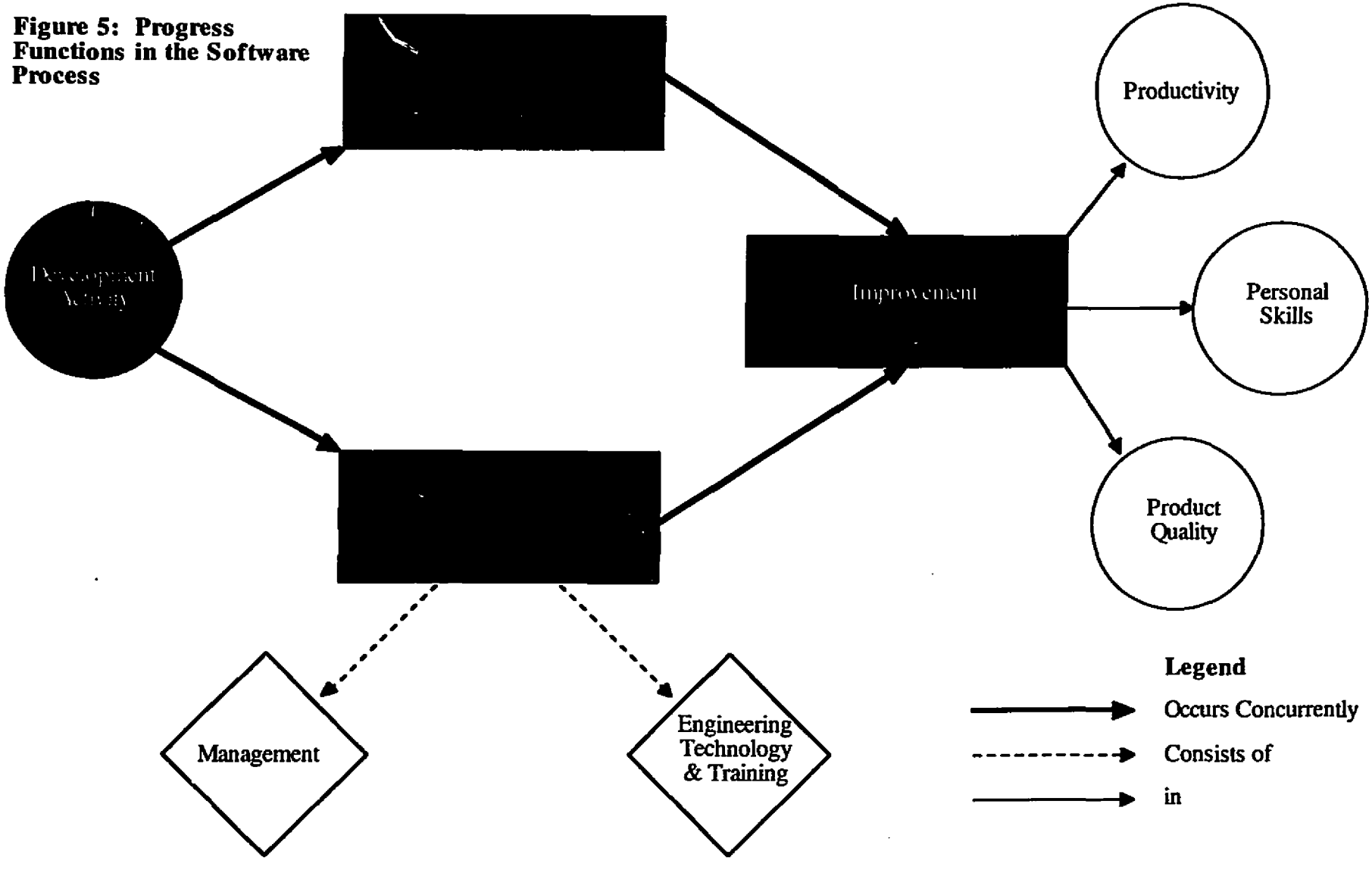


Figure 5: The Main Model (on previous page)

We can now state our problem definition as we go through our model in figure 5. This figure shows that concurrent to any development activity are the two types of learnings: first order and second order. The latter can be attributed to two factors: management motivation and the engineering technology and training. These learnings lead to improvement, which can be in productivity, personal skills or the product quality. Table 1 below describes how each item in figure 5 corresponds to a problem, which we intend to tackle.

Table 1: Problem Definition

Development Activity	Does the complexity of the development activity task affect the rate of learning?
First Order Learning	Is the first order learning related to the personal capabilities and/or experience of the individual?
Second Order Learning	Does the second order learning help significantly more than the first order learning?
Management	Does the management motivation affect the rate of learning?
Engineering Technology and Training	Do the individuals perform better due to the injection of technology and due to training?
Improvement	Which model (linear, quadratic or logarithmic) best describes the learning curve?
Productivity	What is the progress ratio in the improvement in productivity?
Personal Skills	What is the progress ratio in the improvement in the personal skills of the individuals?
Product Quality	What is the progress ratio in the improvement in the quality of the product? How are the productivity and the product quality related to each other? Do these two variables depend on the complexity of the development task?

To solve each problem, we need a measurement instrument. We will now describe our model in depth, which is used in solving the above problems. This would require more specific details of the problem also. Our sections are sequenced as follows (where parenthesis refer to the box shapes in the main model in figure 5):

- 4.1 Improvement (black box)
- 4.2 Second order learning (brown box)
- 4.3 First order learning (brown box)
- 4.4 Development activity complexity (black circle)

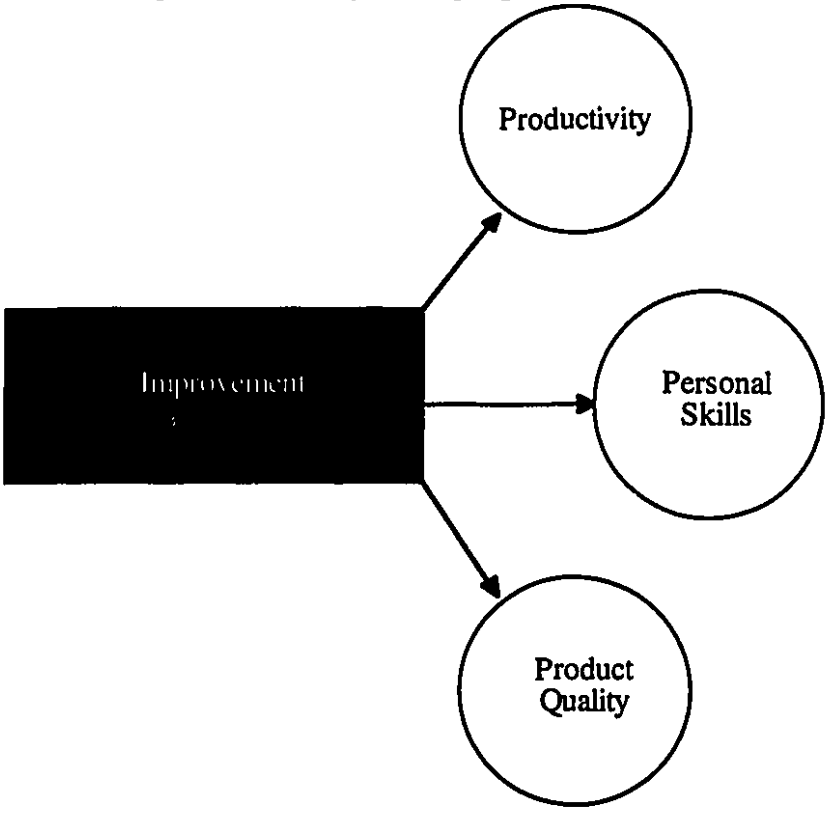
Table 2 below shows the ordering of these sections as we describe the corresponding parts of the model.

Table 2: Parts of model and the section numbers

[illegible]	[illegible]
Development Activity Complexity	4.4
First Order Learning and its relationship with personal capabilities	4.3
Second Order Learning	4.2
Management Motivation	4.2.1
Engineering Training and Technology	4.2.2
Improvement	4.1
Progress in Productivity	4.1.1
Progress in Personal Skills	4.1.2
Progress in Product-Quality	4.1.3
Comparing the second order learning with the first order learning	4.2.2
Relationship between Productivity, Quality, Complexity and Cumulative Output (4 variable model)	4.4

4.1 Model for measuring Improvement

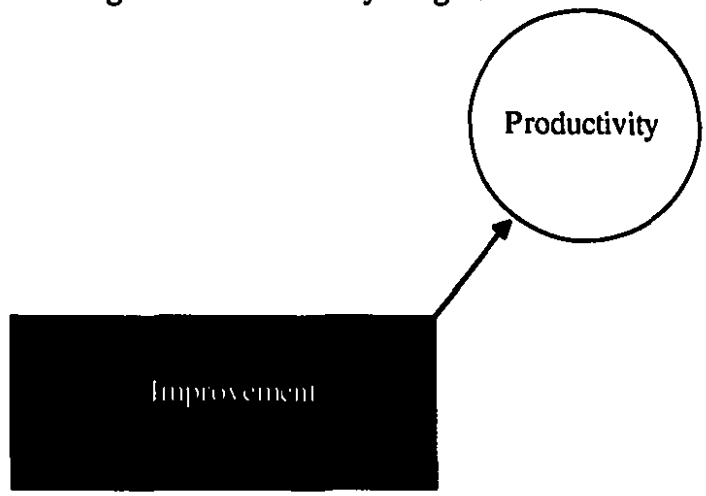
Figure 6: Three types of progress functions



Engineering measurements can be generally characterized as either process or product related. [52] We have emphasized on both these types by studying three different progress functions, represented by circles in figure 6: (i) progress in productivity, (ii) progress in personal skills and (iii) improvement in product-quality. Whereas the product-quality represents the nature of the product produced, the personal-skills represent an inherent part of the personal process used by the software developers, and the productivity represents the rate of development of the product. These three types are now discussed in detail in the following three sections.

4.1.1 Productivity Progress Functions

Figure 7: Productivity Progress Function



Productivity can generally be defined as completing an activity as expeditiously as possible, and has economic connotation of goods and services produced per unit of labor or expense [33]. The programming speed can be measured in output produced per unit labor time. Whereas the latter can be easily measured in programmer-hours or programmer-months, problem arises in measuring the output produced. The simplest and perhaps the most commonly used product metric is the lines of code, LOC, and according to Basili, it should be regarded as a baseline metric to which all other metrics should be compared [29].

But a fundamental problem is that of knowing exactly what is meant by the phrase "lines of code." The unit lines of code per programmer-month is found to consistently penalize high-level languages, since they can encode any logic in much fewer statements than a low-level language would. Hence it is difficult to compare productivity between programs of different languages [33]. In our case we solved this problem by imposing all the subjects to use the same programming language. However, even within the same language, there can be a variety of measures associated with the generic concept of lines of code [12]. For example, for measuring effort, comments should be counted in the source code, but for approximating functionality, the executable statements (and data declarations) may probably be a better measure [12]. According to T. C. Jones [33], this problem is not serious provided it is recognized. The key is to state the counting rules explicitly, and then to adhere to those standards.

For programmer-productivity, we chose the metric *logical LOC per hour*. A logical LOC is more accurate than a *physical LOC* since it uses the number of tokens as a micromeasure of the number of units of information. Logical lines are invariant to editing changes, uniquely definable and correlate better with development effort [30]. These tokens may be operators, operands or any other item, provided they are explicitly stated in the coding standards and then devotedly followed.

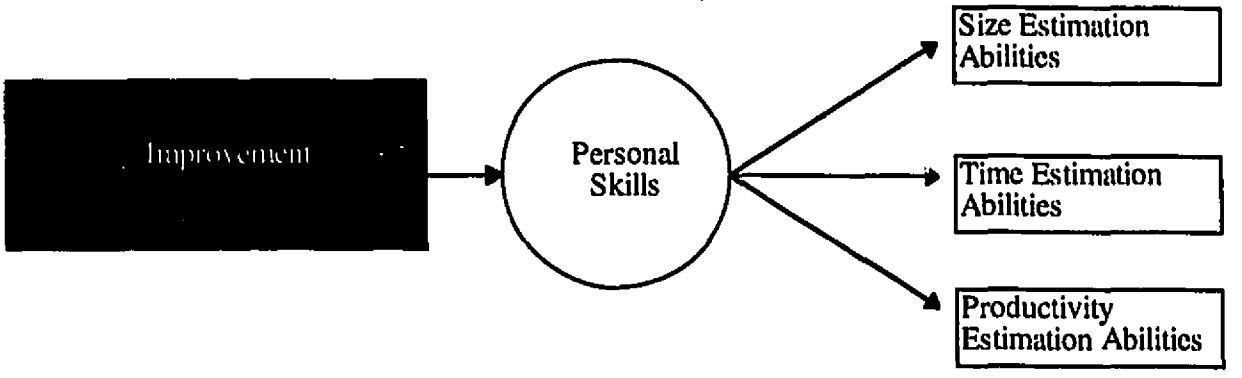
Appendix B shows the logical LOC counting standards which were used by all the subjects. One attribute of a good measuring standard is that it should be automatically countable. Physical LOC have an advantage over logical LOC because it is easy to develop their automatic counters [30]. We overcame this difficulty by defining physical LOC coding standards in such a way that each physical LOC contained one and only one logical token. These physical LOC coding standards are given in Appendix C. The subjects then developed their own physical LOC counters, which indirectly measured the logical LOC. In this way not only did we get accurate data automatically, but also managed keep consistency amongst all the subjects so that they could be compared amongst each other.

Finally, another important point to consider is the number of Reused and Modified LOC. Often experimenters include the number of Reused LOC in the productivity measurements as in [26]. Often the programs consist of over 50% [13] of reused code (it was 54% in our case). Obviously reused code requires much less effort, one fifth of a new LOC by one estimate. According to T. A. Standish [54], modified lines should count as 1/2 reused and 1/2 new. In our case, we have averaged out by not counting the reused code, and giving full weight to modified code. As mentioned before, it is not important whatever standards we pick, as long as we are consistent. Hence we get,

$$\text{Programmer Productivity} = \frac{\textit{No. of New or Modified Logical LOC}}{\text{Programmer Hour}}$$

4.1.2 Personal Skills Progress Functions

Figure 8: Personal Skills Progress Functions



Most of the studies on progress functions use a count of tangible objects to measure the rate of progress. Our work not only investigates productivity-oriented aspect of progress but also investigates progress in the personal skills of the individual developer. We have used the *estimation ability* of a person to model the skills. This estimation can be of size of the programming task, time required to complete the task, or even of the subjects productivity, as shown in figure 8.

Our metrics for measuring the progress in estimation are the Size Estimation Error, the Time Estimation Error and the Productivity Estimation Error. Before each project, the subjects estimate the size of the job (in logical LOC) and the expected time required to accomplish it. Using these two metrics, or otherwise, they also make an estimate of their productivity. After completing the project the actual values of size, time and productivity are used to find the percentage error. Hence,

$$Size\ Estimation\ Error = \frac{|Estimated\ LOC - Actual\ LOC|}{Actual\ LOC} \times 100$$

$$Time\ Estimation\ Error = \frac{|Estimated\ Time - Actual\ Time|}{Actual\ Time} \times 100$$

$$Productivity\ Estimation\ Error = \frac{|Estimated\ Productivity - Actual\ Productivity|}{Actual\ Productivity} \times 100$$

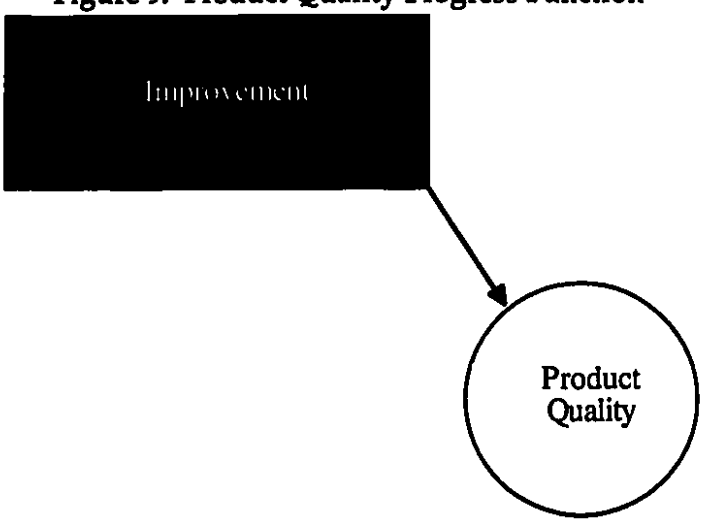
Humphrey [31] has shown that at the organizational level, the percentage error in size estimation for a group of IBM employees decreased with the passage of

projects. One goal of our experiment is to determine whether or not the same phenomenon occurs at the personal level.

The subjects were given second-order training to improve their size estimation ability. This is described in detail in section 4.2.2.

4.1.3 Quality Progress Functions

Figure 9: Product Quality Progress Function



One major difference between our work and most of the past studies is that our work involves measuring the progress in *quality*, besides measuring the productivity progress. In measuring progress, we use cumulative output as a variable which affects the progress. Hence the progress in productivity or quality varies as the cumulative output varies. However, productivity and quality are mutually dependent on each other as well, besides being dependent on cumulative output. At the organizational level, Deming [59][61] has shown that improved productivity is a direct result of improving quality, while Putnam [45] has empirically shown that the defects decline rapidly as the productivity improves. At the personal level, Humphrey [30] has observed that improvements in product quality do not seem to reduce productivity. According to Remus and Ziller [47]:

> *"... the quality of the software development process is continuously improving. Programmers are becoming more proficient at applying defect removal techniques. Therefore, it is desirable to be able to predict quality on the basis of measurements made on the software as it is being developed."*

Since in measuring the progress in productivity or quality, it is not sufficient to just consider the cumulative output, we used a model involving all the three attributes.

The term *quality* has many attributes, and so any attempt to define it should be based on the principle of meeting the users' needs [30]. *Defect rates* (during product development [33]), *defect removal efficiency*, and the *number of defects discovered after the product is shipped* [47], are often cited in the literature as reasonable starting points [36]. However, the latter is inappropriate for our study as the developed programs do not undergo extensive usage. Thus, in our study we used defect rates and the defect removal efficiency.

The defect rate was normalized to defects per hundred LOC. We followed the suggestion of T. C. Jones [33] and lumped together all the defects, regardless of source of origin, and counted them as a single variable, *defects*. Appendix H shows the defect types standard used in our experiment. Similarly, the defect removal efficiency of all the phases was also combined. Here also, the reused code was not used since it was assumed that it does not contain defects. If on the contrary it did, then naturally the programmer had to fix that code, hence making it `modified' instead of reused. Hence our metrics were as follows:

$$Defect\ Quality\ (Defects\ /\ 100 LOC) = \frac{No.\ of\ Defects\ Found}{Total\ New\ \&\ Modified\ LOC} \times 100$$

$$Defect\ \text{Re}moval\ Rate\ (Defects\ \text{Re}moved\ /\ hour) = \frac{No.\ of\ Defects\ Found}{Total\ Time\ taken\ to\ find\ \&\ fix\ the\ defects}$$

The subjects were given second-order training to use code reviews in order to improve quality. This is described in detail in section 4.2.2.

4.2 Model for Analyzing Second-Order Learning

Figure 10: Model for second order learning

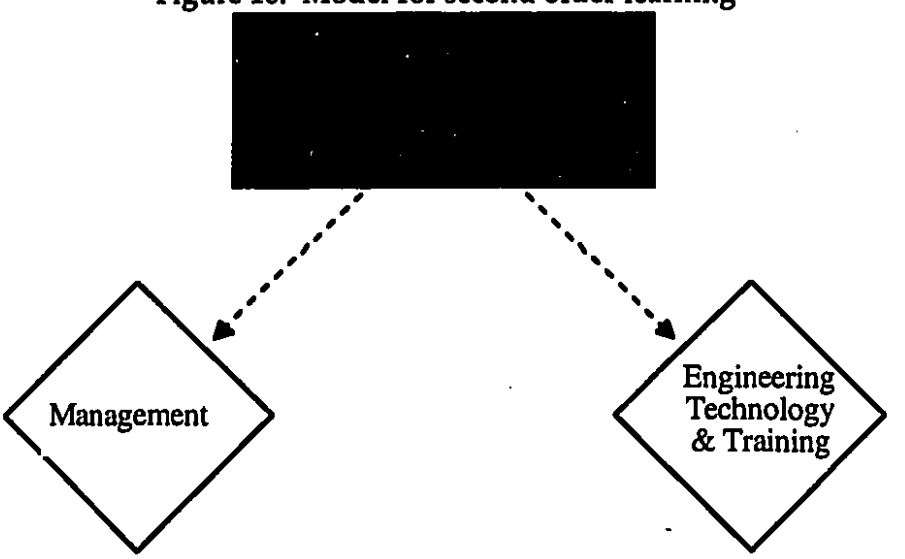
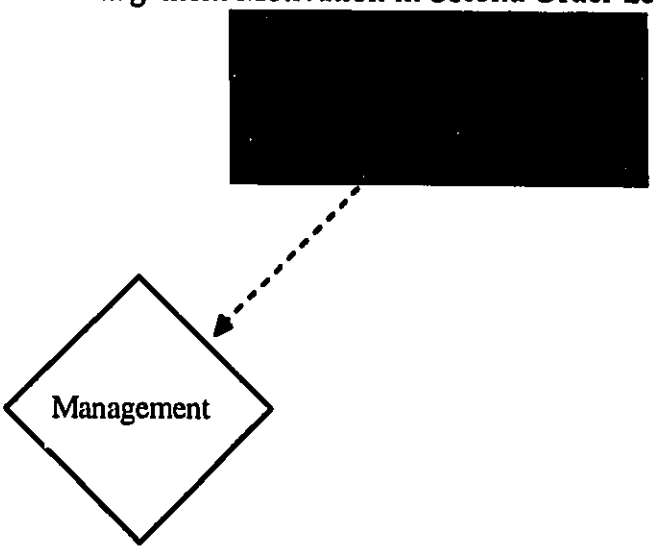


Figure 10 shows that the second-order learning can be divided into two categories: (i) the motivation and incentives given by the management, (ii) the technology changes and training provided by the engineering department. In addition, there can be second order learning from other sources of knowledge also, such as that from the customer for whom the software product is developed, but in our case such sources do not apply. Hence given below is the measurement instrument used to analyze the first two categories only.

4.2.1 Management Motivation

Figure 11: Management Motivation in Second Order Learning



Zultner [61] has interpreted the fourteen points of Deming's approach to adapt them to MIS management. Amongst other things, these deal with the motivation and enthusiasm aspects of the labor force. Enthusiasm can be contagious, and people tend to perform better in an optimistic environment than in the "won't work" environment [31][32]. As mentioned before, incentives and wages on performance during learning period lead to developers learning faster [58]. Therefore, we decided to design an instrument to keep track of the programmers Motivation, Interest and Satisfaction.

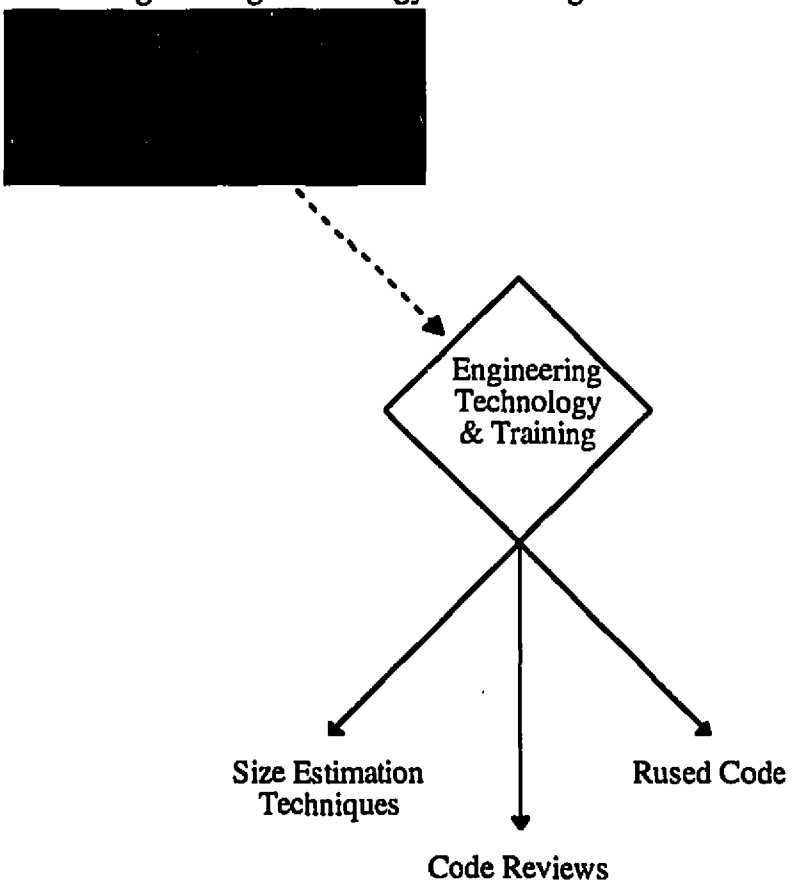
The instrument comprises of a survey form which each subject fills after each project. The survey form is attached in Appendix C. The survey asks the subjects to rate the following metrics:

- Motivation: defined as the desire or incentive given to subjects by the management.
- Interest: defined as the `willing attention' the subjects took in the projects.
- Satisfaction: defined as the pleasure the subjects receive upon fulfillment of a project.
- Usefulness: the extent to which the subjects learnt from the project

The above metrics are subjective. These were quantified on a scale of -5 to zero to +5 (11 categories), and also labeled, such as from `Completely Unmotivated' to `Neutral' to `Extremely Motivated.' Furthermore, the subjects were asked to give their responses separately for the coding project and for the data collection process. Finally, during the personal interviews through the course of the experiment, the students were given clarifications on any ambiguities they had regarding this measurement instrument. Due to these above mentioned steps, we are quite confident that our subjective data quite accurately depicts how the programmers felt at each stage of the experiment.

4.2.2 Engineering Training and Technology

Figure 12: Engineering Technology & Training in 2nd Order Learning



As described above, the software developers get second-order learning from the training given to them through the personal software process program. Two key aspects of this training program are the Size Estimation method and the Code

Reviews (see figure 12). The third aspect, reused code, is of lesser significance in our work, and is discussed later.

These training mechanisms are injected after the 3rd and the 6th projects respectively, out of the total of nine programming projects and one non-programming one. A comparison of student data before the injection of these technologies and after their injection can give an idea of their benefits. Whereas before the injection, the subjects' improvement can be attributed solely to first-order learning, after the injection it is affected by both types of learnings. Based on the data before the technology injection, we can estimate the improvement trend, i.e., we can find the equation of the learning curve before technology injection and extend it forward to predict what the improvement should be in future if only first-order learning continues. We can compare these predictions later with the actual data after the technology injection, which includes the second order learning also. Hence any increase in learning beyond our predicted values could then be attributed to the second-order learning factor. Of course such estimates have several uncertainties involved, and our confidence in them cannot be hundred percent. Such a confidence depends on the number of subjects, the number of measurements taken, the experimental design, etc. These issues are discussed in the experimental design (section 5) and in data analysis (section 6).

Given below is a description of the two training methods. Also described below is the concept of Reused Code, which, if done formally (using libraries or otherwise) as in our case, is also a second-order engineering technology mechanism which helps the programmers beyond the autonomous learning.

SIZE ESTIMATION

Various methods exist in the literature for estimating the cost, size and time of a programming task, e.g., Boehm's COCOMO [16]. However, such models seem to work in certain environments, but not in others [12], and it is hard to tailor them to the characteristics of different environments of the individuals.

At the PSP level, we need an estimation procedure which can utilize the conceptual design at the very early (planning) stage of the project to produce an estimate. One such method is Albrecht's Function Points [3]. However, such methods have a low reliability [36] and are principally used for estimates in commercial data processing [30], instead of small individual projects. Therefore, we used a PROxy-Based Estimating (PROBE) method, designed by Humphrey [30], in which each individual uses his or her own past data to produce a size estimate

for each Object in the conceptual design. These Object sizes are then mapped to get the total program size, using the database of previous programs. Hence this method is customized for their needs of the subjects. Appendix G shows the template used by subjects for estimating size.

CODE REVIEWS

Literature is full of advantages gained by carrying out code reviews. According to Fagan [26], there is evidence that early experience gained from inspections causes programmers to reduce the defects in the later phases. Moreover, reports from industry [50] ag e that code inspections can be up to 20 times more efficient than testing. In an experiment involving professional programmers, Code Reading detected more software faults than did functional or structural testing [13].

As a part of PSP, Humphrey has also emphasized code-reviews or inspections, hence providing software developers with 2nd order training. Special emphasis is paid to those defects which have been occurring most frequently in the past. For this purpose, the subjects analyze their past data and prepare Pareto charts of the most frequently occurring defects. A checklist of these defects is then made, which aids in the code review process (see sample check list in Appendix G).

In order to prepare Pareto charts, detailed past data of defect types is a prerequisite. Therefore, besides measuring the phases of injection and removal of defects and the time taken to fix each defect, the subjects were also required to categorize the defects using a defect types standard, which contained over 190 different defect types. These defect types, though not exhaustive, were more than adequate to give a good insight into the nature of the defect. Subjects were also required to give a one to two line explanation for each defect. The list of defect types was prepared by the author based on his personal experience, Ripley's analysis of syntax errors [48], C-language library's include file errno.h, Leblanc and Fischer's case study of run-time errors in Pascal programs [41], Humphrey's defect type standard [30], and Turbo Pascal's Error Message Codes [40].

Defect analysis is important for defect prevention. Although a qualitative causal analysis can provide feedback on each individual defect, it is akin to studying the ocean floor with a microscope [18]. The other alternative is to use a quantitative analysis, using statistical defect models or software reliability growth [18]. These methods collect precise defect data from a large number of projects, and then use using techniques such as defect control charts or pareto charts, to decrease the number of defects in the future projects. And this is precisely what our subjects

did in this second-order training. Note that the main theme behind these themes is to first measure and then to improve the quality. As Walrad and Moss [59] put it, *"Quality experts are certain that measurement is essential to improving quality. In other words, measurement drives quality."* Apart from using this data in Code Reviews, we did not have any objective on our part to analyze it by phase or type (see future research considerations, section 7.0).

REUSED CODE

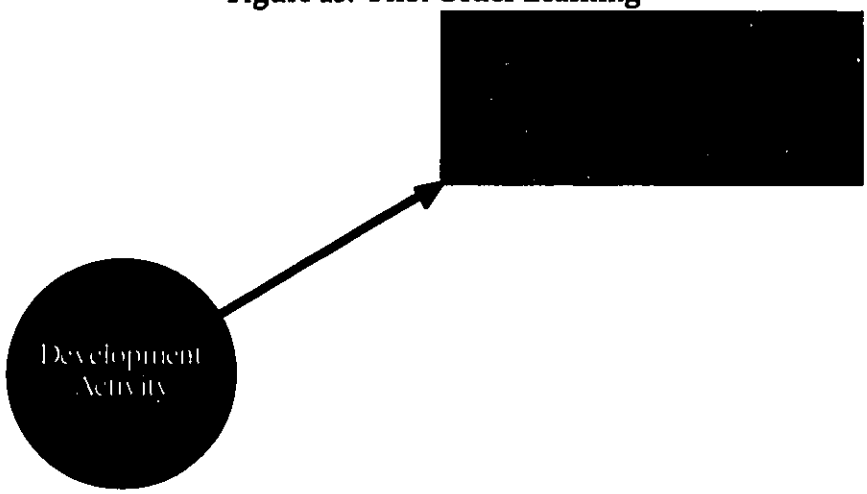
The concept of Reusing Code, if done formally using libraries or otherwise, can be considered as a second-order learning mechanism. However, unlike in the above two cases (Size Estimation training after project 3 and Code Reviews after project 6), in case of Reuse we cannot study the effects of injecting a technology because it starts immediately after the first project. Moreover, not all the subjects are required to Reuse the code, and they do so depending on their specific needs. Hence although we can study what percentage of the code was reused, we cannot deduce how much learning or improvement took place due to the Reuse factor alone. However, by quantitatively studying the amount of Reused code, we can obtain a qualitative impression of its importance, and hence indirectly about the importance of the second-order technology and training.

There has been an on-going debate in the last few years regarding the merits of software reuse [8]. Software engineers have discussed whether reuse provides any major insight into the development process or if it is just another development technique, which may be helpful in some contexts and inappropriate in many others. According to Barnes and Bollinger [8], reuse is a fundamental paradigm of development and until it is better understood, significant reductions in the cost of building large systems will not be possible. Jones [34] speculates that by the year 2000, the percentage of new applications may be only 10-15 percent, and hence software-reuse would be one of the primary factors in the development process. Due to its importance, software reuse is healthy in the form of teaching and application of reusable software abstractions, and the data structure and algorithm books are full of them [54].

In PSP, since several of the projects assigned to the subjects are related to each other, reuse plays an important role in our experiment. Thus in order to study the effects of, we would be studying the percentage reuse in each project as well as the overall cumulative reuse through the course of the experiment.

4.3 Model for Analyzing First-Order Learning

Figure 13: First Order Learning



In the previous section (4.2), we discussed ways to study how much second order learning can contribute in addition to the first order learning. In this section we analyze some internal attributes of the first-order learning.

Determinants of first-order learning might include the person's general experience, specific experience on jobs of a given type, education, sex, age, etc. [42]. We can group all these factors under the category of personal capabilities and study whether or not there is any relationship between the personal capabilities and the progress in performance. For this purpose, we can plot a graph of progress (in productivity, personal skills and quality) on Y-axis against a capability-index of the 12 software developers. The progress of the subjects can be the mean of the following five progress ratios, p:

- p in Size Estimation Abilities
- p in Time Estimation Abilities
- p in Productivity Estimation Abilities -- p in Programmer-Productivity
- p in Defect-Quality

The first three ratios are for subjects' Estimation Abilities while the latter two are for subjects' performance. The main problem now is to assign the capability-index to the subjects. We have decided to use two main factors in calculating this index:

(i) Subjects Past Software Development Experience and Education.

Various factors can be included in the analysis of past experience as listed below. The values for these factors were provided by the subjects themselves, based on their own estimates. A six page questionnaire was used as the primary instrument to gather this data (see Appendix A). These factors include:

- Total number of languages programmed in
- Total experience in programming
- Educational excellence level and degrees obtained
- Total full time and part time job experience
- Experience with software packages
- Experience in Object Oriented Design and Software Engineering

(ii) Subjects performance in the 10 projects

The metrics used here are based on the absolute performance of the subjects during this experiment. Recall that it would have been difficult to compare the performance of the subjects had we not defined our coding and defect standards. The metrics used in this case include:

- Productivity in LOC/hour
- Defects / KLOC
- Defect Removal Rate in Defects Removed/hour
- Grade given to the subjects in the PSP course

In the section on data analysis (section 6.5), we describe in detail how the Capability Index was calculated from the above factors.

4.4 Model for Analyzing the Development Activity Complexity

Figure 14: The Development Activity Complexity

Nearly all the learning curve studies have been carried out with the assumption that the nature and *complexity* of the production or development activity stays constant. However, in the field of software development, such an assumption would be unreasonable. Intuitively, the more difficult a task is, the lower the productivity. Card and Agresti [48] have done empirical work on Design Complexity and shown that it has a strong correlation (R = 0.83) with error-rate (in errors per KLOC) but no correlation (R = -0.49) with productivity (in LOC per hour). If our learning curves suggest an improvement, it can be because of decreasing complexity. Therefore, we had to treat the complexity of the work done as an independent variable. It is an independent variable since we did not control it in this experiment. But complexity is a very general term and may many different terms, so we need some standard measures for quantifying it.

In the field of software, several measures of software complexity have been used. The two metrics which we chose need no introduction[14]: McCabe's cyclometic complexity [43] and Halstead's software science effort [19]. Software Science has several other metrics also, such as the software science length, but in our case we are more concerned with the complexity rather than the length of the program (though it is true that for maintenance purposes, length adds up to complexity). In order to accurately and efficiently carry out measurements using these two metrics, we used a tool, PC-Metric, developed by SET Laboratories, Inc. [44]. This tool has been referenced in other research work also, for example, in [27]. Appendix J shows sample complexity results generated by this tool.

According to Curtis, et al. [20], there is no exact mathematical relationship between the McCabe's and Halstead's metrics, but one should not be surprised if a significant correlation between them occurs. We have availed of this opportunity of having gathered a large amount of accurate data to perform a comparative analysis of the two metrics, besides using them for our learning curve studies. This comparative study is important to our work because if the two metrics are giving completely differing values of complexity, then it would decrease our confidence in how accurately we have quantified this variable.

Generally, complexity is thought to give us an understanding to such software characteristics as maintainability and reliability [52]. However, our concern here is not these post production issues, but the effort required during the production of the product, because it is this effort which we are measuring in our progress studies of productivity. According to Curtis, et al. [20], there is empirical evidence that software complexity metrics were related to the difficulty programmers experienced in understanding and modifying software. Others such as Sunohara, et al. [57] also share similar views. Basili [14] defines effort here as the number of man-hours spent from the beginning of functional design to the end of acceptance testing, which agrees with our definition. However, he points out that how well the various metrics really measure or predict effort or quality is still an issue in need of confirmation since none of these two metrics seem to manifest a satisfactory explanation of effort or quality of the program. But he does believe that if the programs are developed by individuals (as in our experiment), the metrics' correlations with actual effort seem to be strongest.

Curtis, et al. [22], feel that one potential use of complexity metrics is to get feedback during the development of the program. During our experiment, we found that more complex the problem we gave (according to our estimates), more complex was the solution which the programmers developed (based on two different quantitative metrics described below). Hence our assumption here is that the complexity reflects the effort required to do a program, and hence affects the productivity of the programmer.

5.0 Research Method

Having defined the problem along with the general approach we are taking to tackle it, we now describe our *measurement instrument* and the *experiment design*. The term *research method* refers to the entire study, in which we carry out the following activities:

- Defining the Objectives
- Setting the metrics
- Validating our Goals and Metrics
- Designing the experiment for the subjects
- Conducting the experiment
- Collecting data and validating it
- Analyzing the data

The term *experiment design* applies only to the part "Designing the experiment for the subjects" listed above. After describing the measurement instrument (section 5.2) and the experiment design (section 5.3), we give details of our data collection process (section 5.4). But first (in section 5.1), we give a brief background of some research concepts, terminologies and definitions, which will extensively be used in the remaining portions of our work.

In this chapter, often we describe standard concepts, terms and definitions from the past literature. These descriptions are a prerequisite to what follow after them. However, some readers might already be familiar with these details. Therefore, these descriptions are given in boxes, and may be skipped if needed.

5.1 Concepts, Terminology and Definitions

The field of software measurement has been criticized for poor empirical methods and for a lack of theoretical foundations [6]. Useful measures can be developed under a well-grounded measurement theory framework. It is thus important that empirical work in software engineering help strengthen the framework by explicitly defining concepts, terminology and definitions used in the empirical work. This is the purpose of this section with reference to our work.

A theory behind an experiment can have at least three features [38]: Constructs, Relations and Hypotheses. These are given in Box 1.

Box 1: Constructs, Relations and Hypothesis

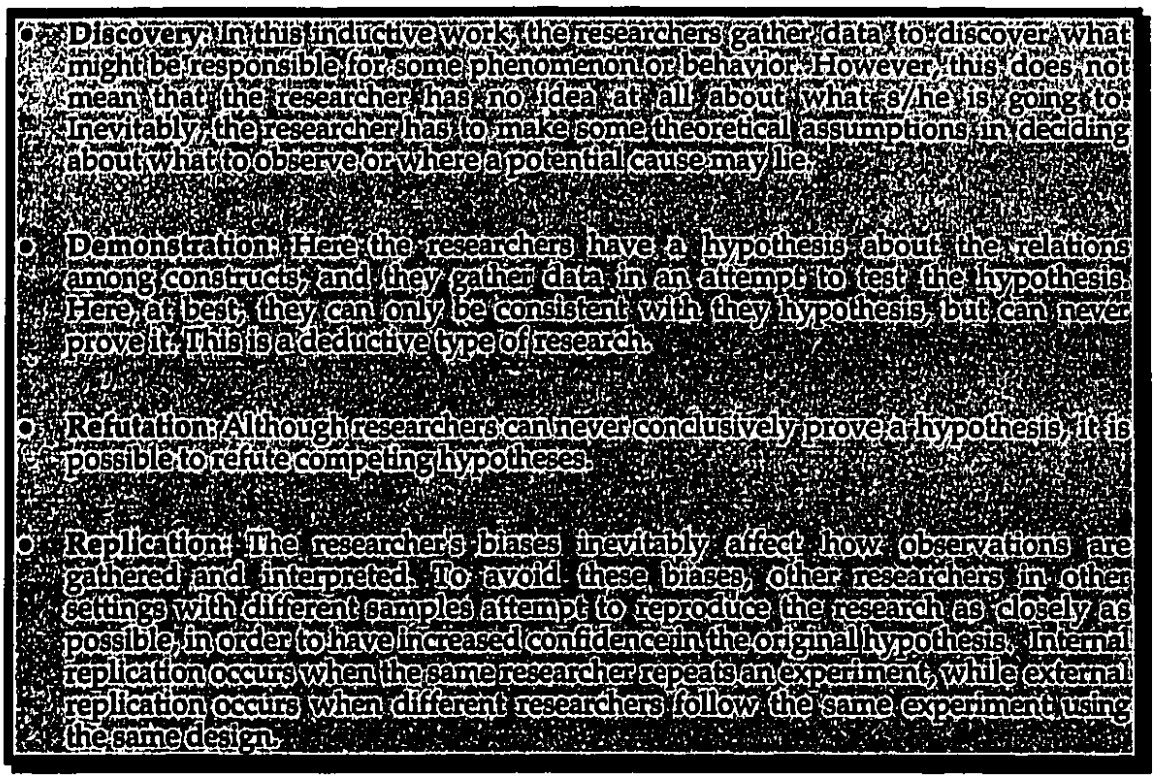
- **Constructs:** These are the factors which are of some interest, e.g., quality, productivity, etc. The theory attempts to explicate or account for these constructs in some way. A construct is a concept, but it has an added meaning of having been deliberately and consciously invented or adopted for a special scientific purpose [37].

- **Relations:** These specify which constructs exert effects on which other constructs, under varying conditions. For example, better 'quality' leads to 'higher productivity' if 'complexity' is kept constant. Here, 'quality' is the causal construct, while productivity is the affected construct. The relation between the two constructs is contingent upon the condition 'complexity.'

- **Hypotheses:** In our terms hypotheses and relations are quite analogous. A theory incorporates hypothesized relations with the observable variables (that can be used to measure the constructs). For example, LOC/hour and Defects/KLOC are two observable metrics. If there is no significant relationship between them, then we cannot accept the hypothesis that 'productivity depends on quality.' The variable representing the causal construct (Defects/KLOC) is the independent variable, while the one representing affected construct (LOC/hr) is dependent variable. Hence the 'variables' are concrete (but partial) representations of the abstract 'concepts.' These 'concepts' are similar to 'constructs.' However, they cannot be synonymous with a construct because any single construct can have different variables.

There can be four different purposes of any research which examines these hypotheses [38]. These are *discovery, demonstration, refutation* or *replication*. Descriptions of each are given in Box 2. These will be discussed in context, as we analyze the data.

Box 2: Discovery, Demonstration, Refutation and Replication



The development of a theory underlying an experiment (constructs, relations and hypothesis), might lead to the development of an instrument for measurement. An instrument must have several characteristics, most of which are validation criteria[45] [49] [38]. These include, amongst other ones, the content validity, the construct validity, the internal validity and the external validity. Other characteristics include interpretability, reliability, effectiveness, statistical conclusion validity and precision. Software engineering literature has details on these issues. However, they are listed again in Box 3, with brief examples on how they are pertinent in our case.

Box 3: Validitys and other related concepts

1. **Content Validity** [25][56]: It is the degree to which the score or scale being used represents the concept about which generalizations are being made. In other words, it assesses whether the instrument metrics are drawn from all possible metrics of the constructs under investigation. For example, to study management motivation given to a subject, are the metrics *satisfaction*, *interest*, *usefulness* and *self-motivation* sufficient or have we missed any.

2. **Construct Validity**: It is the degree to which both the independent and the dependent variables reflect or measure the causal and affected constructs respectively. In other words, are the metrics chosen artifacts of the data collection method or are they describing the true constructs? For example, does the metric defects/KLOC accurately correspond to the term Quality, or is it being used because it can be easily measured? It is also important to test the alternative hypotheses.

3. **Internal Validity**: It is the extent to which conclusions can be drawn about the causal effects of one variable on another. It raises the question of whether the observed effects could have been caused by, or correlated with, a set of unhypothesized and/or unmeasured variables. In other words, are there untested rival hypotheses for the observed effects [56] ? For example, even if our data indicates that lesser defects/KLOC lead to more LOC/hr, can we still state with confidence that better quality leads to higher productivity? Or, is it possible that the *motivation* and *incentives*, two variables which might not have been measured, affected the productivity?

4. **External Validity**: This is the extent to which one can generalize the results of the research to the populations and settings of interest in the hypothesis. For example, if our conclusion is that better quality leads to higher productivity, but our experiment was done only in an academic environment, can we generalize the results to programmers in an industrial environment?

5. **Interpretability**: This is the extent to which the value of a variable can be interpreted within the context of a measurement purpose [25]. Hence the method for defining the metrics should stipulate how a measured value is to be interpreted. For example, would 50 defects/KLOC be considered good quality or bad quality.

6. **Reliability**: It is the degree to which observed relationships are systematic rather than circumstantial [51]. A reliable measure is one that does not fluctuate randomly from one moment to the next. Similarly, a reliable instrument is the one which would yield stable and consistent results on repeated trials.

7. **Effectiveness [25]:** This refers to the extent to which a variable is measuring a construct relative to the other variables that are measuring the same construct. For example, in measuring quality, is defect rate (defects/KLOC during development) a better variable than the variable Total No. of Defects remaining in the product after shipment?

8. **Statistical Conclusion Validity [56]:** This is the extent to which the variables demonstrate relationships not explainable by chance or some other standard of comparison. The sample size and the number of data points play an important role in this case. For example, if we have only one subject and two data readings which show that increase in quality resulted in an increase in productivity, how much confidence can we have in concluding that this phenomenon could not have occurred by chance alone? Furthermore, although the productivity increases, is the increase significant enough?

9. **Precision:** It is the degree to which an instrument is capable of detecting slightest changes [49]. This implies that the instrument has a high degree of susceptibility (to stimulation) and of making relatively fine discriminations of changes in behavior. Hence precision is the sharpness or exactness of the instrument.

There are various models which incorporate the above mentioned factors into an overall experimental strategy. A discussion of such models and our choice of a particular model is discussed below.

5.2 Measurement Instrument

There are a number of approaches for defining software engineering metrics. We have used a derivative of one of the most prominent ones amongst them, Basili's Goal/Question/Metric paradigm, G/Q/M [7]. This derivative method incorporates some improvement suggestions, especially more validation tests [25]. One important aspect of G/Q/M is to define all your goals and metrics in advance and then follow them strictly, instead of getting data first and then observing the trends and patterns found in it to identify `interesting' goals.

We were clear in specifying our goals, as evidenced by the goals published in a workshop position paper [53] at the start of our experiment, submitted prior to the data collection stage. Below we describe the steps which we followed in our derivative of G/Q/M method:

- Step 1. Identify a set of goals based upon your needs
- Step 2. Define the Constructs which quantify these goals
- Step 3. Develop the metrics which provide the data for the constructs
- Step 4. Validate the goals, constructs and metrics.
- Step 5. Define and execute a mechanism for collecting and validating data
- Step 6. Analyze the data collected to study the goals

A detailed explanation of these steps follows.

Step 1. Identify a set of goals based upon your needs [9]

A Goal is considered to be at the conceptual level [11]. It is defined for an object (Products, Processes, Resources, etc.), from various points of view, relative to a particular environment. Listed below is our set of goals. The section number for corresponding measurement instruments is listed next to each goal.

- G1: Identify the Progress in Productivity (4.1.1)
- G2: Identify the Progress in Personal Skills (4.1.2)
- G3: Identify the Progress in Product-Quality (4.1.3)
- G4: Analyze the second-order learning through management motivation (4.2.1)
- G5: Analyze the second-order learning through training and technology (4.2.2)
- G6: Analyze the first-order learning (4.3)
- G7: Study the effects of change in Production Activity Complexity (4.4)

As mentioned in the problem definition, productivity (G1) and quality (G3) are not only related to the cumulative output but also to each other. We had originally planned to study the relationship between these three variables. Hence this was a sub-objective, since it is used in explaining other main objectives. During the course of the experiment we found that complexity (G7) also affects the productivity and quality. This meant that we now had a four-variable model, studying which could be a completely new objective. Hence before performing data analysis, we amalgamated our sub-objective with G7 to get a new definition of G7:

- G7: Analyze the 4-Variable Model of Productivity, Quality, Complexity and Cumulative Output.

Step 2. Define the Constructs which quantify these goals

This is the most difficult step since it often requires the interpretation of fuzzy terms like quality or productivity within the context of the development environment [9]. Sometimes the constructs do not fully satisfy the entire goal. In that case the missing aspects can be noted so that later interpretations of the results can be qualified appropriately. Table 3 lists the constructs for each goal.

Step 3. Develop the metrics which provide the data for the constructs

The goals/constructs are now formalized by making them quantifiable and the actual data needed for them is identified [9]. These data metrics can be considered as the variables needed to explicate our theoretical interests.

Table 3 lists the metrics along with their units, for each goal and construct. Note that most of the metrics are measured against cumulative output (or time), which in our case is represented by cumulative number of LOC. Since this metric is used extensively in our work, we will not repeatedly mention it in Table 3. Given below is Table 3, in which symbol G represents Goal, C Construct, and M is for metric.

Table 3: Goals, Constructs and Metrics

Goals	Constructs	Metrics	Units
G1: Progress in Productivity	C1: Programmer Productivity	M1: Programmer Productivity	LOC/hour
G2: Progress in Personal Skills	C2a: Project-Size Estimation Abilities	M2a: Percentage Size Estimation Error	%
	C2b: Project-Time Esitmation Abilities	M2b: Percentage Time Estimation Error	%
	C2c: Personal-Productivity Estimation Error	M2c: Percentage Personal-Productivity Estimation Error	%
G3: Progress in Product Quality	C3: Defect-Quality	M3(i): Defect Rate	Defects/KLOC

		M3(ii): Defect Removal Rate	Defects Removed/hr
G4: Analysis of Management Motivation in 2nd Order Learning	C4a: Motivation	M4a: Subject's appraisal of Motivation	Scale of -5 to 0 to +5
	C4b: Interest	M4b: Subject's appraisal of Interest	Scale of -5 to 0 to +5
	C4c: Satisfaction	M4c: Subject's appraisal of Satisfaction	Scale of -5 to 0 to +5
	C4d: Usefulness	M4d: Subject's appraisal of Usefulness	Scale of -5 to 0 to +5
G5: Analysis of Technology in 2nd Order Learning	C5a: Improvement in Size Estimation Abilities	M5a: Percentage Decrease in Size Estimation Error	%
	C5b: Improvement in Quality	M5b: Percentage Decrease in Defect-Rate	%
G6: First Order Learning: Relationship between Personal Capabilities and Progress Rate	C6a: Progress Rate	M6a(i): Progress Ratio, p in Size Estimation	Ratio
		M6a(ii): Progress Ratio, p in Time Estimation	Ratio
		M6a(iii): Progress Ratio, p in Productivity Estimation	Ratio
		M6a(iv): Progress Ratio, p, in Prog-Productivity	Ratio

		M6a(v): Progress Ratio, p in Defect-Quality	Ratio
	C6b: Subjects' Experience and Education	M6b(i): No. of Languages programmed in	Number
		M6b(ii): Total Experience in Programming	Months
		M6b(iii): Educational Excellence Level, Degrees obtained	Data mapped to a Quantitative scale
		M6b(iv): Total Full-time and Part-time Job Experience	Months
		M6b(v): Experience with Software Packages	Months/package
		M6b(vi): Experience with OOD and Software Engineering Courses	Number of Courses taken or related Tools used
	C6c: Subjects' Performance (in the 10 projects)	M6c(i): Average Productivity	LOC/hr
		M6c(ii): Average Defect Rate	Defects/KLOC
		M6c(iii): Average Defect-Removal Rate	Defects-Removed/hr
		M6c(iv): Grade obtained in the PSP course	Course G.P.A

G7: Analyze the Four-Variable Model	C7a: Programmer-Productivity	M7a: Programmer-Productivity	LOC/hr
	C7b: Defect-Quality	M7b: Defect-Rate	Defects/KLOC
	C7c: Complexity of Code	M7c(i): McCabe's Cyclomatic Complexity	Cyclomatic Complexity
		M7c(ii): Halstead's Software Science Effort	Number
	C7d: Cumulative Code Output	M7d: Total Cumulative Logical Code	LOC

Step 4. Validate the goals, constructs and metrics.

In section 5.1, we had given details of various validation criteria. In our experiment, validation has been given a foremost priority. There is a general lack of understanding of the meaning of validation of software measures [6] and unfortunately, software measurement research is often suspect because of a lack of rigor and unjustified claims. According to Straub, et al., instrument validation has been inadequately addressed in MIS research [56]:

> *"Because of rapid changes in technology, often the research issues are handled with dispatch. Lack of validated measures in confirmatory research raises the specter that no single finding in the study can be trusted. In many cases this uncertainty will prove to be inaccurate, but, in the absence of measurement validation, it lingers."*

We were cautious in our choice of selecting metrics. Most of the metrics were objective, while the subjective ones were carefully quantified. Wherever appropriate, we selected those metrics which have been used on numerous occasions in the previous studies. Nevertheless, in order to formally validate our goals, constructs and metrics, we contacted seven experts in the field of software measurements, and asked them to fill out a fifteen page validation form (see Appendix D). This form gave details of the metrics, and asked the experts to check

for, amongst other things, content validity. This survey was followed by detailed interviews with these experts wherever a divergence in views was found. Such a method of validating metrics is sometimes also referred as Face Validity [38], in which a group of judges evaluate the measuring technique and suggest their opinions. Face Validity is a subjective process, but we can calculate a validity figure by computing the amount of agreement among judges. However, in our case there was little divergence, and that too mostly amongst subjective metrics. Most of the experts had no objections on the objective metrics since these have been used extensively in past research works. From the previous table, which gave a listing of all the metrics, we list only the objective ones below (Table 4).

Table 4: Objective Metrics

[illegible]	[illegible]	[illegible]
M1 M7a M6c(i)	Programmer-Productivity	LOC/hr
M2a M4a	Percentage Size Estimation Error	%
M2b	Percentage Time Estimation Error	%
M2c	Percentage Productivity Estimation Error	%
M3a M7b M6c(ii)	Defect Rate	Defects/KLOC
M3b M6c(iii)	Defect-Removal Rate	Defects-Removed/hr
M5b	Percentage decrease in Defect Rate	%
M6a(i)-(iv)	Progress Ratio, p	Ratio
M7c(i)	McCabe's Cyclomatic Complexity	Cyclomatic Complexity
M7c(ii)	Halstead's Software Science Effort	Number
M7d	Total Cumulative Logical LOC	LOC

The metrics such as LOC/hr, Defects/KLOC, Defects Removed/hr, McCabe's Cyclomatic Complexity and Halstead's Software Science Effort need no

introduction in software engineering research. A description of their past usage, their drawbacks and why they have been chosen rather than other metrics is given in chapter 4. Furthermore, following Jones' suggestions [33], we have consistently defined every metric, and have then resolutely followed the definitions. Therefore, for example, by defining Productivity to be Programmer-Productivity, we imply that by this metric we measure productivity of a programmer and nothing else. Also, the relation between the constructs and the variables is straightforward, e.g., LOC/hr represents Programmer-Productivity directly. Doubts might arise in case of construct-goal relationship though. For example, does defect quality really reflect quality. As mentioned earlier, we selected those constructs and metrics which have been used in previous studies repeatedly. Hence we are confident that in our case the variables have high effectiveness and the construct and content validitys are strong.

We were very concerned initially about the internal validity. Our original model was to study the constructs/variables against cumulative output alone. However, when we began considering unhypothesized variables, we had to take into account the motivation and incentives. Our search for greater internal validity lead us to develop even 4-variable models (along with several other variables which were held constant or static). After incorporating all these variables, we believe our model has a strong internal validity.

External validity has always been a critical issue for laboratory studies. In our case this problem holds, and even though we had some subjects with extensive experience in industry, great concern should be taken in generalizing our results to be applicable in general industrial software environments.

Our instrument is reliable. In our case there was no need to carry out Test-Retest correlation or Split-Half correlation [38] to check for reliability. This is because the nature of our experiment was, in essence, to take the same measurements week after week at least nine times. Consistency amongst our results during all the nine rounds, by itself, is a proof of high reliability.

The metric Progress Ratio, p, is a derived metric, i.e., it is calculated from other basic metrics. This metric has been used in most of the learning curve studies, and hence was selected by us so that the results could be compared. The metrics for calculating estimation errors are also only a simple calculation based on other basic metrics. Finally, the metric Cumulative Output has been used instead of Time, because of its past usage in learning curve studies. In fact, we prefer it over time because the output produced per unit time is not constant, and hence the experienced gain per unit time varies.

Now we discuss the subjective metrics from Table 3, listed again in Table 5. Note that we have a large number of subjective metrics because it was our goal to have high content validity. These subjective metrics are completely independent of most of the other variables in our work, and have been studied to show that they are static and stay constant. Hence any objections to the use of these metrics should not offset the overall results obtained from our work.

Table 5: Subjective Metrics

	Subject's appraisal of Motivation	Scale of -5 to 0 to +5
M4b	Subject's appraisal of Interest	Scale of -5 to 0 to +5
M4c	Subject's appraisal of Satisfaction	Scale of -5 to 0 to +5
M4d	Subject's appraisal of Usefulness	Scale of -5 to 0 to +5
M6b(i)	Number of Languages Programmed in	Number
M6b(ii)	Total Experience in Programming	Months
M6b(iii)	Educational Excellence Level, Degrees obtained, Majors and Minors	Data mapped to a Quantitative scale
M6b(iv)	Total Full-time and Part-time Job Experience	Months
M6b(v)	Experience with Software Packages	Months/package
M6b(vi)	Experience with Object Oriented Design and Software Engineering Courses	Number of Courses taken or related Tools used
M6b(vii)	Grade obtained in the PSP course	Course G.P.A

Some of these metrics might seem objective, e.g., number of languages programmed in, total job experience, grade obtained, etc. However, they are not objective because we had to design our own scales in order to calculate a Personal Capability Index from them.

An important feature of all these metrics is that they have been quantified, although they could have been left qualitative. Appendix E shows the instrument for measuring motivation (metrics labeled M4). It's check boxes are labeled with both qualitative titles and quantitative figures. A prominent feature in this instrument is the unusually large scale used (-5 to 0 to +5). This was designed to improve the precision and the sensitivity. In particular, the negative range is aimed at facilitating greater interpretability of pessimistic feedback from the subjects.

Metrics labeled M6 are used for two constructs: (i) subjects' experience and education and (ii) subjects' performance in the 10 projects. Data for these metrics was collected at the beginning of the experiment using a six page questionnaire (see Appendix A). This questionnaire was designed and validated (by face validation) by the software engineering group at McGill university. However, it includes a section designed and validated by Humphrey and his colleagues for their own research purposes. The data was then transformed in such a way so that the Personal Capability Index could be calculated. This transformation mapping was designed by the author such that the precision, sensitivity and interpretability were duly considered.

This questionnaire was the only instrument which was used at the beginning of the experiment (and was used only once). Therefore the question of reliability arises here. For this purpose, during the third and fourth projects, all the subjects were interviewed individually. During these 30 minute interviews, they were re-asked all those questions where ambiguous, illogical or doubtful answers were initially given. Due to contradictions in the subjects responses, some of the metrics had to be dropped. For example, the students were asked before the experiment to estimate the total number of LOC they have programmed. At that time, some subjects had no prior experience with measurements, and some gave answers such as 209,000 LOC in six years. At the time of the interview, however, they had measured some of their work and admitted that those values were exaggerated. Hence in order to maintain reliability, such metrics were removed from the study.

Step 5 Define and execute a mechanism for collecting and validating data

Data collection is a core part of any empirical study. We paid particular attention to collecting valid data. We will discuss this in detail in Section 5.4.

Step 6 Analyze the data collected to study the goals

We gave prime importance to statistical conclusion validity. Various statistical techniques were used, and tests for significances were conducted. Complete details of data analysis are given in Section 6.

5.3 Experiment Design

Scientists need viable form with which to express scientific aims. Without significant content, established theory and strong hypotheses, the design of research lacks a strong foundation. And without form and structure adequately conceived and created for the research purpose, little of value can be accomplished [56]. In our work, we paid particular attention to such issues despite the numerous budgetary and time constraints we were facing.

5.3.1 Terminology of Experiment Designs

There are various different experiment designs, and in order to understand them it is necessary to give the definitions of *random sampling, random assignment and matching* [38] [37] [49]. These three methods are used in selecting and assigning subjects from a general population into experimental groups. Readers familiar with experimental design techniques, especially those in social relations and clinical studies would be familiar with these terms and hence may skip Box 4.

Box 4: Random Sampling, Random Assignment & Matching

- **Random Sampling:** It is the procedure for selecting the subjects from a population.
- **Random Assignment or Randomization:** It is the procedure used once a sample of subjects has been selected and is to be expose to a treatment. It is a "fair" way of assigning subjects to two or more groups so that the groups do not differ before the treatment begins. In this case we assume that the groups would be equivalent.
- **Matching:** In some cases the experimenters try to create two equivalent groups of subjects by matching them based on pre-information data collected from them. This strategy is known as Matching, and is the antonym for randomization.

Now we will explain some terminology which we will be using (see Table 6). These notations have several variations in the literature, and hence should be properly understood in order to follow our experimental design.

Table 6: Terminology

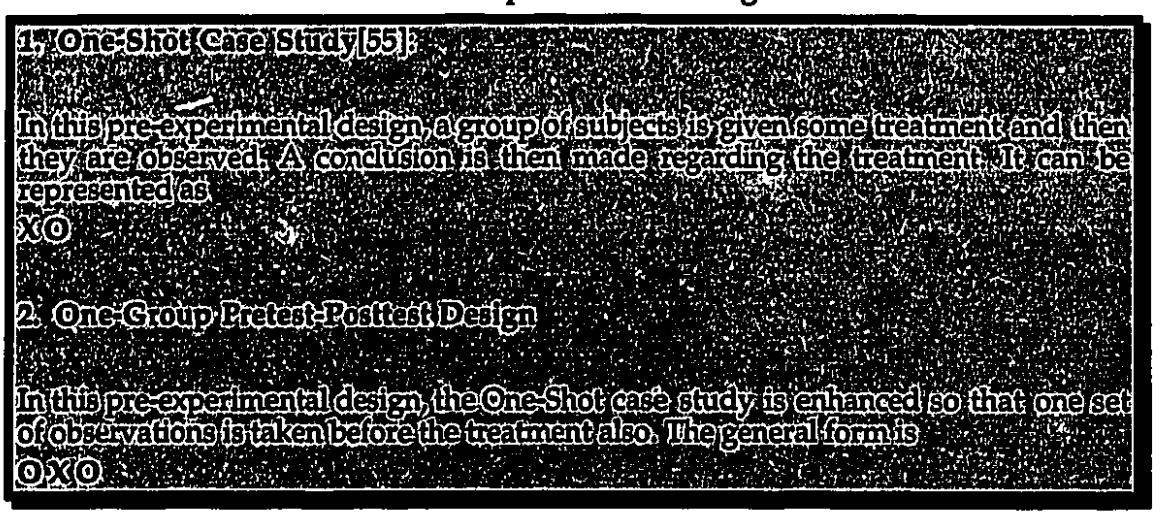
Symbol	Definition
G	Group of subjects, randomly selected from a population. When referring to more than one group, we imply that the two or more groups have been randomly assigned from the subjects who have been randomly selected from the population.
O	Observation (e.g., the act of taking data), a dependent variable, an effect
X	Treatment (e.g., induction of technology), an independent variable, a cause
~X	Treatment X was not given to the group

These notations will now be used in describing the experimental designs. These designs can be divided into three categories: (i) Pre-experimental, termed `pre` since these designs do not satisfy the criteria of being fully and (ii) True Randomized experimental, which are the most valid scientific designs but costly to implement since they need large sample sizes and (iii) Quasi-experimental, a hybrid between the above two, which can be scientifically valid but not fully true experimental.

(i) Pre-Experimental Designs

In these designs (see Box 5), there is a total absence of control, and hence they are of minimal value in establishing causality [49]. The two types in this category are the One-Shot Case Study and the One-Group Pretest-Posttest Design. These are the most basic designs and readers familiar with them may skip Box 5.

Box 5: Pre-Experimental Designs



(ii) True Randomized Experimental Designs

Randomized true experimental designs involve more than one group of randomly assigned subjects. The five types which we selected for this category (see Box 6) are the Control-Group Comparison, Pretest-Posttest Control Group, Solomon, Factorial and Within Subjects/Repeated Measures designs. These designs have often been followed in literature and hence readers familiar with them may skip Box 6.

Box 6: True Randomized Experimental Designs

1. Control-Group Comparison:

This experimental design uses two groups, G1 and G2, where one of the groups is a control group. Hence the treatment X is applied to only one group, while the other control group is observed without any treatment.

G1 X O
G2 ~X O

2. Pretest-Posttest Control Group Design:

This design is an extension of the Control-Group design. It uses a set of observations before the treatment as well as one after the treatment.

G1 O X O
G2 O ~X O

3. Four Group True Experimental (Solomon) Design:

This design is an elegant example of the fundamental logic of experiment design. Besides having a control group, this design also caters to any affects produced on the subject by the initial pre-test observation. However, due to its complexity and the prerequisite of a large sample size, it is seldom used. Its format is as follows:

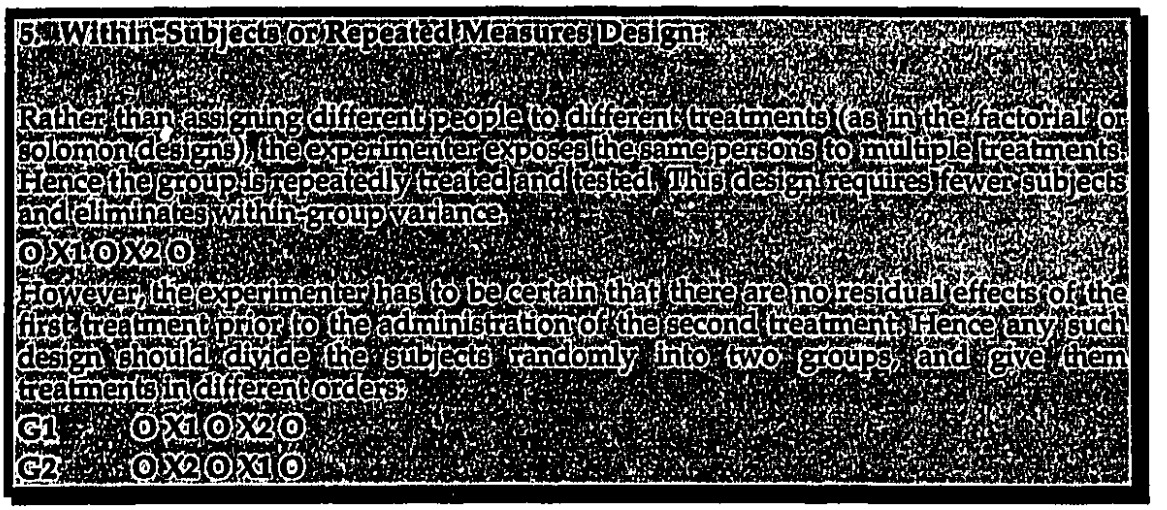
G1 O X O
G2 X O
G3 O ~X O
G4 ~X O

4. Factorial Designs:

This design is used when there are two or more independent variables (or factors), i.e., at least two different treatments, X and Y. Furthermore, each variable can have two or more discrete values. Hence a design with 2 independent variables (e.g., Satisfaction and Motivation) each having two possible values (e.g., Low and High) would be called a 2 x2 factorial design. If X and Y are the variables with values X1, X2 and Y1, Y2 respectively, then this design can be represented by:

G1 X1 Y1 O
G2 X1 Y2 O
G3 X2 Y1 O
G4 X2 Y2 O

Hence the entire design contains every possible combination of the independent variables.



(iii) Quasi-Experimental Designs

There is a vast difference between the pre-experimental design and the randomized (and/or true) experimental designs. Whereas the former cannot be considered scientifically valid, the latter are usually tedious and expensive, often requiring a large sample size. A compromise between the two can be achieved by modifying the pre-experimental design with some forethought and planning, into a scientifically usable quasi-experimental design. An example of this type of is the *time-series* design.

The time-series is an extension of the one-group pretest-posttest design (O1 X O2). Instead of one observation, it uses several observations before and after the treatment. Hence,

O O ... O X O O ... O

One difficulty with these longitudinal (or time series) designs is that learning occurs over time and hence time itself is a variable in a sense. However, this is not a problem in our case because `time' is exactly the variable which we are studying.

Sometimes the treatment occurs only once, but its effects continue afterwards also. Or, sometimes the treatment is injected in the middle of the experiment, and then continuously fed in. In that case we get:

O O ... O X O X O X O X O

Sometimes the time series design is used with a control group. If the two groups are randomly assigned, then it is a full experimental design (instead of being a quasi one).

G1 OO...OXOOO

G2 OO...O~XOOO

Note that this is different than the within subject design, where there are multiple treatments in differing orders.

5.3.2 Threats to Validity

This section describes some of the objections which are raised on the above mentioned designs (see box 7) [38]. These also include the threats to the validity of the experiment such as *maturation, history, instrumentation, mortality, selection and testing.* Other concerns include *the evaluation apprehension*, the *demand characteristics* and the *Hawthorne effect.* Some of the readers might already be familiar with these threats, and hence may skip box 7 without losing any information about our experimental design.

Box 7: Concerns and Validity Threats to the Design

1. Maturation: It is any naturally occurring process within the subjects that can cause a change in their performance, e.g., fatigue, boredom, intellectual maturation, etc.

2. History: This refers to any event that coincides with the treatment and could have a similar effect as the treatment, e.g., research equipment failure, experimenter being replaced, other political and economic events that affect the subjects, etc.

3. Instrumentation: It is any changes in the measurement procedures, such as the adaptation of some better data collection method during the middle of the experiment.

4. Mortality: This refers to any dropout of subjects from the study during the middle of the experiment. This may cause discrepancies in the pre-test and post-test observations.

5. Selection: This refers to any strategy of assignment or selection of subjects for treatment, that is not random. A matched strategy, no matter how accurate, would have a selection threat.

6. **Testing:** Sometimes the pre-test observation sensitizes the subjects, for example, by making them believe that they should relax or slow down. e.g., a subject finding out about his/her extremely high productivity in first observation might slow down during the next observations. Hence this pretesting alone (without any treatment) could affect the posttesting observation. This phenomenon is referred to as Testing validity.

7. **Evaluation Apprehension:** Sometimes the subjects pay particular attention to the experimenter's behavior and the experimental setting to discover the purpose of the study and try to respond "correctly" by giving socially desirable responses which may please the experimenter. This experimental artifact is known as evaluation apprehension, and causes a bias in the results in favor of what the experimenter is attempting to achieve.

8. **Demand Characteristics:** Sometimes experimenters intentionally or unintentionally give the subjects hints about how they are supposed to behave. Sometimes the experimenters do not know what hints their subjects might perceive. Usually the subjects respond favorably to these demand characteristics, hence producing a bias in the results.

9. **Hawthorne Effect** [35]: It is the tendency for subjects to show increased productivity under any new situation in which their performance is being monitored.

5.3.3 Our Choice of Design

In this section we describe the rationale for our choice of a particular research design, why we rejected various other design alternatives, and what objections may be raised on our design.

As discussed above, the pre-experimental designs are not sufficient for a scientific study. Therefore we rejected them. One important requirement of a design is to have a control group, so that one group receives the treatment while the other one does not. However, we are in an extremely unique position because one of our main research goals is to study the first-order learning for which the only treatment needed by the subjects is *time*. Hence, even if we have a control group, which receives no treatment from us, it would have differences in the pre and post tests just because of the elapsed time and the subjects would have learned more due to autonomous learning. Because of this unusual variable which we are studying, it is difficult to apply a control group, and hence a true experimental design.

However, we could have used a control group to monitor the second-order treatment, which is the training given to the subjects. One group could have been given the training while the control group could have carried out without training. Such a design would have been a true experimental one, and we would have preferred to use it, but could not because we had the limited choice of those subjects who took the PSP course. Thus we could not have a control group. Furthermore, such true experimental designs would have needed a large sample of subjects, at least enough for two large groups. The closest we could get to using a true experimental design was to use a within-subject design, because in it the control group is needed only to vary the order of the treatments, instead of varying the types of treatments. Since our design resembled the true repeated-measure within-subject design, most of the features of any true experimental design were incorporated in our work. These include:

- pre-test and post-test observations
- repeated within-subject measures over long periods
- random selection of subjects from a population of graduate computer science students

In fact, our design at least is equal to a Time-Series Quasi-experimental design, and at most is equal to a Repeated-Measure Within-Subject True experimental design. As mentioned before, a repeated-measure design has the pattern:

O X1 O X2 O

According to Sheil [59]:

> *"The high degree of variability among programmers of similar background makes simple experimental designs (in which different participants are used for each condition) prone to negative conclusions, as slight systematic differences between conditions tend to be washed out by large within-condition variation. One of the standard techniques for controlling this is the use of `repeated-measures' designs, in which each participant is observed in more than one condition."*

However, the experimenter has to use two groups to make sure that the effects of the first treatment have vanished before the start of the second treatment. Hence we get:

G1 O X1 O X2 O

G2 O X2 O X1 O

However, in our design the two second-order treatments which we chose to study are independent or mutually exclusive. These two treatments are:

- X1 = Size Estimation Training
- X2 = Code Reviews for Defect Removal Training

Obviously, the above two treatments do not effect each other in any way. Hence we do not need two separate groups, and our design can be reduced to:

O X1 O X2 O

Another reason why we did not make two groups out of our sample was that we knew about the background (experience and performance) of all the students in detail. Hence it would not be justified if we had made two groups (which would not have been equivalent or matched) and then pretended that they were equivalent on the grounds that we had used random assignment. Although scientifically such a random assignment cannot be questioned, but we would not have been satisfied with such a design knowing that the two groups are not matched.

A positive aspect of our design is that there is not one but three observations before and after each treatment, hence providing more reliability. Hence in our case:

O O O X1 O O O X2 O O O

where each observation is taken at a one week interval. This can he written as:

O1 O2 O3 X1 O4 O5 O6 X2 O7 O8 O9

The reason why there are three observations in between treatments is that one of the treatments which we are studying is `time.' In other words, since we are studying the first-order learning, we are taking measurements with the passage of time and hence time itself is a treatment, or an independent variable. Of course, simultaneously we give second-order treatments also, which are X1 (size estimation techniques) and X2 (code review procedure). Since these two are

independent or mutually exclusive (i.e., post-treatment effects of one do not influence the other), we can break up our above design model into:

O O O X1 O O O O O O

O O O O O O X2 O O O

And these are actually two separate Time-Series Quasi-experimental designs. As mentioned before, if used with caution, Quasi-experimental designs can be used in valid scientific studies. In our case, we not only have such a design but also have several added features which make it resemble a true within-subject design.

We will now discuss step by step the threats to validity (discussed in section 5.3.2) and see how seriously they effect our experiment, if at all (see Box 8).

Box 8: Concerns and Validity Threats, from our design's perspective

Maturation: Of course, it is not possible to stop maturation from occurring, regardless of how good the design may be. In our case also, the subjects might be getting increasingly enthusiastic, bored or fatigued. With weekly training lectures, they might also be getting more and more intellectually mature. However, the time-series designs do allow us to assess the plausibility of maturation as a rival explanation [55]. If maturation is the cause, it should appear as a trend before the treatment as well as afterwards.

1. History: Fortunately, we were lucky that none of our subjects had to get replaced. In addition, we also questioned our subjects and found that no one was taking some other course where C++ (the language we used) or similar material was being taught to them. Hence we can confidently rule out the threat of History.

2. Instrumentation: At the beginning of the experiment, we had carefully worded out all our strategies and explicitly defined the measurement instrument. As mentioned earlier, these details were published at that time in a position paper [53]. We then adhered to our instrument and were consistent throughout. There were no changes made in the data collection techniques, the training schedules, etc. Hence the threat due to instrumentation can be ruled out.

3. Mortality: Our subjects have been our prime focus of attention throughout the experiment. We paid careful attention to each individual subject. No subject dropped out at any time and there were no missing projects. Hence the threat due to Mortality does not apply in our case.

4. Selection: This threat occurs when some form of Matching is done between groups. In our case no such matching took place. Our subject sample consisted of graduate computer science students who had enrolled for the PSP course at McGill University. Obviously we wanted our population to be only the graduate

computer science students, since they come the closest to real-world software industry programmers (as compared to undergraduate students or to the students from other academic majors). Since all the graduate students have equal rights in taking this course, and the decision to take the course was taken by the participants, this process is considered random.

5. **Testing:** This validity tests for any effects which the pretests might have on the subject. Solomon design uses a control group which does not have a pretest at all. One advantage of time-series designs is that they rule out the threat from Testing because there is such a large number of observations. For example, if one suspects that the difference between any two observations O4 and O5 resulted not from treatment but from the sensitizing effects of the pretest (O4), one can look at the preceding and succeeding intervals to see whether the repeated testing produced similar differences along the entire series. If there are no differences at any other points, it would not be plausible that the testing at O4 alone would have created an effect at O5 [38]. Hence our design, which is used to measure to effects of X1 (Size Estimation Training) and X2 (Code Reviews), is quite safe from the threat from Testing. We are, however, still skeptical about this threat when it comes to measuring the first order learning effects. This is because in our experiment *time* by itself is a treatment. Hence, in between every observation (e.g., O4 to O5), there is a treatment of *time*. But since we have nine observations, the subjects soon get used to these tests and hence the effect of these tests on subjects is only negligible.

6. **Evaluation Apprehension:** This attribute is an experimental artifact [38], which remains to plague the experiments regardless of how good the design is. The only way it can be avoided is to completely hide the fact from the subjects that they are being used in an experiment. However, lying to subjects is abusing the subjects, which is considered unethical in the world of social relations. Definitely our subjects must have faced evaluation apprehension. Nevertheless, on our part, we tried our best to minimize this apprehension. For example, although the subjects were told that their data would be used for some form of research requirements, they were never told the research goals or objectives until after the experiment. Another important aspect is the confidentiality. The subjects were reminded repeatedly that their data would be revealed to no one, so that the may not be apprehensive of any forms of evaluation, e.g., by their employers. Furthermore, there were only a few subjective metrics where the subjects had to 'decide' and produce an answer. Nearly all the data was objective, which the subjects had to measure based on how they had done in the project, and did not require their personal discretion. Finally, since most of our work does not deal with absolute values but with relative performance (i.e., how much the subjects learned with respect to the first project), as long as the subjects show evaluation apprehension consistently throughout the experiment, there is no chance of a validity threat.

7. **Demand Characteristics:** No intentional cues or hints were given to the subjects. However, since one purpose of the PSP course was to teach the subjects how to improve, naturally several improvement directed techniques were taught, which unintentionally act as hints since the subjects knew we wanted them to improve. Unfortunately due to the very nature of the experiment, we had to teach the subjects improvement related issues, and hence it was very difficult for us to have avoided these hints.

8. **Hawthorne Effect:** This effect is probably of negligible significance because of two reasons. First, it is usually most dominating when there are only one or two observations. For a period of 10 weeks with continuous weekly measurements, subjects tendency to be more productive settles back to normal. Second, as mentioned before, most of our data values are used relative to each other, and hence as long as this effect continues through every observation, it should balance out.

To summarize the above discussion, nearly all the major threats to validity have been removed or reduced. Although there are some threats which still exist, we can be quite confident that our design is strong and valid. Now we explain the details of the implementation of our design.

5.3.4 Implementation of our Design

The experiment was done on the students who had enrolled in a course entitled 'Personal Software Process.' This course was designed by Humphrey of the SEI and taught at McGill University by Madhavji. Below we now the background of the subjects, the training lectures and projects, and the environment.

SUBJECTS

Twelve full time graduate students had enrolled in the course. With the exception of one subject, all had either a bachelors degree in computer science or were enrolled in the masters degree in computer science. That one exception, though not a computer science major, had two years of programming experience of over 20,000 LOC in five different languages and had taken prior computer science courses. Hence, none of the twelve subjects was new to the field of computer science. Since all the students had enrolled in the course based on their personal choices, the experimenters had no influence in their selection process. Hence the subjects represent a random selection from the population of graduate computer science students. Exactly half of the students had prior full-time as well as part-time job experience, while the other half had no job experience at all.

None of the subjects had any experience in C++, the programming language used in the course. Hence, all the subjects were at the same starting point in the learning curve. However, this was not their only common denominator. They all were similar in the sense that they all had extensive experience with C and with other programming languages. The statistics are given in Table 7:

Table 7: Statistics on the Subjects

Number of Males	10
Number of Females	2
Mean experience with C++ Language	0 months
Median experience with C++ Language	0 months
Mean experience with C Language	28 months
Median experience with C Language	24 months
Mean total programming experience	6.5 years
Median total programming experience	6.0 years
Number with full-time and part-time job experience	6
Number with no job experience	6

TRAINING LECTURES AND PROJECTS

Every week two lectures, each of duration one and a half hours, were given to the subjects in which they were taught ways to improve their personal process. After each set of lectures, the students were assigned a programming project, which utilized the techniques taught to them, so that the students actually implement those methods, and hence learn them. There were a total of ten projects, one per week, of which nine were programming ones while one was only on analyzing the data. Table 8 some statistics collected on the average size of each project and the average time spent on them:

Table 8: Statistics on Projects

Average Total Size of a program (including Reused Code)	201 LOC
Average Time spent on a project	4 hours 27 min.
Average Reused Code per program	81 LOC
Average New and Changed Code per program	120 LOC
Average Defects recorded per program	10.3

The students were told repeatedly and explicitly that although a minimum criteria is to complete the project, they would not be graded on the project, but on how complete their data is. Similarly, they were reminded that they would not be graded on how good their productivity, defect quality, etc. are, but how well they record the metrics related to them, e.g., LOC, number of defects, etc. (See next section, 5.4, for details on data collection)

ENVIRONMENT

In experiments requiring only one project and a couple of observations, it is easy to make it mandatory for all the subjects to use the same environment, e.g., the same computer laboratory. However, for a ten week long experiment, some flexibility has to be given to the subjects. We provided the subjects with latest computers, compilers, environmental conditions, etc., but still a couple of subjects preferred to work at home on their Personal Computers. The remaining ten of the twelve students used the same machine and the same compiler, though three of them used a different computer lab (a different room) to telnet to that machine. The statistics on environment are given in Table 9 below:

Table 9: Statistics on Environment

Subjects using standard machine and (Gnu) compiler	10
Subjects using personal computers and Borland compiler	2
Subjects using standard machine from the standard laboratory	7
Subjects using standard machine from different laboratories	3

The difference in the laboratories has negligible effects on our results, since each laboratory provides the students with ample work area, adequate desks, proper air conditioning, etc. However, the two subjects who used computers from home, with different compilers were certainly using a different environment.

Unfortunately, there is no reasonable way for us to account for these variations. Curtis and Vosburgh, et al. [21], found that the programming environment (characterized by the development computer) explained for less than 24% of the variation found in the productivity of the programmers. In our case, 10 of the 12 subjects have similar environments, and hence any variation caused due to the other two programmers would not be very significant (roughly 2/10 of 24%).

5.4 Data Collection

Basili [10] has described a goal-directed method for data collection and one of the most important aspects of it is to validate the data. Most of the data collection forms which we used in this experiment were initially validated and designed by Humphrey and his colleagues, and later by the software engineering group at McGill University, which made some changes and enhancements during the validation process. We found that Humphrey's data collection techniques were in complete harmony with Basili's goal-directed method. However, in addition, from our research perspective, we had to develop some extra measurement instruments as well.

There are various different data collection techniques, of which we used the following (see Appendix F for sample forms):

- Logs
- Forms
- Templates
- Spreadsheets
- Databases
- Summary Reports
- Automatic LOC Counters
- Automatic Complexity Analyzers

It is beneficial to include the data-suppliers in the data-collection design process and to interview them [10]. Our validation process included weekly data reviews, consistency checks, repeated instructions to the data-suppliers and detailed interviews. The subjects were also asked to analyze their data themselves on spreadsheets so that they can gain insight on how the data is being used. For this purpose, they were asked to use G/Q/M to identify some of their own goals and

then to analyze those goals for which the data was available. At no point did we mention our own goals to them. Following are the steps we took for our data collection process:

1. We devised an initial questionnaire for subjects background. It consisted of a 6 page form, of which two pages were prepared by Humphrey. The other four pages were prepared and validated by the software engineering group at McGill University.

2. Every week the subjects were given a project, along with detailed instructions on how to complete it. These projects were such, that they helped the subjects in following the PSP. e.g., developing LOC counters for measuring their program sizes, developing software packages to help them in size estimation, etc. In order to remove ambiguities, the subjects were given a description of the requirements in the class lecture. Further clarifications, if needed, were given by electronic mail to all the subjects, who checked the mail regularly. Individual help was also offered to any subject, if desired, based on an open door policy. Such help was frequently sought. Peer help was allowed during the projects also.

3. The subjects were required to collect detailed and accurate data. They were told explicitly that the main criteria for grading them was the quality of the data they collected. This data collection required the filing up of several logs, forms, templates, reports, etc., and can be divided into three stages:

- Planning Stage: In this stage the subjects make their estimates of the program size, time, productivity, number of defects, etc.

- Concurrent Data Collection: This is the stage when the subjects continuously gather data as they design, code and test their program. An important measurement in this case is that of the time spent on each phase and activity. All the students used either a stopwatch or the clock displayed on the computer screen. During this stage the subjects also noted down the details of the defects.

- Post-Mortem: Here the subjects completed the summary reports and carried out the other immediate analysis of the data.

4. Subjects data was then checked for consistency and validity. The nature of the data is such that if not collected properly at one place, its effects show up at other places also. For example, a wrong value of time in the time-record

log would effect the productivity calculations as well as the total time figures in the project plan summary. Hence consistency checks were made to make sure that the data in various forms matched with each other. Logical validity tests were also carried out to check that the values of some common variable were rational, i.e., a value of productivity equal to 200 LOC/hr, though not impossible, would be highly questionable. Similarly, the students counts of the LOC were rechecked in several random cases by using other automatic counters. Data values given by subjects on the hard copies were compared with those provided by them in the databases, and those in turn were compared with their data analysis reports. Even the basic addition and division tasks carried out by the students were randomly sampled and checked. In brief, all sorts of checks were made to make sure that no student lagged behind in data quality.

5. A weekly feed-back report was then given to each subject. The subjects were informed of any errors they have been making. They were given comments on how well they are following the coding standards. This feedback included separate sections for the program and for the data. Whenever necessary, these reports were followed by verbal advice to some particular subjects. In some cases, especially during the beginning of the experiment, the subjects were asked to correct those calculations where the data had been wrongly interpreted. Of course the fundamental data entries were never changed once they were recorded. It was in these feed-back reports that the subjects were assigned grades as well. Overall, these feedback reports made sure that the data is not deficient in correctness, consistency or completeness.

6. Between the third and fourth project, all the subjects were interviewed. Basili [10] stresses on the importance of interviews and says that the lag between the filling of the forms and the interviews should be kept minimum. Each interview was at least 30 minutes long and detailed minutes were recorded. Here the subjects were asked details of any assumptions they have been making in their data collection process. Since people differ in interpreting the directions given to them, it was tried to understand what approach each subject was pursuing in comprehending directions given by us. We found that these interviews added to our knowledge and were extremely useful in removing the threats to validity. In addition, they were useful to the subjects also since each subject was given extra time to ask any questions about the data collection process, and most of them ended up clearing several ambiguities.

7. During the second half of the experiments (projects 6 through 10), the subjects had enough data points from the first half (projects 1 through 5), and hence were asked to carry out analysis of the data. Usually this was done by appending an analysis problem with the project. However, project 8 was completely devoted to the analysis, and the subjects were asked to use G/Q/M to identify their own goals and then to analyze them using their data. The primary objective of such analysis was for the subjects to study their improvement process. However, a secondary objective was for the students to better understand their data. We believe that this led to even more reliable data from the students.

From the above descriptions, we note that under the constraints of time and budget, we have attempted to ensure that our data is as consistent, complete and correct as it can get.

6.0 Data Evaluation and Analysis

This section presents a detailed analysis of the collected data. Once again, we stress the validity of this data. For any significant analysis, the statistical validity has to be carefully checked, and that is why we put in a great deal of effort in carrying out various statistical tests.

Our experiment generated a large volume of data. For example, one of the data bases used by us had over 1000 rows and 40 columns and consumed over half a megabyte of memory, and there were over 5 such related data bases (see Appendix K for a sample section from spreadsheet). Sophisticated spreadsheet and statistical packages had to be used for analyzing this data, and advanced charting packages were needed to plot the graphs. The primary structure of our data was as follows:

- Each of the 12 students (X_1, X_2, ... X_{12}) contributed to ONE project data point (Y)
- There were 9 (Y_1, Y_2, ... Y_9) such projects (hence a total of 9 x 12 = 108 data points)
- Such sets of 108 data points were taken for about 75 different variables such as the defect rate, productivity, etc. (see Appendix K for a listing of some of these variables)

For such huge data, there are various assumptions which we need for carrying out the statistical tests. It is difficult to prove all these assumptions to be correct, however, most of them can be explained satisfactorily. Some of the main assumptions are listed and justified in Box 9 [39]. These are the *existence, independence* and *normality* assumptions. Readers already familiar with them or interested in just the data results may skip Box 9.

Box 9: Assumptions in Data Analysis

- **Existence:** The probability distribution of the values of the random variable Y, have finite mean and variance. In our case, this assumption is obviously true because all our variables have finite values.
- **Independence:** The Y observations are statistically independent of each other. Again, in our case, the data for one project is independent of the data from other projects, so this assumption is also satisfied.
- **Normality:** This is perhaps the most important assumption. It requires that, for any fixed combination of $X_1, X_2, \ldots, X_{12}$, the value of the variable Y is normally distributed. In other words, for example, the values of productivitys obtained from 12 students for some particular project should be normally distributed; i.e., some of the students might have very high productivitys, some might have very low ones, but most of them would have productivitys somewhere close to the mean value, hence forming a Gaussian curve.

We recognize that this assumption cannot be proven fully since our sample size is only 12. Although experiments with dichotomous variables in social relations often consider only 5 subjects per factor as a sufficient number, in our case even 12 subjects are not ample to prove that their data values are normally distributed. This is because the tests for normality such as the Chi-squared test require at least 20 data points to draw any inferences about the degree of normality [30]. We can plot the data and visually see if it looks something close to normal or not, but that is not a valid statistical test. Nevertheless, the very nature of our variables is such that it is safe to assume that they are normally distributed. For example, it is just not rational to think that there would be more students with high productivity values than the mean value, and only a few students with very low productivity values. Hence we would safely presume that our assumption is correct and carry out our statistical tests. Note that these statistical tests are usually carried out in most of the research works without explicitly stating the nature of the assumption. We have at least evaluated this assumption drawn a conclusion that we can safely use it, although we cannot prove this conclusion given our sample size.

We will now consider all our goals (G1 through G7) separately. These goals (from section 5.2) are relisted here for convenience.

- G1: Identify the Progress in Productivity (4.1.1)
- G2: Identify the Progress in Personal Skills (4.1.2)
- G3: Identify the Progress in Product-Quality (4.1.3)
- G4: Analyze the second-order learning through management motivation (4.2.1)
- G5: Analyze the second-order learning through training and technology (4.2.2)
- G6: Analyze the first-order learning (4.3)
- G7: Analyze the 4-Variable Model (4.1.3 and 4.4)

Table 10 lists the sequencing of the sections along with the goals studied in them.

Table 10: Sequencing of Sections in Chapter 6

The Six Learning Curves	G1 G2 G3	6.1
Complexity Analysis	Leads to G7	6.2
4-Variable Model	G7	6.3
Second Order Learning: Engineering Technology	G5	6.4
Second Order Learning: Management Motivation	G4	6.6
First Order Learning	G6	6.5

6.1 The Six Learning curves

This section deals with the first three goals, G1, G2 and G3. Table 11 gives the six learning curves studied, corresponding to each goal.

Table 11: Goals 1, 2 and 3

G1	Progress in Productivity	Productivity LC
G2	Progress in Personal Skills	Size Estimation Ability LC
		Time Estimation Ability LC
		Productivity Estimation Ability LC
G3	Progress in Product Quality	Defect Rate LC
		Defect-Removal Rate LC

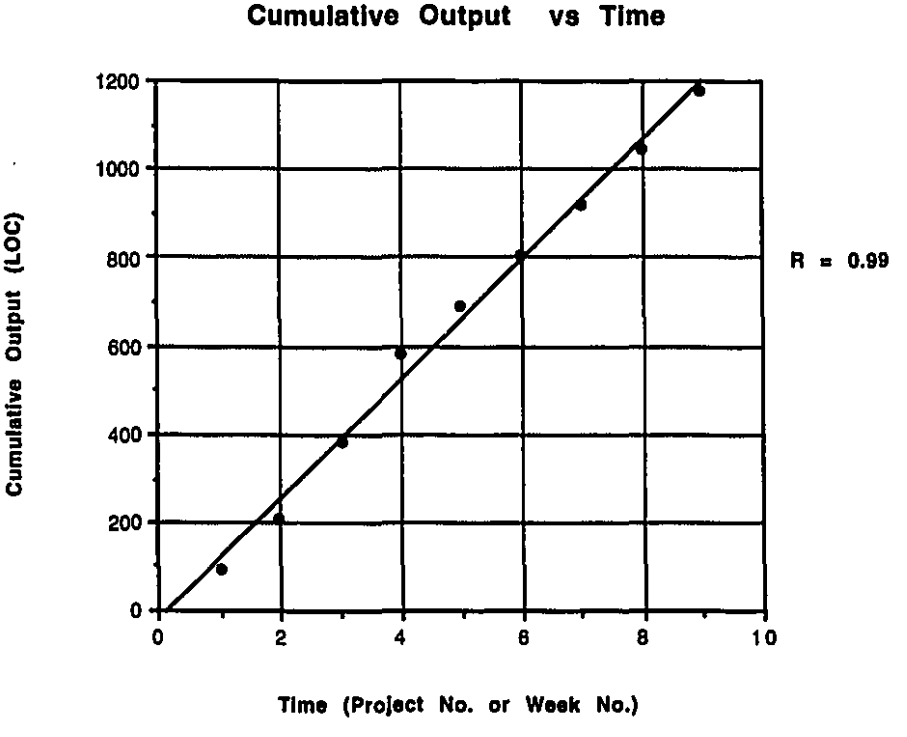
We decided to fit both the linear and the quadratic models, to investigate whether there was any significant advantage of using a quadratic one over the simple linear one. However, in the past literature, the progress ratio, p, calculations have been done on log-linear curves. Moreover, the log-linear equations are intuitively

easier to comprehend. Hence all the following three models were studied:

1. Linear
2. Quadratic
3. Log-Linear

As described earlier, for X-axis, we used Cumulative Output instead of Time. The Cumulative Output at some project N is the sum of all the lines of code from projects 1 through N. Our results would not have been much different even if we had used Time as the variable, since there is a correlation of greater than 0.99 between the two variables (at 0.001 significance level). This significance level represents the probability of the points lying on the straight line by chance only. Figure 15 shows a plot of Cumulative Output against Time.

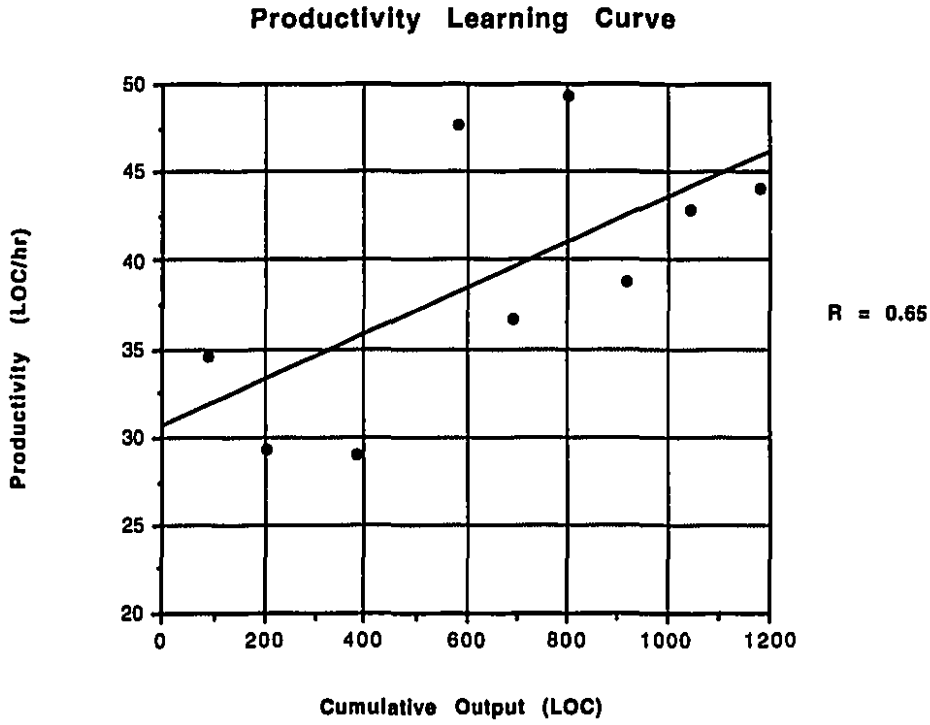
Figure 15: Graph of Cum. Out. vs. Time



Figures 16 to 21 show the linear models of the six learning curves. The log-linear and the linear equations of the corresponding curves are also listed below them. The correlation coefficient, R [55], the significance level, and the progress ratio, are also listed along with each linear graph. As done in past literature, the progress ratio has been calculated from the equation of the log-linear graph only since there

is no theoretical method of calculating it from the linear graph. The equation for the calculation of this progress ratio, p, is given in chapter 2.

Figure 16: Graph of Productivity vs. Cum. Out.



$$Productivity = 15.33\ Cumulative\ Output^{0.15}$$

$$Productivity = 30.56 + 0.0130\ Cumulative\ Output$$

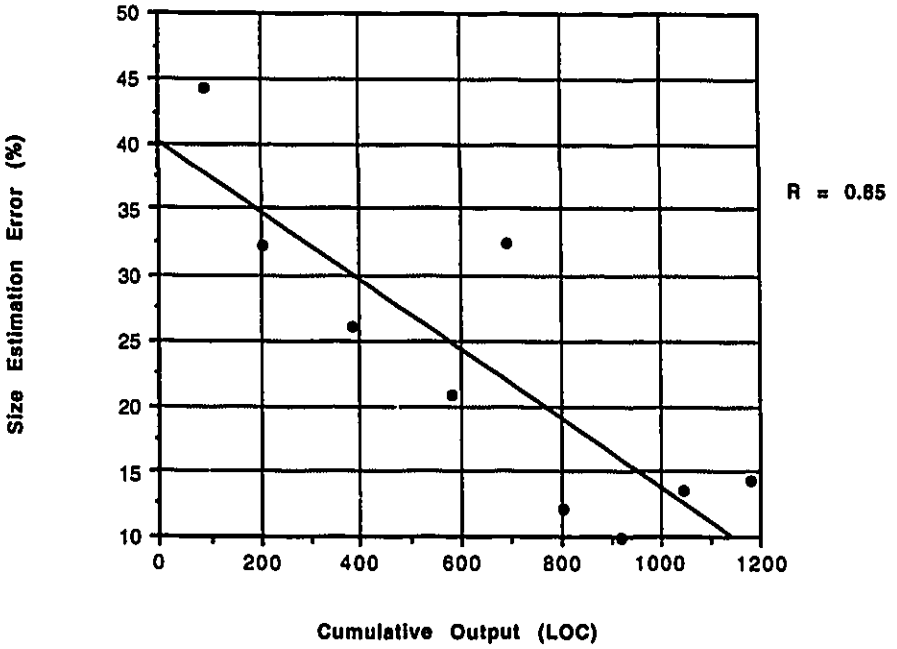
The top equation is the log-linear one, while the bottom one is the linear equation.

Linear Graph: R = 0.65 (0.05 level)

Log-Linear Graph: R = 0.63 (0.05 level)

Progress Ratio, p = 11%

Figure 17: Graph of Size Estimation Error vs. Cum. Out.



$$Size\ Estimation\ Error = \frac{496.94}{Cumulative\ Output^{0.51}}$$

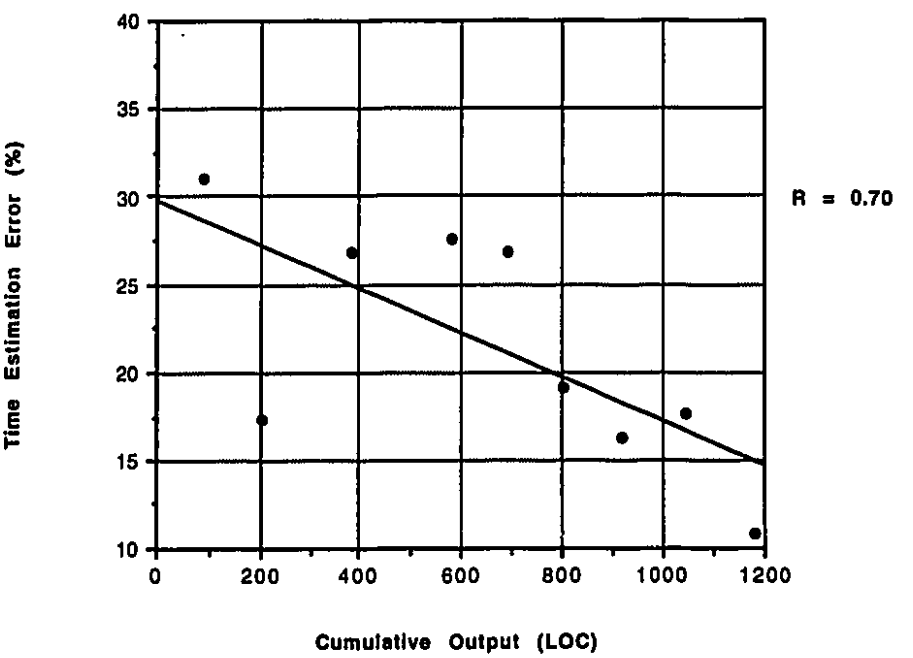
$$Size\ Estimation\ Error = 40.278 - 0.0266\ Cumulative\ Output$$

Linear Graph: R = 0.85 (0.0025 level)

Log-Linear Graph: R = 0.83 (0.0025 level)

Progress Ratio, p = 30%

Figure 18: Graph of Time Estimation Error vs. Cum. Out.



$$Time\ Estimation\ Error = \frac{87.98}{Cumulative\ Output^{0.23}}$$

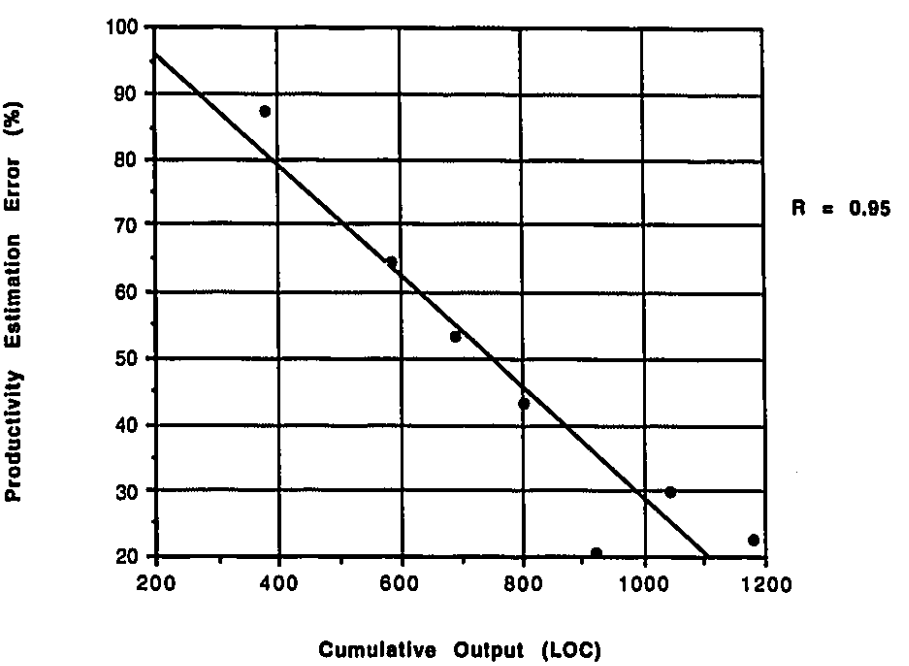
$$Time\ Estimation\ Error = 29.75 - 0.0126\ Cumulative\ Output$$

Linear Graph: R = 0.70 (0.025 level)

Log-Linear Graph: R = 0.58 (0.05 level)

Progress Ratio, p = 15%

Figure 19: Graph of Productivity Estimation Error vs. Cum. Out.



$$\text{Pr}oductivity\ Estimation\ Error = \frac{226360.16}{Cumulative\ Output^{1.30}}$$

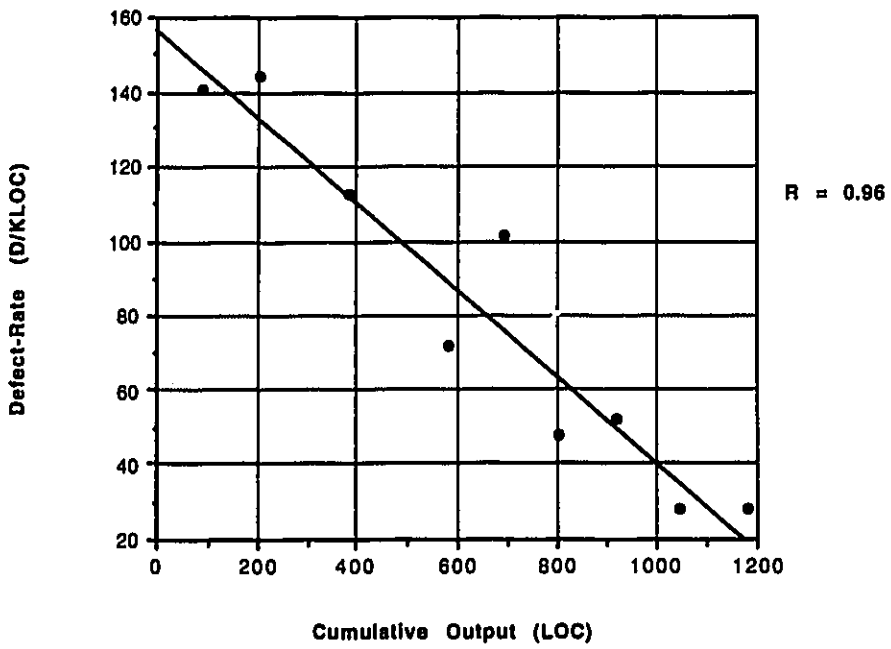
$$\text{Pr}oductivity\ Estimation\ Error = 112.92 - 0.0839\ Cumulative\ Output$$

Linear Graph: R = 0.95 (0.001 level)

Log-Linear Graph: R = 0.92 (0.001 level)

Progress Ratio, p = 60%

Figure 20: Graph of Def-Rate vs. Cum. Out.



$$Defect\ Rate = \frac{3650.90}{Cumulative\ Output^{0.64}}$$

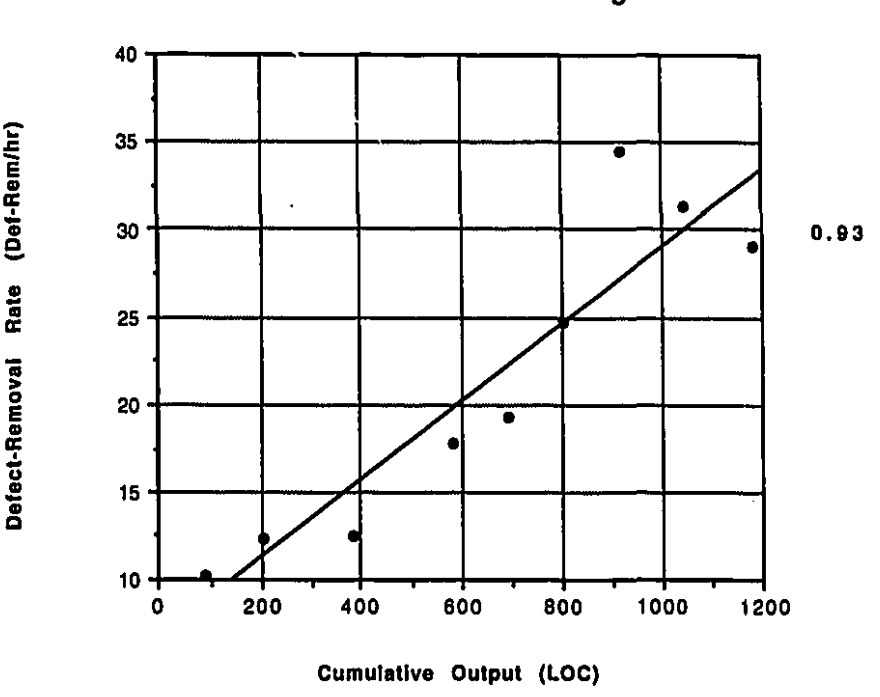
$$Defect\ Rate = 156.74 - 0.1162\ Cumulative\ Output$$

Linear Graph: R = 0.96 (0.000025 level)

Log-Linear Graph: R = 0.85 (0.0025 level)

Progress Ratio, p = 36%

Figure 21: Graph of Def-Rem. Rate vs. Cum. Out.



$$Defect\ Removal\ Rate = 0.9679\ Cumulative\ Output^{0.48}$$

$$Defect\ Removal\ Rate = 6.75 + 0.0214\ Cumulative\ Output$$

Linear Graph: R = 0.93 (0.00025 level)

Log-Linear Graph: R = 0.91 (0.00025 level)

Progress Ratio, p = 40%

The above results are summarized below in Table 12, which shows the Pearsons correlation coefficient, R, and the significance level for each of the six linear learning curves. Table 13 shows the corresponding values for the log-linear curves.

Table 12: The correlations and significance levels of the linear learning curves

Learning Curve, LC	Correlation, R	Significance Level
Productivity LC	0.65	0.05
Size Estimation Ability LC	0.85	0.0025
Time Estimation Ability LC	0.70	0.025
Productivity Estimation Ability LC	0.95	0.001
Defect Rate LC	0.96	0.000025
Defect-Removal Rate LC	0.93	0.00025

The best quadratic fit was also found and the equation of the quadratic curve and the new value of R were determined. An analysis of variance was done with ANOVA tables and F tests. It was found that in none of the six cases, the new value was significantly better than (at the 0.20 level) the linear curve fit. Hence, for simplicity, only the linear model can be used.

Table 12 shows that the values of R for the Productivity Estimation Ability LC, the Defect Removal Rate LC, and the Defect Rate learning curve are all very high (over 0.9). Since these results have high significances also (of at least 0.000025 level), we can safely conclude that there is a strong linear relationship between these variables and the cumulative output. Only the productivity learning curve has a low value of R (0.65, at 0.05 significance level). This is due to the fact that the productivity does not depend on cumulative output alone, but also on the Defect rate as well as the complexity. Hence in the next section (6.2), we discuss the complexity and then in section 6.3, we develop a larger model which incorporates these other variables as well.

For plotting the log-normal graphs, the logarithmic values of the Y and X axis were plotted against each other on linear scales. In our case, the log-normal graphs for the six learning curves resembled the linear graphs. Hence these graphs have not been printed, but their equations have been listed under each of the figures 16 through 21. For convenience, the values of the correlation coefficient, R, and the significance levels for the log-linear models are listed below in Table 13. The learning rate, which is calculated only from the log-linear model, is also given in Table 13.

Table 13: The correlations, significance levels and the learning rates of the log-linear learning curves

Learning Curve, LC	Correlation, R	Significance	Learning Rate
Productivity LC	0.63	0.05	11%
Size Estimation Ability LC	0.83	0.0025	30%
Time Estimation Ability LC	0.58	0.05	15%
Productivity Estimation LC	0.92	0.001	60%
Defect Rate LC	0.85	0.0025	36%
Defect-Removal Rate LC	0.91	0.00025	40%

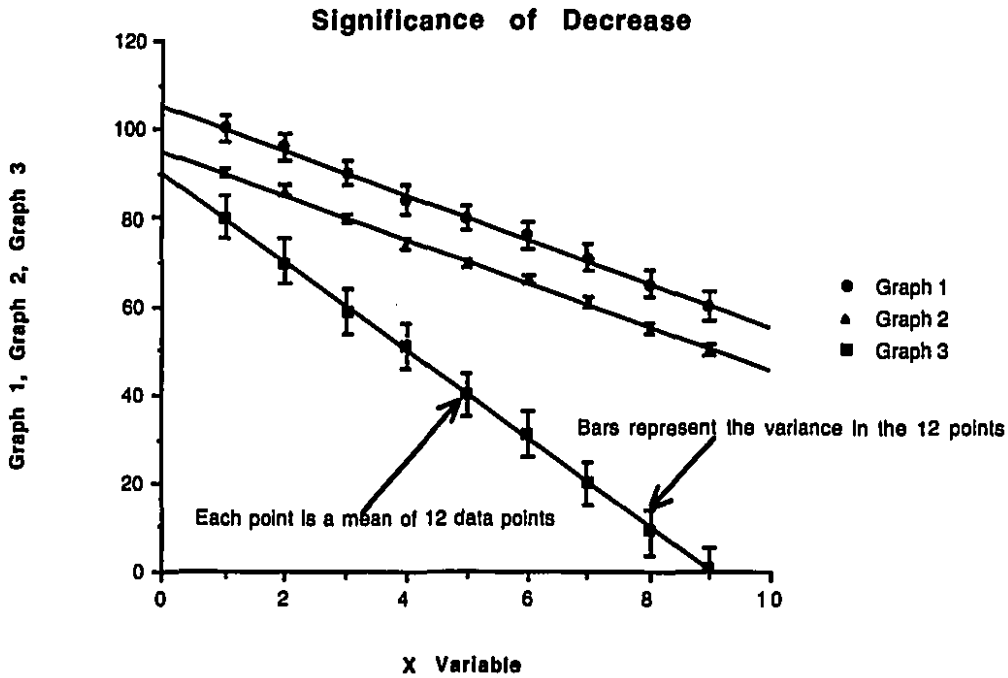
Table 12 and 13 show that the values of R in log-linear curves are slightly less than those of their corresponding linear curves. This difference is most visible in the Time Estimation Learning curve, where the drop in R is quite substantial. The lowest learning rate obtained (11% in the productivity learning curve) is not accurate since we have mentioned before also that the productivity depends on other variables as well. The other learning rates range from 15% to 60%, with a mean of 36.2% and a median of 36%.

The sketches of the learning curves visually, and their progress rates quantitatively, give us an intuitive idea of how fast the learning was and how quickly the cost per unit output decreased. However, one can always argue that perhaps this increase in performance (or decrease in cost per unit) in the learning curves is occurring by chance only. Hence we decided to carry out tests for comparing the differences in the data points.

Recall that each data point is a mean of 12 other points, and hence we are in effect comparing several means to see if they have decreased significantly or not, relative to the variation within each mean. For this purpose, a t-test can be used to compare two means, while an F-test can be used to compare multiple means. Figure 22 gives an example of three fictitious sets of data points (representing learning curves), all with high correlations and significances. Each data point is a mean of 12 data points. The length of the error bars shown on each data point represent the variance of these 12 points. The significance of the differences in means depends on two things: (a) the slope of the line and (b) the length of the error bars.

Graph 1 and Graph 2 have approximately the same decrease (slope), and hence are parallel. However, the variance of the 12 data points (shown by the length of the bars) is much greater in Graph 1 than in Graph 2. Hence the difference in the points may not be as significant in Graph 1 as in Graph 2. Note that the variance in the points in Graph 3 is the highest of all the three graphs, implying that the significance should be less. However, since the decrease in the values of these points (or the slope of the line) is very high (steep), the difference in the points in Graph 3 may be more significant than the other two graphs.

Figure 22: Sample graph showing significance of decrease



The results of these F-tests are given in Table 14 (i.e., the significance levels of the differences between the points), on five of the six learning curves. These significances represent the probabilities that the increase in improvement (or the decrease in the cost per unit) could have occurred by chance only. These should not be confused with the significance values in Tables 12 and 13, which represent the probability of the points lying on the straight line by chance only. The Defect Removal Rate learning curve was not used since several of its individual data points (i.e., one of the 12 students x 9 projects = 108 data points) had zero or one defect only, which lead to unreliable results.

Table 14: Significance levels of the differences

[illegible]	[illegible]
Productivity LC	not significant at 0.2 level
Size Estimation Ability LC	0.005
Time Estimation Ability LC	0.1
Productivity Estimation LC	not significant at 0.2 level
Defect Rate LC	0.001

The differences in the means of the Size Estimation Ability and the Time Estimation ability were significant at very high levels. The differences in productivity learning curve were not significant (even at 0.2 level), as expected, since as mentioned before, it does not depend on the cumulative output alone but also on complexity. However, what concerned us was that the differences in the Time and Productivity Estimation abilities were not significant either. While the former was still significant at 0.1 level, the latter was not significant even at 0.2 level (see Table 14 above). Yet when we look at the learning curve of productivity estimation ability (figure 9), we find the highest value of learning obtained (60%) and a correlation of 0.95 (at 0.0005 level). So why is there a contradiction?

The answer lies in the fact that the tests for comparing the differences in the means look at the variance within each mean also (i.e., variance between the 12 points whose mean is taken). In case of productivity, several subjects initially had no clue as to what their productivity was. As a result some of their estimates were as high as 481% away from the actual values. In one case the mean error of all the 12 subjects was as high as 87%. Therefore, due to this high within group variance, the effect of the differences in the means seemed to be not significant, although our learning index, slope and correlation show that it is quite high.

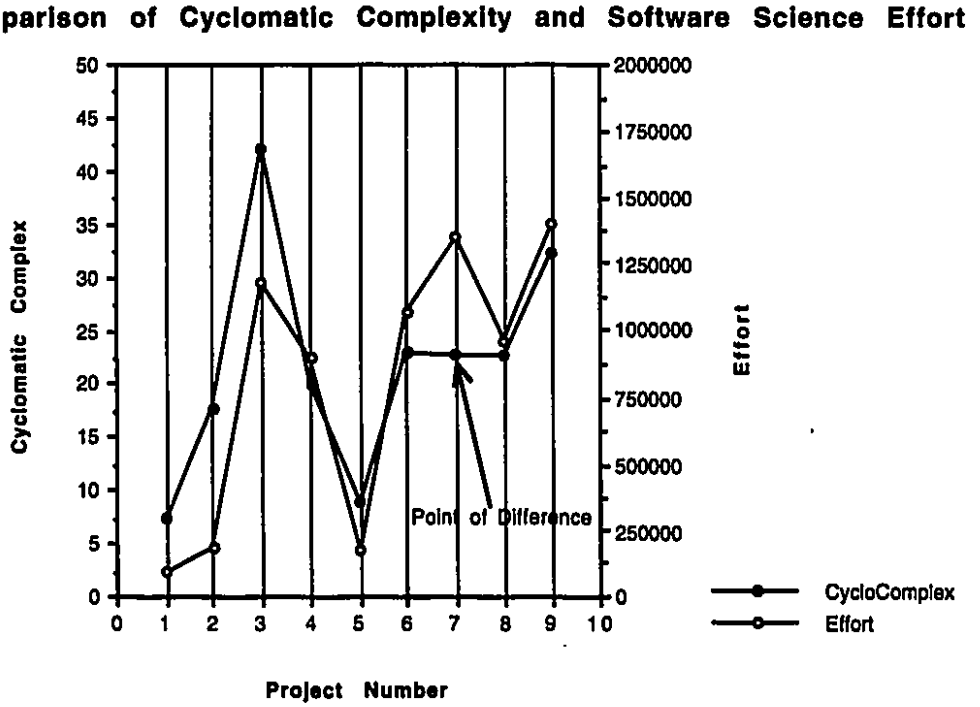
Of all the six learning curves, the poorest correlation has been found in the Productivity learning curve because productivity depends on defect quality as well as on the complexity. However, complexity is not something which can be measured easily. The next section (6.2) discusses an analysis of complexity, which will then be incorporated in our model in section 6.3.

6.2 Complexity Analysis

As described in section 4, the two most popular complexity metrics are the Software Science of Halstead and the Cyclomatic Complexity of McCabe. We wanted to select the metric which most closely resembled the solution complexity of the projects. For this purpose we decided to first carry out a complexity analysis of both the metrics on all the projects.

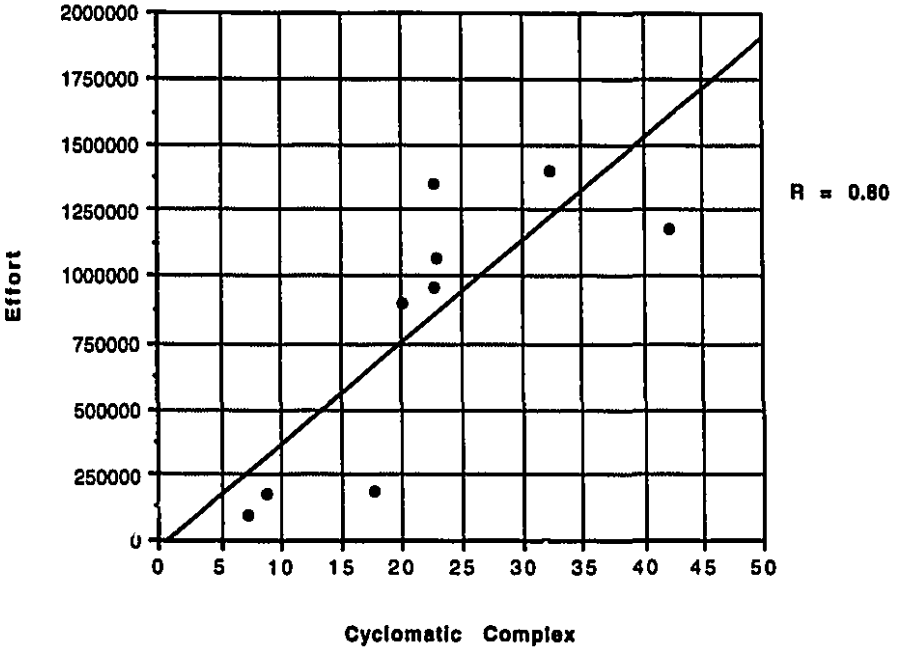
There are various software science metrics, such as *length, volume,* etc., of which *effort* comes closest to our requirements. Unfortunately, the scales (order of magnitude) of the effort metric are quite different from those for cyclomatic complexity's scales. Whereas the average value of latter was found to lie between 5 to 50 in our projects, the former ran from 30,000 to 1500,000 (see Appendix J for sample complexity results from PC-Metric tool). Hence, for comparison purposes, we decided to plot the two on normalized scales, using a double-scaled graph as shown in Figure 23.

Figure 23: Graph of Complexities vs. Project No.



This graph gives an excellent visual idea of how well the two complexity metrics correlate. The only significant difference between the two metrics lies in project 7. Figure 24 shows a plot of the two metrics against each other, with a correlation coefficient of 0.80 (at 0.005 significance level).

Figure 24: Graph of the two complexities against each other

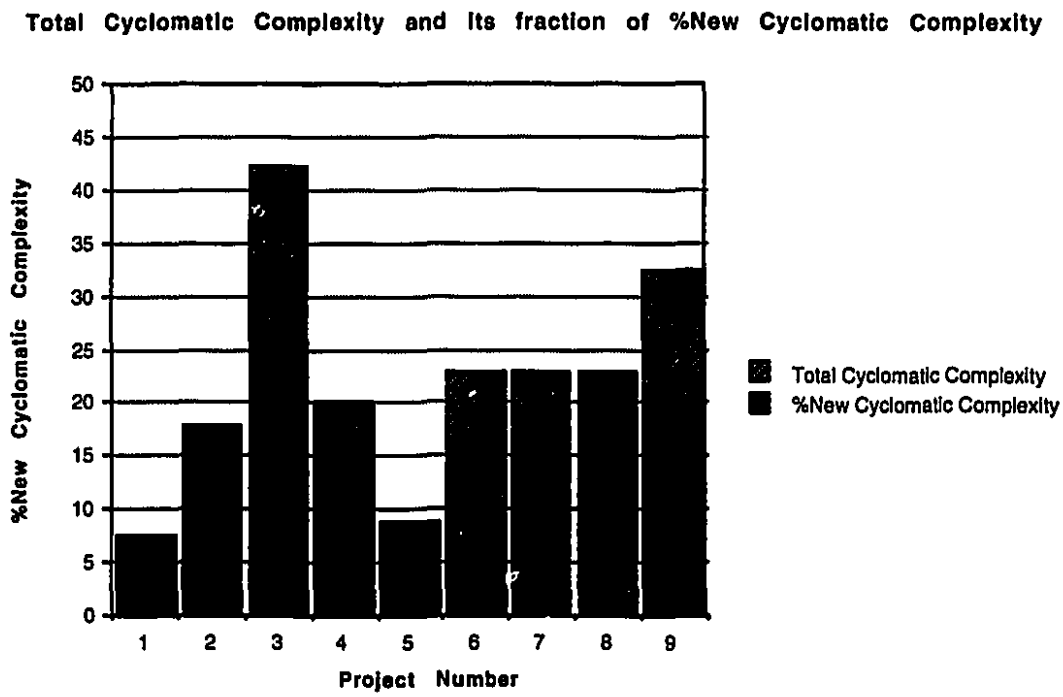


At the end of the experiment, these graphs were shown to some of the subjects at random. These subjects further consolidated our belief that the complexity of the solution was quite closely depicted by these two metrics. For example, all the subjects asked said that project 3 was the most difficult one, and that project 5 was a lot simpler than its predecessors (projects 3 and 4). Hence, the opinion of the subjects coincided with both the complexity metrics (after only the %New code was considered, see next paragraph). This vastly increased our confidence in trusting the two metrics. However, we had to use only one metric for our analysis, and so it was decided to use the Cyclomatic Complexity because its scale is more convenient to comprehend due to smaller numbers.

There is one more problem, however, and that is regarding the reused code. The complexity metrics depend on various factors and one of them is length. Since we are measuring the complexity of the solution, the reused code adds extra length at

practically inexpensive solution-cost. Hence a program with 75% reused code (as in project 6) has a comparatively high solution complexity than it would have if we consider only the new and modified (25% of the) code. Hence we decided to decrease the cyclomatic complexity values with the same fraction as the percentage reused code, and call it %New Complexity. Although the best thing would have been to use just the new and changed code and find its complexity separately, we did not have a precise track of which code was new and which was reused. Therefore, we just used the fractional method, assuming that in the long run of 9 projects, things would average out. Figure 25 shows the values of %New Complexity along with the Total Complexity. Note that the %New Complexity of project 2 is much more than that of projects 6, 7 and 8 although the latter had higher total complexities. These result are also in accordance with what the subjects had told us about their views on the solution complexity.

Figure 25: Graph of the Total and New Complexities vs. Project No.



Now that we have agreed on using the %New Cyclomatic Complexity, we can go ahead and plug it into our existing model of productivity and defect-quality, as they vary with the cumulative output. This 4-variable analysis is done in the following section.

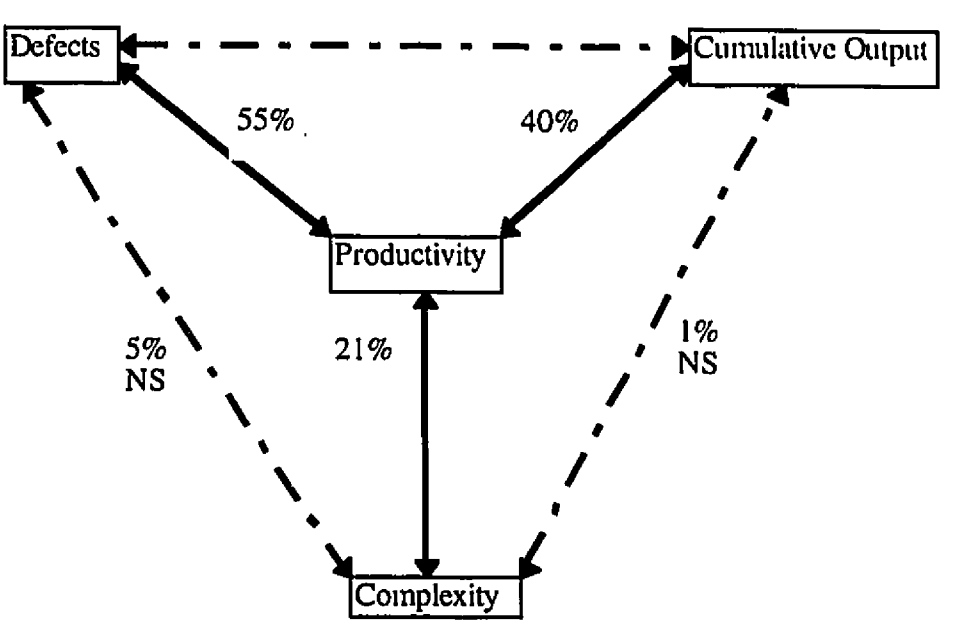
6.3 The 4-Variable Model

Both, the productivity and the defect rate, should depend upon the cumulative output, due to the first-order learning process. However, whereas in case of the defects-cumulative output relationship, we get a value of R as high as 0.96, in the case of productivity-cumulative output relationship, the value of R is only 0.65. If complexity is the other variable affecting productivity (and hence keeping the value of R low), then it should do the same for defect rate. It is logical to believe that both the defect rate and the productivity would depend on the complexity. Higher the complexity, more should be the number of defects injected and less should be the productivity. But in our case, apparently, the complexity is affecting only the productivity. In this section, we will study how these four variables are affecting each other, and hence how they effect the progress rate.

In order to analyze this 4-variable model, we calculated the *shared-variances* between these variables. The amount of agreement between any two measures tells us the extent to which they are measuring the same thing. This is called the amount of shared or common variance [38] [39]. Shared variance between a dependent variable A and an independent variable B tells us what percentage of A can be associated by the causal relationship with the variable B.

Figure 16 shows how the variables are related (NS means not significant). Here 40% of the variation in Productivity can be associated by the cumulative output factor, while 55% can be explained by the defect-rate. Note that an additional 21% is due to complexity, making the total variance exceed 100%. This is because defect-rate and cumulative output are not mutually independent either, and hence there is an overlap of variances which causes the sum to exceed 100%.

Figure 26: Model for Productivity



In Figure 16, Complexity and Cumulative Output are our independent variables. We will now study the two dependent variables, which are productivity and defect-rate, and try to develop 3-variable equations (models) comprising of these four variables. We can also produce 4-variable equations, but with our sample size of 12 subjects only, their significance would not be appropriate. In fact, even for 3-variable models, a sample size of 12 is not fully adequate, unless very high values of correlation are found. But in this experiment, due to our constraint of having only 12 subjects, we don't have any choice but to go ahead with a 3-variable model. Also, we will now be using the log-linear models instead of the linear ones as done in most past studies, because for 3-variables, they give a better representation of the relationship between the variables.

6.3.1 Productivity

We had found in section 6.1 that unlike the other learning curves with high values of correlations, in case of the productivity learning curve, the correlation was only 0.63. Hence cumulative output alone is not sufficient to account for the changes in productivity. In fact, it only has a 40% shared variance with productivity. This means that some third variable should be added to our model to get a better picture. We had contemplated that the complexity can be that third variable. However, contemplation alone is not enough, and we need appropriate statistical checks. For this purpose we used the partial correlation coefficient,

which give the correlation between two variables when a third one has been controlled for. So, partial $R^2_{Y.X|Z}$ means that after having controlled for the variable Z (i.e., after having the knowledge of the relationship between Y and Z), by what percentage are our errors in predicting Y decrease when the variable X is also added to the model [38] [39].

Figure 27: Adding the third variable to Productivity model

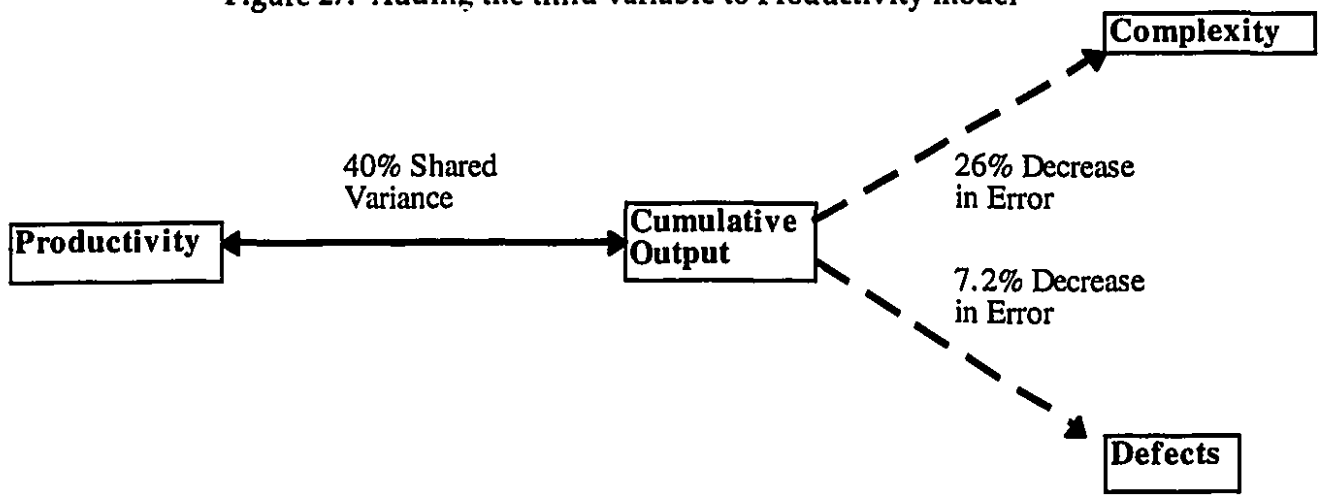


Figure 27 shows that the productivity and the cumulative output share 40% of the variance. Now, after controlling this association in productivity caused by the cumulative output, our errors in predicting the productivity decrease by another 26% if in addition to cumulative output, we also have the knowledge of complexity. Similarly, our errors would decrease by 7.2% if we include the defects-rate in the model of productivity and cumulative output. Obviously, if we can only have a 3-variable model (given our small sample size), it would be more desirable to include complexity instead of the defect-rate. Hence our model becomes:

$$\Pr oductivity = \frac{a\ Cumulative\ Output^{b}}{Complexity^{c}} \quad \{Equation\ 1\}$$

$R = 0.76$

$a = 21\ b = 0.16\ c = 0.15$

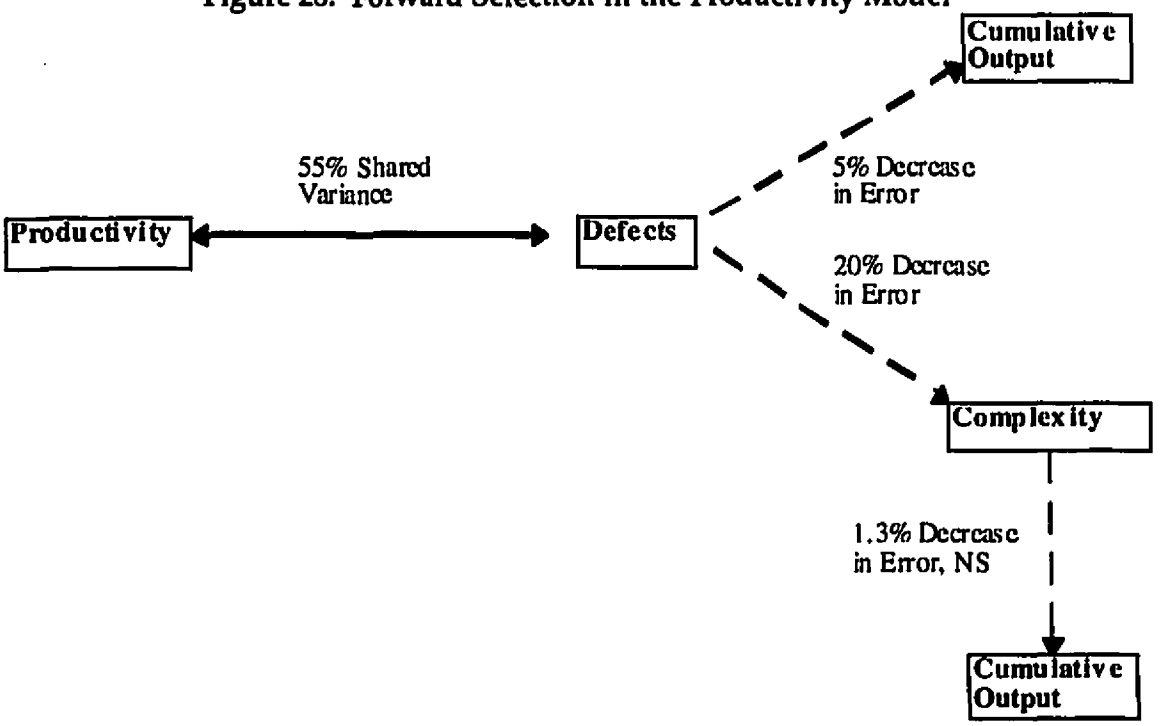
This equation suggests that more the complexity, less the productivity, and more the cumulative output, greater the productivity. This is in complete harmony with our rational expectations. Now the correlation has increased from 0.63 to 0.76. Adding a fourth variable (defect-rate) would not cause any significant decrease in

the error, at the expense of increasing the complexity of the equation. Furthermore, a 4-variable model derived out of only 12 subjects, would not have high statistical significance. Hence we might assume that this 3-variable equation is the most optimum one.

However, we are wrong. As it turns out, our initial assumption of starting with a productivity-cumulative output relationship, and then subsequently adding other variables, was not fully correct. We could have got better results had we used the *forward selection* method [39].

In this method, we start by selecting variables which are the most important and continue step by step adding other variables in order of importance. From figure 26, the highest shared variance which productivity has is with defect-rate (55%) and hence our model should start with a relationship between these two variables before adding cumulative output or complexity. Figure 28 shows that after accounting for the defect-rate, there can be a further decrease of 20% in error in predicting productivity if we include the complexity variable in our model also. However, by including cumulative output to the original two variables, the error decreases by only 5%.

Figure 28: Forward Selection in the Productivity Model



Hence we select complexity as the third variable and get the following model:

$$Productivity = \frac{a}{Defects^b\ Complexity^c} \quad \{Equation\ 2\}$$

$R = 0.81$

$a = 142\ b = 0.22\ c = 0.15$

Adding the fourth variable (cumulative output) to this model would further decrease the error by only 1.3%, which is not significant. Hence we retained to our three variable model, which implies that higher the complexity or more the defects, the value of productivity decreases. The correlation coefficient now is 0.81, which is greater than 0.76, which we obtained in the previous model (equation 1). Note that the forward selection method always leads to the highest possible value of correlation.

An interesting point about this model is that it apparently does not include the cumulative output, and hence our whole purpose of studying the learning curve (i.e., progress with cumulative output) seems to be understated. However, this is not true because of the obvious reason that the variable defect rate has a 71% shared variance with cumulative output. Hence there is a strong relationship of correlation 0.85 for the model:

$$Defects = \frac{a}{Cumulative\ Output^b}$$

Substituting this in the equation 2 above gives us the equation 1. Hence though seemingly different, both the equations are linked to each other in such a way that they fit in logically and intuitively.

6.3.2 Defect-Rate

One way to find a model for the defect-rate is to simply use equation 2, and express defects in terms of the other two variables. Hence from,

$$Productivity = \frac{a}{Defects^b\ Complexity^c} \quad \{Equation\ 2\}$$

we can get

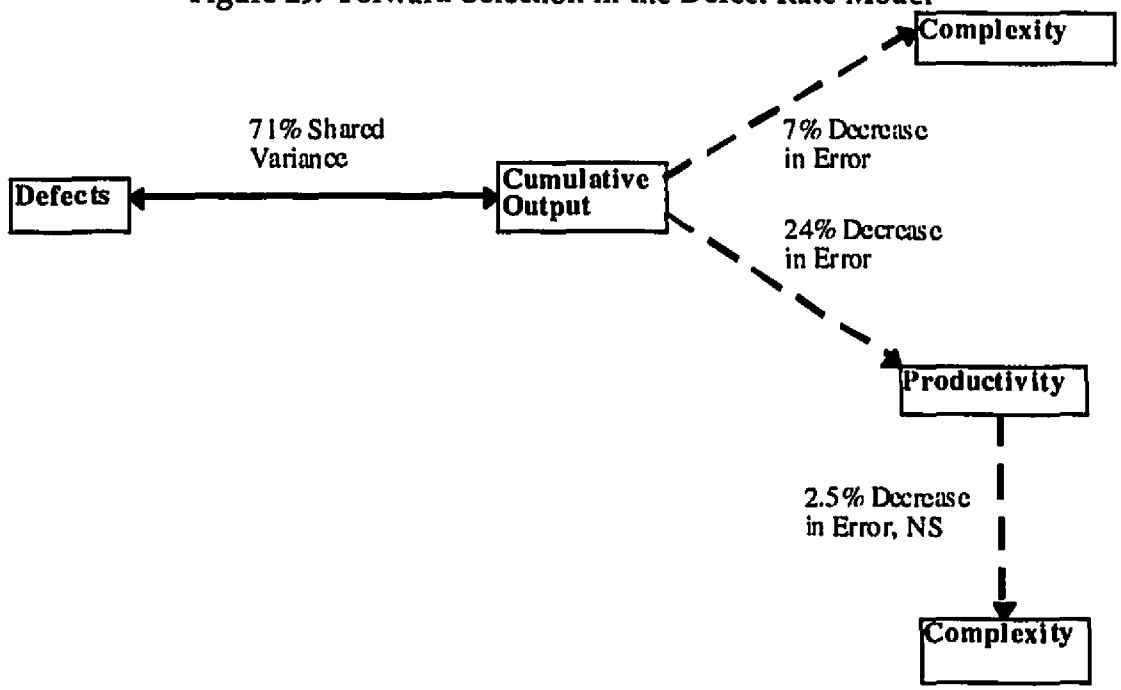
$$Defects = \sqrt[b]{\frac{a}{Productivity\ Complexity^{c}}}$$

or

$$Defects = \frac{d}{Productivity^{d}\ Complexity^{e}} \quad \{Equation\ 3\}$$

However, we know that the most optimum method for finding the highest correlation is forward selection. Figure 29 shows the application of this method.

Figure 29: Forward Selection in the Defect Rate Model



The highest shared variance of defects was with cumulative output (see Figure 26). After accounting for this 71% shared variance, the largest decrease in error for predicting the defect-rate was obtained by adding productivity to the model. This error decrease is 24%, which is much more than the 7% obtained by adding Complexity. Hence we get the following model:

$$Defects = \frac{a}{Cumulative\ Output^{d}\ Productivity^{e}} \quad \{Equation\ 4\}$$

$R = 0.90$

$a = 65300\ b = 0.45\ c = 1.11$

Notice that the correlation has increased from 0.85 to 0.90 and also that it is higher than the correlation of 0.81 in equation 3. Adding complexity to this model further decreases the error by only 2.5%, which is not significant.

This model shows that as the time passes, the defect-rate decreases. But in addition to this, it also confirms the relationship which we derived in equation 2, i.e.:

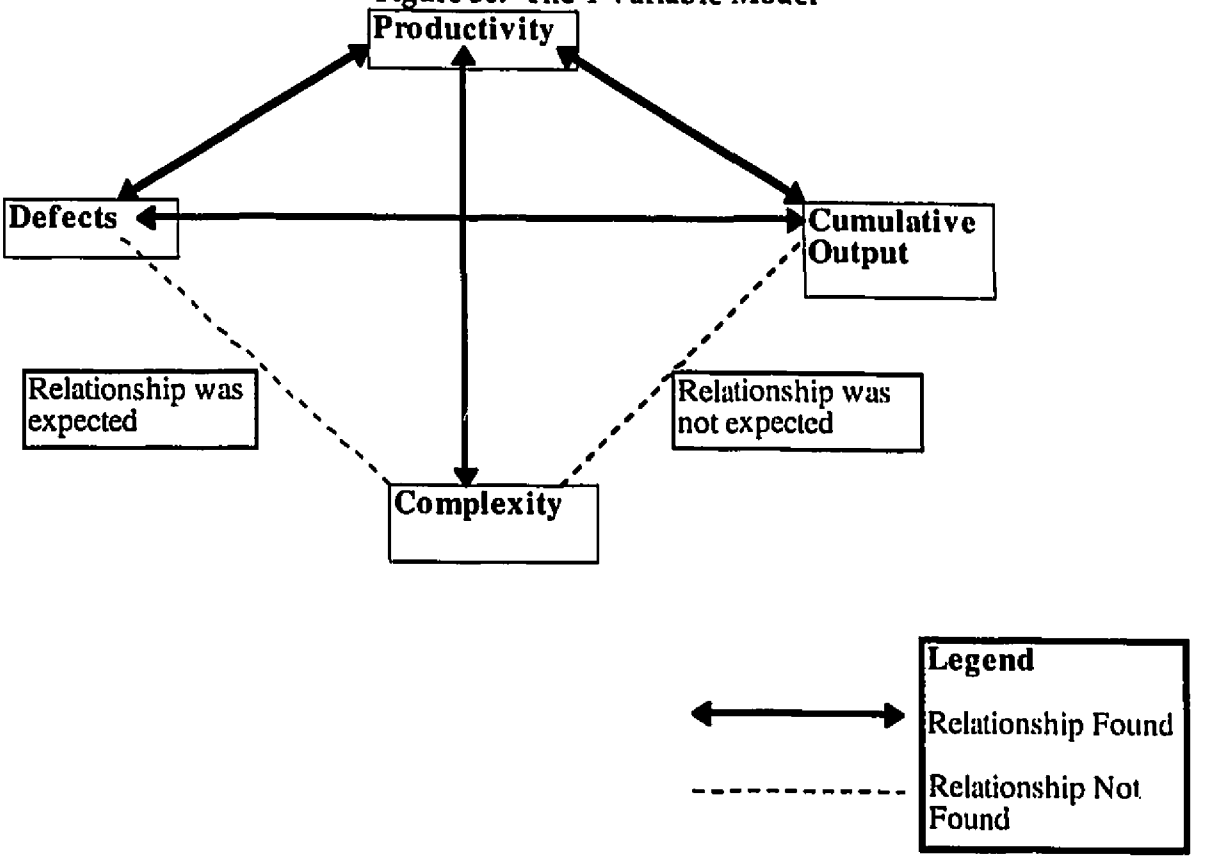
from equation 2, $Productivity = \frac{a}{Defects^b}$

from equation 4, $Defects = \frac{c}{Productivity^d}$

The first relationship is easily explainable. The more defects one injects, the longer it takes to debug them, and hence the lower the productivity. However, the second relationship, which actually is the same as the first one but in a different form, is a bit hard to comprehend intuitively. What it means is that a high defect-quality is attributed to having a high productivity. A high productivity means that you are following a better software process, and that in turn means that your defect-rate is low.

Figure 30 gives a summary of how the four variables are related to each other.

Figure 30: The 4-Variable Model



We were never expecting a relation between the complexity and the cumulative output, since the projects were randomly assigned without any knowledge of the complexity. Rest all the variables are related to each other with the exception of the defect-complexity relationship. Although we should be expecting that more the complexity higher the defect rate, we did not obtain any significant causality between the two. The reason, as mentioned before also, is that we got a very high correlation between defects and cumulative output, which dominated the relationship of defects with other variables.

6.4 Engineering Training in 2nd Order-Learning

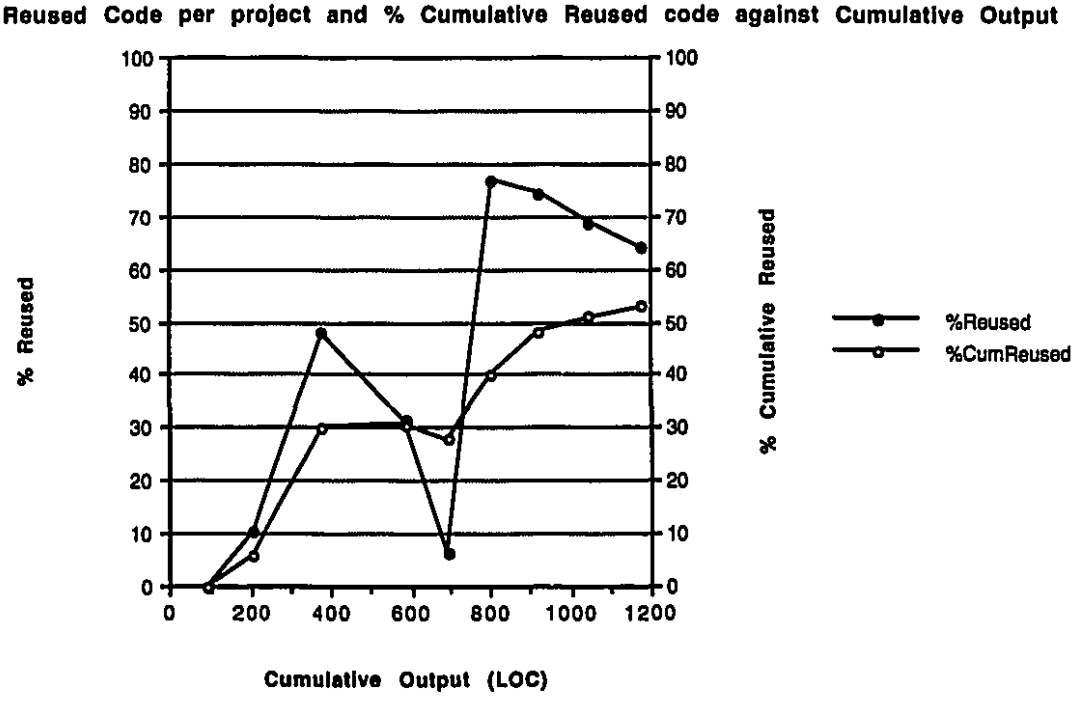
In this section, our main objective is to find if the second order learning has indeed helped the subjects considerably beyond the first order learning. As mentioned earlier, this is difficult to study and it is even more difficult to prove results statistically in this case. Nevertheless, we made an attempt to mathematically analyze the fractions which each order of learning contributed. In section 4.2.2, we had described three factors which can be attributed to the second

order learning. These were:

- Reused Code
- Size Estimation
- Code Reviews

Reusability is an attribute of the second order learning because it is the technical training which teaches this concept to the subjects and it is the technology which the subjects use to maintain the libraries for reused code. However, it is difficult to differentiate between the two types of learnings using the reused code alone as a measure. This is because we can never be sure of when the reused technology was injected in the process. Therefore, we will use the other two constructs (those of Size Estimation and Code Reviews) to analyze the second order learning, since we will be formally injecting them at specific points in our process. As far as the reusability is concerned, all we will mention is that this phenomenon was used extensively throughout the experiment. Figure 31 gives an overview of this.

Figure 31: Reused Code vs. Project No.



The %Reused Code for a project N is the reused code in project N, as a percentage of the total code in project N. The %Cumulative Reused code for a project N is the

sum of all the reused codes for projects 1 through N, as a percentage of the total cumulative LOC at project N. The final cumulative percentage of reused code in the 9 programming projects was 54% (see figure 31, last data point in the %Cumulative Reused graph). This includes the 0 % reused code during the first project, where obviously there could have been no reuse. During the last half of the experiment (the last four projects), the average of the four points representing percentage reused code had increased to above 70%. Hence without going into any mathematical treatment or statistical tests, one can get a fairly good idea of how important a role the reused code (and hence the second order learning) played during the course of the nine projects. In any case, in our experiment we pursued with the other two technologies, Size Estimation and Code Reviews, and not Code Reuse.

Now we will turn to our experimental design (explained in section 5.3.3).

We had mentioned that if X1 (Size Estimation training) and X2 (Code

Reviews) are independent or mutually exclusive, we can have the following quasi-experimental time series (with within subject) design:

OOOX1OOOOOO

OOOOOOX2OOO

In case of Size Estimation training (X1), our design can be represented as:

O1 O2 O3 X1 O4 O5 O6 O7 O8 O9

Here the observations (O1-O9) represent the percentage size estimation error. If we take the mean (M1) of O1-O3, then that is the average size estimation error before the training was given to the subjects. Similarly, the mean (M2) of O4-O9 is the average error after the treatment. In our case M2 was less than M1 by 21.26% (see Table 14). A comparison of `two means' test [49] (using t-tables) showed that our difference between the two means was significant at 0.05 level. This statistically confirms the fact that subject's performance after the training was much better than that before the training. A similar analysis on the Code Reviews training (X2) with the defect-rate as the metric showed that M2 < M1 by 20.79%. However, in this case the results are significant at only 0.15 level. This is because the variance within the two groups (O1-O6 and O7-O9) was quite high and hence there is a 15% chance that our results could have given a high difference between the two group-means by chance only. Table 15 summarizes these results.

Table 15: Means before and after the injection of technology

Technology	Pretest Observations	Posttest Observations	Percentage M1 < M2	Significance of Difference
Size Estimation Techniques	O1 - O3	O4 - O9	21.26%	0.05
Code Reviews	O1 - O6	O7 - O9	20.79%	0.15

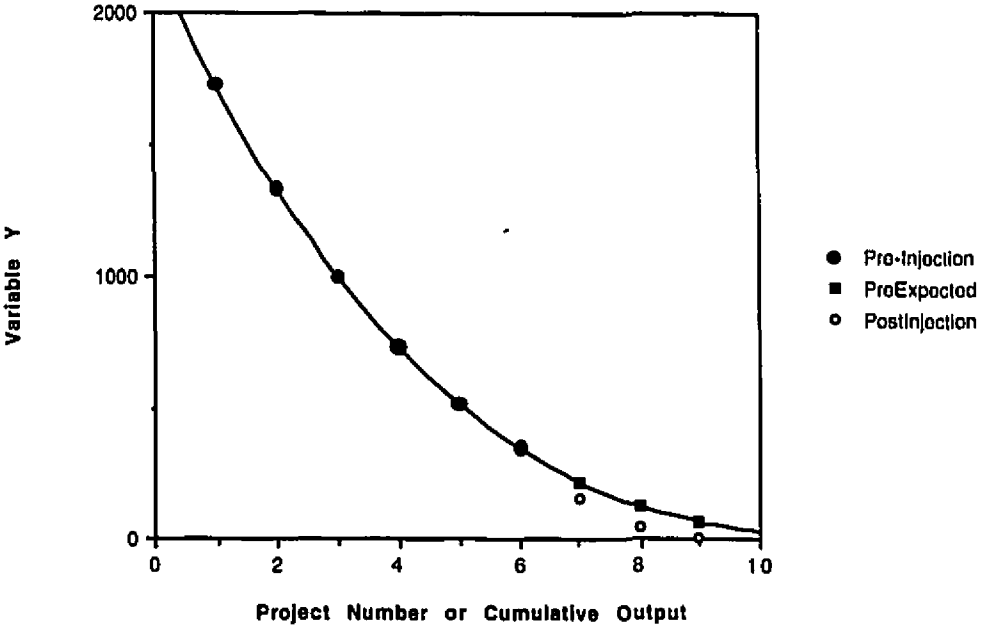
Regardless of the significance level of the above two results, the only conclusion, which we can draw is that the performance of the subjects is better after the training than it was before the training. Before the training, the subjects were using only the first-order learning. After the training, the subjects incorporate a second-order learning mechanism. However, we should not forget that even after the training, the subjects do continue to undergo first-order learning also, since such a learning cannot be stopped just by the injection of second-order training. Therefore, we can say nothing about the effects of second-order learning since it is probable that the increased performance after the training might have occurred due to some increase (due to any unforeseen circumstances) in the level of first-order learning.

To solve our problem in finding the proportion of each type of learning, we decided to use a different approach. Consider the Code Reviews (X2). Our design can be represented as:

O1 O2 O3 O4 O5 O6 X2 O7 O8 O9

If we plot only the points O1 through O6, and find the best linear or log-linear fit through them, then that will be the equation of the learning curve representing the first order learning alone. This equation would represent the number of defects in terms of the cumulative output (or time). Based on this equation, we should be able to predict what the number of defects should be for the cumulative outputs corresponding to the next three observations (O7, O8 and O9) after the training. In other words, we would be extending our learning curve beyond the cumulative output level of O6 (as shown in figure 32) in order to predict the Y-axis (defect) values for the X-axis (cumulative output values) for the points O7 through O9.

Figure 32: Sample graph showing interpolation



These values can then be compared with the actual values obtained for the defect rates corresponding to the cumulative outputs for the points O7 through O9. Whereas the predicted values represent only the first order learning, the actual points would represent the extra decrease in the defect-rate attributable solely to the second order learning aspect. The mean of the three predicted values can then be compared with the mean of the three actual values. In our case, the mean for the actual values was 12.62% less than the mean of the predicted values. Hence the addition of the second-order learning helped the subjects improve by about 13% in addition to the improvement they were undertaking due to the first order learning alone. These results are given in Table 16.

Table 16: Percentage decrease due to technology injection, compared to expected (interpolated) values from first-order pre-injection learning only

Technology	Percentage Decrease
X1 = Size Estimation Techniques	6.5 %
X2 = Code Reviews	12.62 %

A similar analysis on the size estimation training yielded the figure of 6.5% reduction in estimation error due to the second order learning in addition to the first-order learning. Hence our results are about 6% in one case and about 13% in another. These differences can be accounted for as follows: for X1, there were only three pre-treatment values. Since we are following a logarithmic model, these values show a very steep decreasing trend. Hence the predictions which we make about the future points are based on the same sharply decreasing curve. Therefore, we predict much lower values than normal since our pre-treatment data set is very small (only 3 points). This is why the actual values seem to be not significantly lower than our predicted values. However, in case of X2, we had a pre-treatment set of 6 data points, which gave us a curve which had more or less settled down after the initial steep fall. This large pre-treatment data set gives us a better idea of what to predict for future. Hence the predicted values are not lower than what they normally should be. This is why the actual values show a more significant decrease than the predicted values.

6.5 Analysis of First Order Learning

Having shown that the second order learning does contribute in addition to the first order learning, we will now try to analyze the latter to see what factors might affect it. As mentioned before, determinants of first-order learning might include the person's general experience, specific experience on jobs of a given type, education, sex, age, etc. [42]. Our approach is to study if there is any relationship between the personal capabilities of the subjects and the rate of learning. For this purpose, we decided to give each subject a capability index, based on (i) Experience and (ii) Performance. We believe that experience alone is not a sufficient predictor in judging a person's capability level. A person with a 10 year programming experience might still be less productive and/or slower in learning than a person with a 5 year experience. Hence we should also look at the performance factors of the subjects, such as the absolute (not relative) values of the productivitys of the subjects. In this way we can study if a person who is more productive or has a lesser defect rate is faster in learning than a person with lower levels of productivity or defect rate.

Given below shows our scheme of calculating the capability index. By no means are we stating that this scheme is the most optimum one. Naturally there can be various other ways of ranking the subjects, with no specific method being the best one. However, what we tried was to include a wide variety of factors encompassing most of the fields of experience and performance. Since the

capability index is a quantitative number, we had to develop quantitative scales. Although these scales have been chosen arbitrarily, we made all possible attempts to keep the sensitivity high, and to assign them weights properly. We are quite confident that given the wide variety of factors considered by us, our method is good. The summary of our procedure is given in Table 17 while the details are attached in Appendix L. Note that points given in parenthesis represent the maximum possible points in that category.

Table 17: Details of Personal Capability Index

Category	Metric	Points
Experience (60)	Total Experience in Programming	15
	Diversity in Programming Languages	10
	Level of Education	10
	Job experience in computer software/hardware related field	15
	Experience with Software Packages	5
	Experience in Software Engineering and Object-Oriented Design	5
Performance (40)	Productivity	15
	Defects/KLOC	10
	Defect Removal Rate	5
	Grade assigned to the subject in the PSP course	10

Our procedure requires detailed data from each subject about each and every language the subject has programmed in, each and every software package used, the time the subject has used these languages and packages, the total job experience and level of education of the subject, etc. This data was obtained directly from the subjects using a 6 page questionnaire (see Appendix A). For reliability, the subjects were later interviewed (after 3 weeks) and their responses to the questionnaire were rechecked. The nature of the questionnaire was such that the subjects were made to give even the finest details. In one case the subject was returned the

questionnaire since insufficient data had been supplied. The first 4 pages were designed by the software engineering group at McGill University while the last two were designed by Humphrey at Software Engineering Institute.

The learning index, p, for each subject was then plotted against its corresponding capability index. Since there were six learning curves, we had six learning indices. So we took the mean value of them; however, we had to ignore the learning index for the defect removal learning curve since there were cases when the subjects had none or only one defect in the projects, which lead to unreliable data. So the following five learning indices were used, which again can be divided into two categories:

(i) Estimation Skills

- p in Size Estimation Abilities
- p in Time Estimation Abilities
- p in Productivity Estimation Abilities

(ii) Performance

- p in programmer productivity
- p in defect-quality

Table 18 shows capability index listed with the learning index.

Table 18: Capability Index vs. Learning Index

[illegible]	[illegible]	[illegible]
1	35.6	30.4
2	37.1	37.6
3	37.1	14.4
4	38.3	17.9
5	46.7	31.4
6	47.3	30.5
7	50.4	16.0
8	52.1	15.4

9	55.6	26.8
10	59.5	10.8
11	62.4	-18.0
12	74.4	12.2
Mean	49.71	18.79
Std. Dev.	11.44	13.99

The capability index ranged from 35.6 to 74.4 with a mean of 49.71 and a standard deviation of 11.44. The mean of roughly 50 is exactly the mid-point of our total range of 0-100, and hence boosts our confidence in the scheme used by us. The average learning index is 18.79% with a standard deviation of 13.99. However, observe that there is an outlier with p = -18.0, which represents negative learning (or forgetting). Actually, this subject had a p = 18.8 in the Performance category, but unfortunately had a p = -42.5 in the estimation category. Probably something had gone wrong with the subject, which lead to poorer estimates. According to Sheil [51],

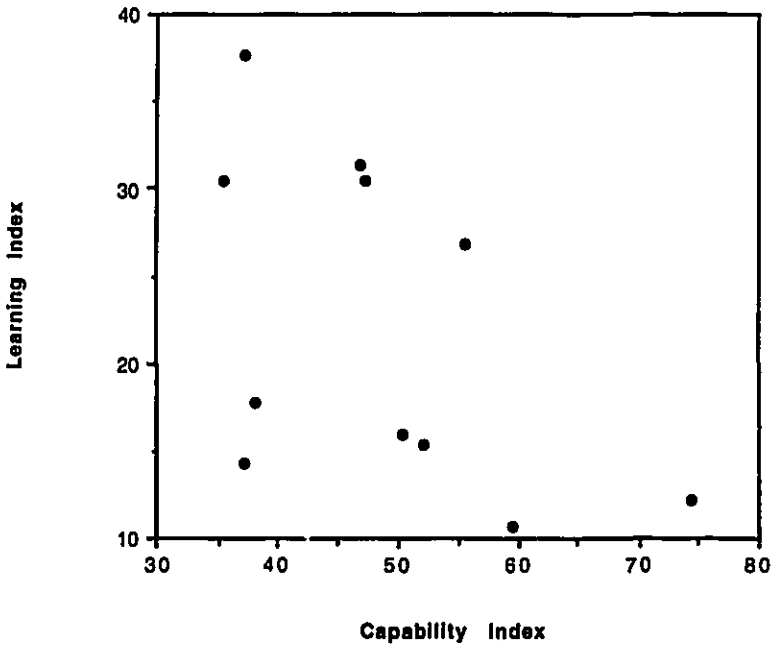
> *"One of the more striking symptoms of high individual variability is that one occasionally finds a small number of participants whose scores on some measure are far outside the range for the group to which they belong. Their presence not only invalidates the common statistical techniques, but it can both mask real differences (by increasing the variance) and create illusory ones. Although non-parametric statistics and/or data recoding can be used to avoid the technical problems, it is far preferable simply to discard outliers before the analysis. The reason for this is that very extreme observations strongly suggest that the individual is not typical of those to whom the results are to be generalized. For example, such an individual might be doing something quite different from the other participants, possibly as a result of having misunderstood the instructions. (The classic example is the participant who falls asleep during a reaction time experiment.)"*

After discarding this outlier, our average learning index was 22.13% with a standard deviation of 8.91. Note that this is very close to the value of 20% as reported in various past research results [29][60]. The breakup of the two categories of the learning index shows that the average learning index in the estimation category was only 12.23% while that in the performance categories was as high as 28.60%. Hence obviously, for our subjects, estimation was a more difficult task to

improve in as compared to improving their performance in productivity and defect-quality.

Figure 33 shows a graph of the learning index against the capability index. No statistically significant or conclusive relationship was obtained.

Figure 33: Graph of the two indices against each other



We tried to study the causality amongst the individual categories also, i.e., between the two learning categories (estimation and performance) and the two capabilities categories (experience and performance). However, again we could not find any conclusive evidence of any relationship between the categories.

Nevertheless, we did discover a phenomenon in our experiment, which is quite expected. We found that those subjects whose performance level is already quite high, learn less than those whose base performance level is low. For example, it is easy for a subject to increase the productivity from 30 LOC/hr to 60 LOC/hr (100% increase) as compared to a subject who has to increase from 60 LOC/hr to 120 LOC/hr. The latter may increase from 60 LOC/hr to 90 LOC/hr, but that would be only a 50% increase. Such a phenomenon was quite obvious and expected, and we are merely confirming its existence. However, besides that, we did not find any

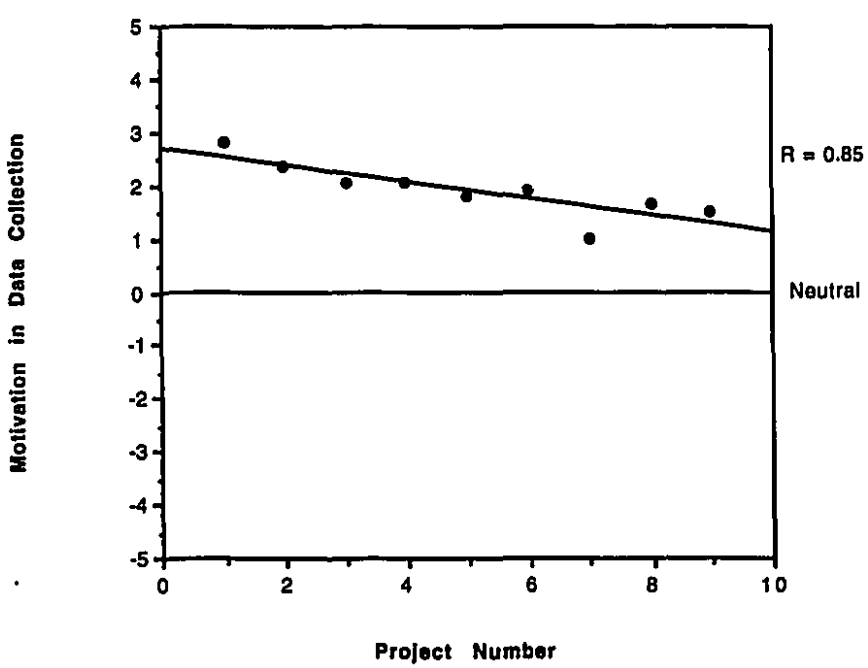
other relationships or causalities. Of course, we are not implying that no other relationships exist; we are only stating that more work with larger sample sizes is needed to make improved predictions about any such relationships.

6.6 Management Motivation in Second Order Learning

In section 4.2.1, we had given the details of the survey instrument used in measuring the (a) motivation, (b) interest, (c) satisfaction of the subjects after every project as well as (d) the degree to which they found the projects to be useful. These were quantified on a scale of -5 to 0 to +5. The subjects responded separately for the programming aspect and for the data collection aspect for these four variables. Hence there were a total of eight variables.

All these variables were then plotted on a discrete scale of 1 through 9 (representing the nine projects). Figure 34 shows a typical plot of how all the eight plots looked like.

Figure 34: A typical graph of the subjective metrics



These eight plots had a mean value for slope of -0.15 and a mean value for correlation coefficient of 0.79. For all eight variables, on the scale of -5 to 0 to +5 (completely unmotivated to neutral to extremely motivated), the average value obtained was 1.8 (where 1.0 = slightly motivated and 2.0 = quite motivated). Table 19 below gives details of these statistics:

Table 19: Statistics on the subjective measures

Attribute	Mean	Std. Dev.
Correlation Coefficient, R	0.79	0.055
Slope	-0.15	0.031
Values for all the 8 variables	1.8	0.15

The low values of standard deviations show that the variance in the data was not high. The correlation coefficients are quite high (on average, 0.79) with a high significances (at least at 0.05 level). However, in this case we are not interested in how well the data lies on a straight line, but rather, on whether or not the data is increasing or decreasing. Recall that our objective is to find if the motivation and the other attributes increased or decreased during the course of the experiment.

For this purpose we carried out a comparison of means test (since each of the nine points is a mean of the 12 data points for the students). It was found that none of the eight variables showed any significant change, even at 0.20 level. In other words, the probability that the slight decrease which we obtained in all the variables could have occurred by chance alone is more than 1 in 5, which cannot be considered significant at all. Hence, for all practical purposes, we can claim that the variables stayed constant. This implies that in doing data analysis on other variables (such as defect rate, productivity, etc.), we can be confident that these other variables are not dependent upon these subjective variables (e.g., motivation).

7.0 Discussion and Comparison with Related Work

Section 5 discussed the four types of research purposes: discovery, demonstration, refutation and replication. Discovery means finding something new out, which was not known before. Demonstration means to verify that some hypothesis or theory is correct. Demonstration is sometimes more accurate than Discovery since in it more validation is carried out, as compared to Discovery which might take place without formal prior goals. A first-time demonstration (i.e., when a theory has not been empirically verified before) can be as important as the Discovery. Usually replication is not that important since although it increases our confidence in some hypotheses, it does not tell us something new.

Our work is a good example of Discovery and first-time Demonstration. In case of some parts of our work, past data exists from other studies (though in differing environments and sometimes for different objectives). Hence there is a flavor of external-replication also in our work. Note that an ideal replication involves carrying out the experiment with the same experimental design, preferably under the same environment as in our case. In this section we discuss the significance of our results and compare them with other related work.

The prime objective of our work was to study the learning curve, although it turned out that we carried out various other tasks of importance. Our results of the progress functions are a replication of the past work done in numerous other fields of industry, manufacturing and management in the sense that we have shown that learning indeed does occur. However, in the field of software engineering, our work is one of the first detailed empirical studies of its kind. Hence here we have shown that our hypothesis that learning occurs in this field also is correct. This is a first-time Demonstration. We also obtained some concrete insight into how much of what type of learning occurs (first order or second order). This is more close to discovery since in the field of software engineering, no detailed hypotheses about these have been developed.

Our work is comprehensive since we used three different models: linear, quadratic and log-linear, and produced detailed quantitative results. The most important feature in the learning curve is the progress ratio, p. Past work

[29][60] has shown that a 20% value of learning rate or progress ratio is most often cited. Our average value of the learning index for the 12 subjects was 22.13% (section 6.5), which is close to the past results. However, we also showed that there are cases where the learning rate can be much higher. The value of the progress ratio for the learning curves ranged from 15% to 60%, with a mean and a median of 36% (section 6.1). These results suggest that in the field of software, much higher rates of learning are possible as compared to other manufacturing fields such as the aircraft construction, etc. More such studies would help in generalizing this aspect. This increases the importance of incorporating progress functions in managerial decision making in the software field.

There are several aspects related to learning which, although we studied, did not yield any thing conclusive. The analysis of first order learning is a good example. We found that in our case the personal capabilities of an individual, which were further refined into experience and performance, do not have any relationship with how fast a person learns or how well a person performs (section 6.5). However, we could not refute this hypothesis either. In other words, we cannot prove that these variables have no relationship. It just so happened that in our case there was none; other studies might give perfectly valid results showing that in their case some form of relationship does exist. One important aspect, however, of this study was the detailed procedure used in calculating the capability index, using 10 different criteria. Past examples of such detailed techniques in the field of software engineering are not known to us.

Our study showed that the subjective measures related to motivation, stayed constant throughout the experiment (section 6.6). Here also, we cannot hypothesize that these variables will always remain constant. However, we stress on the need for measuring them in such studies, since if they are not constant then they have the potential to affect the results. We would also like to point out the difference between the data analysis based on personal discretion and the statistical data analysis. All the subjective measures clearly indicate that they are slightly decreasing as the experiment proceeds. However, statistically speaking, this decrease is not significant and hence they *must* be considered as not decreasing, or constant. In such cases we have to opt for statistical results, since the personal discretion might be rational, but it can never be proven. On the other hand, statistical methods have been in use for many years, and are known to provide accurate deductions from the data.

Our second main objective was to compare first order and second order learning. It is difficult to measure which one helps more in the overall learning process. Gone are the days of Gestalt who used to stress on the first order learning, such as experience and insight. Researchers such as Arrow [5], Levy [42], Adler [2], Clark [2], Hirsch [29], etc. have hypothesized that second order learning is equally or more important. However, there are only a few studies which give empirical evidence to this, such as [29]. Furthermore, several such studies are on past data only, such as the ship-building during the second world war [46][24], and ignore any form of experimental design. We believe that it is important for researchers to gather data themselves instead of relying on past logs from manufacturing firms, because it is improper to impose an experimental design on work that has already been done. In this sense it can be considered that our work has advanced the knowledge in the field of software.

Our results showed that the second order learning helps the individuals to learn up to 13% more than the first order learning alone (section 6.4). We are not satisfied with these results because we believe that the actual number can be much higher. This low value has been obtained because we had no control groups. If we had one randomly assigned group with engineering technology and training and the other one without it, then the results obtained would have shown higher differences. However note that even the value of 13% is not low. In large organizations with huge budgets, this percentage can save millions of dollars, even after accounting for the cost of injecting the technology and providing the training. However, detailed studies are needed for confirming these facts.

Finally, our third main objective was the analysis of the 4-variable model. For this purpose we had to carry out a comparative complexity analysis of the Halstead's and McCabe's complexity metrics. Note that there are numerous papers written on these two metrics, and several of them have tried to compare the two [20] [14]. Our comparison consists of a large sample size of 108 programs, and gives a correlation of 0.80 (section 6.2) between the two (at 0.005 level). This result alone can be published as a study.

It is difficult to describe the results of our 4-variable model. On one hand it can be termed as a discovery, since we originally had the three variable model in mind, and ended up discovering that complexity was a fourth variable also. But on the other hand, all the results are so much in harmony with intuition that they are merely a demonstration of what we were expecting. For example, the hypothesis that more the complexity less the productivity is

not a discovery but a demonstration of what we had a belief in due to rational thinking. In some cases, past work has also been done on various separate parts of our 4-variable model. For example, Putnam [45] and Humphrey [30] have given figures showing that increased productivity is due to increased product-quality. Hence our work is sort of a replication of previous studies, though in a different environment.

An interesting aspect about our work is that there are completely unexpected results or unusual deductions. For example, we showed that the learning does occur, that the second order learning is useful in addition to first order learning, and that the productivity, defects and complexity are related just the way one should expect them to be related. So then what is important about our results? The answer to this is that although most of these hypotheses had been thought to be true intuitively or logically, they had not been verified before, especially in the field of software engineering. Furthermore, our work involves a true scientific study and hence its results can be considered to be valid and reliable. Finally, we believe that our study has focused on nearly all the aspects of the learning curve model, and hence provides completeness in work. Nevertheless, there are still various other points which need further study. These are discussed in the next section.

8.0 Conclusion and Future Work

In this work, our objective was to study the progress functions for individuals, who can expect continuous improvement in productivity as a consequence of (i) a growing stock of knowledge and experience gained by repeatedly doing the same task (first-order learning) or (ii) due to technological and training programs injected by the organization (second-order learning). Progress functions are important since they are used in industry for making managerial decisions regarding cost estimating and budgeting, production and labor scheduling, product pricing, etc. [4][29] While considerable research on this topic has been done in industrial and manufacturing sectors [60], we found little emphasis in the software process field.

It is important that software firms begin paying more attention to the progress functions in making managerial decisions. According to Dr. Hashim, assistant professor of software engineering in Malaysia, this trend is being followed increasingly in Asia, although only a little has been published on it. In fact, the promotions of the employees in various software firms depend on if they have followed the learning curve properly or not.

Our results confirm that a 20% learning is normal. The PSP course is currently being taught at McGill University again, and one goal of our research group is to replicate our experiment to get a better confidence on this figure. We strongly encourage other researchers to carry out external replication as well, especially in the industrial environment. Giving the industry a verified figure would be doing a service to it. Such research results can then be used for making delivery schedules, estimating the product-quality and cost, as well as on keeping track of how well the personal capabilities of the employees are, e.g., in estimating the size of the projects.

We feel that more work is needed in differentiating the first order and the second order learnings. As mentioned in the data analysis, we feel that the 13% figure we calculated regarding the contribution of second order learning beyond the first order is underestimated. A verified figure would allow the managerial personal to initiate training programs, purchase new technology and stress more on organizational attempt to induce learning amongst the employees. No management would do this unless they are certain that the percentage increase in improvement is significant. Hence a cost-benefit study

is also needed in this field. We would recommend that a controlled experiment should be designed in order to carry out this task.

Another interesting aspect of our work has been the 4-variable model. Although in the past various studies have been carried out on static models involving productivity, defect-quality and complexity, not much work has been carried out on the dynamic models involving time or cumulative output. In our work, we have produced equations on the relationships between these models. Of course the parameters of these equations would differ from environment to environment. Hence further replications are needed to gain a better confidence on these values. Furthermore, in our case the subjective variables such as the motivation turned out to be constant. This might not be the case in other studies. Hence more attention should be paid to these subjective variables also, and see if they have an effect on the 4-variable model.

There are various topics which we referred to in our work only briefly but which can be studied in more detail. Although we carried out a detailed complexity analysis, controlling or varying the complexity was never our objective. Work can be done on controlling the complexity and then observing how it effects the other two variables, productivity and defect-quality. Similarly, the latter variable has only been studied in the context of its relationship with the other variables. However, given the detailed data available to us consisting of each and every defect from a range of 190 defect types, can be used to carry out a more thorough study with prime focus on defects alone. Such work might include studying the defects by the phase they are injected in.

One of the secondary objectives of our work was to study the personal capabilities of the subjects. Our method of assigning a capability index, though very detailed, is highly subjective. More work is needed on developing a procedure which is more acceptable in the software engineering community, so that the results can be consistently compared with other works. We had the limitation of a small sample size. If possible, studies should be carried out involving a much larger number of subjects. Any relationships between the learning index and the capability index might then be more obvious.

Although the PSP has been around in the software engineering field for a while, relatively less attention has been paid to it. Business firms are usually busy on improving their organizational processes because they feel that they have a higher cost-benefit ratio. Data is needed to convince them that the PSP

is also useful. Controlled studies are needed on the PSP itself, to see how much it helps the individuals.

Our research model covered a wide variety of topics from the field of software engineering. We are very satisfied with the effort we put in the design and execution of our experiment and also in the analysis of the data. As a result, we are more than happy with our results as well. We believe that our work can contribute to the software engineering community and would be pleased if other researchers explore our objectives in further detail.

APPENDIX A

The Initial Questionnaire

Student Questionnair

McGill University Winter '94 CS 631

This form must be filled out by all the students, irrespective of their job experience. Please fill this before you start work on Ass # 1, and return it on Thursday Jan 20, 1994.

Name _____

Date _____

Please list all Degrees attained (Bachelors or above) _____

Please list all your Majors/Minors _____

Total Full-Time job experience in Computers related field _____

Other Part-Time job experience in Computers related field _____

Please list all the languages you have programmed in for MORE than a semester (e.g, Prolog, LISP, Scheme, Assembly Intel/Motorolla, Ada, Fortran, Algol, Basic, Pascal, Object Pascal, C, C++, COBOL, etc) Also indicate the approximate total LOC written in these languages. Put an asterisk on your favorite language. *(LOC stands for a Physical Line in your program source code, not including the blank and comment lines).*

Language	**Total LOC & Total Months**

Please list any other languages in which you have written at least one program. Also indicate the approximate total LOC written in these languages

Language	**Total LOC & No. of Programs**

Total LOC written in any Language _____

Total years of Programming experience _____

Please list all the Database related packages you have used (eg, DBASE IV, RBASE, ORACLE, SYBASE, SQL, Relix, FOXPRO, WindowBase), and the time you have used them for:

DBASE	Total Time

Please list all the Spreadsheet related packages you have used (eg, MS-Excel, LOTUS 1-2-3, Quattro, LOTUS IMPROV), and the time you have used them for:

SpreadSheet	Total Time

Please list all the Statistical Tools you have used (eg, SAS, SPSS, Systat), and the time you have used them for:

Statistical Tool	Total Time

In the space below, please list any other package, tool, language, etc related to Software Process Engineering, User Interfaces, Statistics, etc which has not been asked for above. You may also take this space to describe some Software Process Engineering related project you might have undertaken at your job or at some undergraduate course:

_____

_____

_____

_____

_____

SELF APPRAISAL

Please list the ONE (or at most two) principal language(s) you have been using recently (or have used the most) for programming: _____

For the above mentioned Principal language(s), and also for C/C++, please read the definitions given below and then answer the questions:

Total Output: This is defined as the total LOC you have written in a programming language.

Based on your prior experience, estimate your Total Output in LOC you have written.

Principal _____ C/C++ _____

Programmar-Productivity: This is defined as the number of LOC you develop per unit of time.

Based on your prior experience, please estimated your productivity in LOC per hour.

Principal _____ C/C++ _____

Defect-Quality: This is the number of Defects or Errors which a programmer makes per 100 LOC. Please look at the Defect Types Standard for a listing of all the types of errors.

Based on your prior experience, please estimate your Defect-Quality in No. of Defects per 100 LOC.

Principal _____ C/C++ _____

Reusability Rate (New LOC Only): Sometimes you directly copy/paste LOC from your own previously written programs (without modifying those lines) to save time and effort. Reusability can then be defined as the No. of Reused LOC per every 100 LOC you write.

Based on your prior experience, please estimate your Reusability Rate (New LOC Only) in No. of Reused LOC per every 100 LOC programmed.

Principal _____ C/C++ _____

Reusability Rate (New & Modified LOC): Sometimes one has to slightly modify the Reused lines, which are copied/pasted from ones own previous programs.

If we add these modified lines also to your answer above, what will be your Reusability Rate in Reused/Modified LOC per 100 LOC.

Principal _____ C/C++ _____

Defect-Removal Efficiency: This is defined as the number of defects you find and remove per hour of debugging (finding and fixing) time.

Based on your prior experience, what do you think is your Defect-Removal Efficiency in No. of Defects (traced & fixed) per hour.

Principal _____ C/C++ _____

Time Estimation Error: Before one starts to code, one has a rough/precise estimate of how long it would take to finish the program. Often this estimate is wrong and the actual time is different, say 10% more or less than the estimated time. This Percentage Time Estimation Error can be calculated by taking the difference in the Actual and Estimated Times, and converting it to a percent.

Base on your prior experience, what % error do you think is there in your Time Estimation.

Principal _____ C/C++ _____

Size Estimation Error: Similarly, programmers usually have a rough/precise estimate of how long their program is going to be, in LOC. The difference in the actual program size as compared to the planned program size gives the % error for size estimation.

Based on your prior experience, what do you think is your % size estimation error.

Principal _____ C/C++ _____

In the space below, please describe what experience (Total LOC and Time) you have in using C++ (not C alone) and in any other Object Oriented Design experience.

_____

_____

_____

_____

Thank you

Disciplined Software Engineering - Student Questionnaire

NOTE: PLEASE COMPLETE BOTH SIDES

General:

Name _____ Date _____

Instructor _____ University _____

Education:

Highest degree attained: _____ Major: _____

Current field of study: _____

Have you had courses in?

Statistics ____

A physical science ____

Software project management ____

Formal software methods ____

Software:

Programming Languages you have used:

C _____ C++ _____ Pascal _____ Object Pascal ____

Other _____

Language you will use in this course: C++

Approximate total LOC written in this language: _____

Approximate total LOC written in any language: _____

The year you wrote your first program: _____

Design methods you use: _____

Do you use formal methods? _____

List the formal methods you use: _____

_____

Number of years you have used these methods: _____

If you would be willing to answer a follow-up questionnaire in the future, please give a permanent address through which you can be reached:

_____

_____

_____

_____

_____

_____

If you have been employed to write programs, please answer the following questions:

The Tools and Methods You Use:

Principal language you use: _____
Approximate total LOC written in this language: _____
Approximate total LOC written in any language: _____
Design methods you use: _____
Formal methods you use: _____
Years you have used these methods: _____

Your Personal Process:

Do you make a project plan before development? ____
Do your plans include size estimates? ____
Do your plans include defect estimates? ____
Do your plans include resource estimates? ____
Do your plans include schedules? ____

Do you do personal design reviews? ____
Do you do these reviews before you write the code? ____
Do you do personal code reviews? ____
Do you do these reviews before you test? ____
Do you do these reviews before you compile? ____
Do you have peer inspections of your code? ____
Do you do a personal review before the inspection? ____
Do you compile before the inspection? ____
Do you unit test before the inspection? ____

Estimated lines of code you develop per hour: ____
Approximate defects/KLOC in your first program test: ____
Approximate % defects you find before first test: ____
Approximate % defects you find before first compile: ____
Approximate % of your development time spent in personal reviews: ____
Approximate % of your development time spent in inspections: ____
Approximate % of your development time spent in compile: ____
Approximate % of your development time spent in test: ____

If you are familiar with the CMM, please estimate the overall process maturity of your organization, your most recent project, and your personal processes:

Your organization's approximate overall process maturity: ____
Your project's approximate overall process maturity: ____
The approximate overall maturity of your personal process: ____

APPENDIX B

Logical LOC Coding Standards

Counting Standards for Logical LOC

1. Blank lines are not counted
2. Comments on separate lines are not counted
3. No statements or punctuation marks within '(...)' or '[...]' are counted.
4. Count once each occurrence of ',' in declarations.
5. Count once each occurrence of the following selected key words and tokens (exception is For-Loop statement)::

   ;
   {
   }
   };
   #
   case
   default
   do
   else
   for
   if
   public
   switch
   while
6. Count all function/method/main/procedure declarations.

Now, using the above counting standards, we will develop coding standards, such that each Physical LOC will contain ONE and ONLY ONE logical LOC. In this way, by simply counting the Physical LOC, we would be (indirectly) counting Logical LOC, hence avoiding the difficult task of writing a Logical LOC counter.

APPENDIX C

Physical LOC Coding Standards

Coding Standards for Physical LOC

(representing ONE and ONLY ONE Logical LOC for each Physical LOC)

DECLARATIONS

1. For declarations, declare only one variable per line.
2. All function/method/main/procedure declarations should normally be written on one line, except for when they cannot fit in on a single line. The I/O parameters should be declared within the parenthesis of the calling Function's header.

MAIN BODY

3. For the main body of the program/functions, each and every line should have ONE and ONLY ONE of the following key words or tokens (exception is For-Loop statements and function/main/procedure headers):

```
;
{
}
};
#
case
default
do
else
for
if
public
switch
while
```

COMMENTS & BLANK LINES

4. There is no restriction on the number of Blank lines.
5. There is no restriction on the comments which are on separate lines.
6. For the Program Header, it is recommended that the students follow the C++ Coding Standards given in the handout attached to the Defect Type Standards. In particular, the following topics should be considered:
   - Program name and number
   - Version and release information
   - Who developed it and when
   - The function Performed
   - Any special usage guidance and warnings
7. For the Procedure/Function/Method Headers, it is recommended that the students follow the C++ Coding Standards given in the handout attached to the Defect Type Standards. In particular, the following topics should be considered:

Function performed
Return format and limits
Any special usage guidance and warnings

8. For inline comments, the following topics should be considered:
   Purpose and function of complex operations
   Purpose of any special parameters and variables
   Nature of any special data types
   Any other comments to clarify program operation

Examples of VALID codes:

```
#include <stream.h>                  //one and only one # token  per line
#define  TRUE 1                      //one and only one # token per line
                                     //blank line
typedef    struct {                  //one and only one {  token per line
    float  average;                  //only one declaration per line
    int    marks;                    //only one declaration per line
    char   name[40];                 //only one declaration per line
    } student;                       //one and only one };  token per line
                                     //blank line
union x {                            //one and only one {  token per line
    char   ch;                       /only one declaration per line
    int    i;                        //only one declaration per line
    }                                //one and only one }  token per line
                                     //blank line
class queue {                        //one and only one {  token per line
    int    q[100];                   //only one declaration per line
    int    front,                    //only one declaration per line
           rear;                     //only one declaration per line
public:                              //only one Keyword public per line
    void init(void);                 //only one declaration per line
    void enque(int i);               //only one declaration per line
    int  dequ(void);                 //only one declaration per line
    };                               //one and only one };  token per line
                                     //blank line
void func1(void)                     //Function declaration takes one line
    {                                //one and only one {  token per line
    int    temp1;                    //only one declaration per line
    int    temp2;                    //only one declaration per line
    float  temp3,                    //only one declaration per line
           temp4,                    //only one declaration per line
           temp5;                    //only one declaration per line
    static int x;                    //only one declaration per line
                                     //blank line
```

```
    x = x + y;                          //one and only one ; token per line
    return(x);                          //one and only one ; token per line
    }                                   //one and only one } token per line
                                        //blank line
int queue::enque(void)                  //Method declaration takes one line
    {                                   //one and only one { token per line
    if (front == rear)                  //only one Keyword if per line
            {                           //one and only one { token per line
            cout << "underflow";        //one and only one ; token per line
            return 0;                   //one and only one ; token per line
            }                           //one and only one } token per line
    else                                //only one Keyword else per line
            {                           //one and only one { token per line
            rear++;                     //one and only one ; token per line
            return q[rear];             //one and only one ; token per line
            }                           //one and only one } token per line
    }                                   //one and only one } token per line
                                        //blank line
main (void)                             //Main declaration takes one line
    {                                   //one and only one { token per line
    do                                  //only one Keyword do per line
            {                           //one and only one { token per line
            x = x+y;                    //one and only one ; token per line
            y = y-x;                    //one and only one ; token per line
            }                           //one and only one } token per line
            while                       //only one Keyword while per line
                    (x > y);            //one and only one ; token per line
                                        //blank line
    while ( (x > y) && (x > 0) )        //only one Keyword while per line
            {                           //one and only one { token per line
            x = x+y;                    //one and only one ; token per line
            y++;                        //one and only one ; token per line
            }                           //one and only one } token per line
                                        //blank line
    for (count = 1; count < 10; count++) //for loop is exception case
            {                           //one and only one { token per line
            cout << "hello\n";          //one and only one ; token per line
            putchar('.');               //one and only one ; token per line
            }                           //one and only one } token per line
                                        //blank line
    switch(ch)                          //only one Keyword switch per line
            {                           //one and only one { token per line
            case '1':                   //only one Keyword case per line
                    check_spell();      //one and only one ; token per line
```

```
                break;                  //one and only one ; token per line
        case '2':                       //only one Keyword case per line
                x = x + y;              //one and only one ; token per line
                y++;                    //one and only one ; token per line
                break;                  //one and only one ; token per line
        default:                        //only one Keyword default per line
                cout >> "Err";          //one and only one ; token per line
        }                               //one and only one } token per line
}                                       //one and only one } token per line
```

Examples of INVALID codes:

```
void                                    //Function declaration takes 4 lines
func2 (a, b)
int         a;                          //Arguments/parameters must be
float       b;                          //declared as func2(int a, float b)
{
int         local;
local = local + x;  a++;                //Two ; tokens  in one line
b--;
}

int         x, y;                       //Two declarations in one line
queue       aque, bque;                 //Two declarations in one line


while (x > y) x++;                      //2 Keywords/tokens, while & ;
while (x > y) {                         //2 Keywords/tokens, while & {
        x++;
        y--;
        }
```

APPENDIX D

Validation Form

Validation Form

Name _____

Status (MSc2, Prof, etc) _____

Field of Research _____

Experience (Yrs) in Software Engineering _____

Introduction

You have been selected to fill out this form because of your professional experience related to the field of Measurements in Software Process Engineering. We would really appreciate if you can spend 30 minutes of your precious time to answer some of the questions given below. But first, please read the following details of the Survey we are carrying out. You are kindly requested to give suggestions on Validating the Goals-Questions-Metrics used in this survey.

Objective

Our main objective is to study the Personal Software Process. We have a group of about 10 students, who would be writing about 10 small programs over a period of One Semester. For each program, we would gather data on their performance, such as the number of Errors they make, the number of Lines they Code, the period of Time it takes them to write the programs, etc. These Metrics are labeled M1, M2, M3, ... (a detailed listing is attached). We plan to use these metrics to answer certain questions, labeled Q1, Q2, ... These Questions are specifically designed to give us information about our Objectives or Goals, labeled G1, G2, ... Your task is to raise any objections you find about these Goals, Questions or Metrics and suggest any improvements. Your Validation should consist of checking for correctness, consistency and completeness. Please also feel free to add new Goals, Questions or Metrics in the space provided. In particular, you should be keeping an eye for objections such as:

Validity of Goals: Are the Goals important enough?

Questions Validity: Do the questions accurately answer the Goals under consideration?

To facilate in your validation process, given below are some items which you might check for in validating the Metrics and Questions [Validating Instruments in MIS Research, Straub] [Measurement: The key to application development quality, Walrad]:

Content Validity: Are instrument measures drawn from all possible measures of the properties under investigation?

Construct Validity: Do measures show stability across methodologies? Are the data a reflection of true scores of the kind of instrument chosen?

Reliability:	Do measures show stability across the units of observation? Could measurement error be so high as to discredit the findings?
Internal Validity:	Are there untested rival hypotheses for the observed effects?
Statistical Validity:	Do the variables demonstrate relationships not explainable by chance or some other standard of comparison?
Efficiency:	Does the metric produce a desired effect with a minimum of effort, expense, or waste?

Metrics

Given below is a listing of the 18 metrics used. To keep our task simpler, most of these are Objective metrics, and are already being used worldwide for various software related measurements.

M1 **Size Estimation Error**

This is the % error with which a student estimates the size of the program which is to be coded.

(| Estimated LOC - Actual LOC | / Actual LOC) x 100

Any Remarks on this Metric. Should we not use this metric? If not, then why not? Should this metric be modified? If so, then how? Should a new Metric be used instead? If so, please give details. It might be better to fill in your comments for each metric after you have read the Goals and Questions.

_____

_____

_____

M2 **Time Estimation Error**

This is the % error with which a student estimates the time required by him/her to code a program.

(| Estimated Time - Actual Time | / Actual Time) x 100

M3 **Real Defect Quality**

This is the number of defects per KLOC. The total number of coded lines includes the Reused lines also. For detailed work, this metric will be used separately for all the possible types of defects (e.g, Syntax Erros, Run Time Errors, etc).

(No of Defects of Type X / Total New & Reused LOC) x 1000

M4 **Apparant Defect Quality**

This is the number of defects per KLOC. The total number of coded lines does not include the Reused lines. For detailed work, this metric will be used separately for all the possible types of defects.

(No of Defects of Type X / Total New LOC) x 1000

M5 **Reused Code**

This is the number of LOC which are reused per KLOC.

(No of LOC Reused / Total LOC) x 1000

M6 **Real Programmer Productivity**

This is the number of LOC which a student codes per programmar-month (of 160 hrs). These include the Reused lines which are simply copied from previous work, and are NOT modified:

(No of New & Reused LOC / Total Time-hrs) x 160

M7 **Apparant Programmer Productivity**

This is the number of LOC which a student codes per programmar-month. These do not include the Reused lines.

(No of New LOC / Total Time-hrs) x 160

M8 **Student GPA**

This is the Cumulative Grade Point Average of the student at McGill University.

M9 **Students Self-Appraisal of Defect Quality**

This is the estimate of what the student thinks is his/her personal defect quality, in No of Defects per KLOC. This esitmate is taken only once at the begining of the semester.

M10 **Students Self-Appraisal of Programmer Productivity**

This is the estimate of what the student thinks is his/her persoanl programmer productivity, in No of LOC per Programmer-Month. This esitmate is taken only once at the begining of the semester.

M11 **Students Self-Appraisal of Size Estimation Error**

This is the estimate of what the student thinks is his/her average error rate in estimating the size of the programs before they are coded. This esitmate is taken only once at the begining of the semester.

M12 **Students Self-Appraisal of Time Estimation Error**

This is the estimate of what the student thinks is his/her average error rate in estimating the Time of the program which is to be coded. This esitmate is taken only once at the begining of the semester.

M13 **Students Self-Appraisal of Reusability**

This is the estimate of what the student thinks is his/her level of Reusing the code, in No of Reused LOC per KLOC. This esitmate is taken only once at the begining of the semester.

M14 **Students Interest**

This is the level of interest which a student takes in writing a program, as judged by the student on a scale of [-5 to 0 to +5] standing for [Completely Disinterested to Neutral to Very Interested].

M15 **Students Motivation**

This is the level with which a student is motivated to write a program, as judged by the student on a scale of [-5 to 0 to +5] standing for [Complete Lack of Motivation to Neutral to Very Motivated].

M16 **Students Satisfaction**

This is the level of satisfaction which a student achieves by writing a program, as judged by the student on a scale of [-5 to 0 to +5] standing for [Annoying to Neutral to Very Satisfying].

M17 **Defect Removal Efficiency**

This is the number of defects removed by the student per hour of time spent in finding and fixing the defects.

(Total No of Defects Found / Total Time taken to Find and Fix the Defects)

M18 **Students Self-Appraisal of Defect Removal Efficiency**

This is the estimate of what the student thinks is his/her persoanl Defect Removal Efficiency, in No of Defects Removed per Hour. This esitmate is taken only once at the begining of the semester.

Please list below any other Metrics which you think might be useful and describe how they may be used. If possible, give the number of the Question(s) they might be used to answer.

Goals and Questions

Having defined the metrics we would be using, we are listing 6 Goals and 13 Questions. Under each question, the names and numbers of the metrics which will be used are also listed.

G1 Investigate the Learning Curve of the Students in the context of the Personal Software Process with respect to the Estimation Ability.

Remarks/Objections/Modifications

Q1 How does the students ability to Esitmate the Size of the Program improve with time?

[M1 Size Estimation Error]

Q2 How does the students ability to Esitmate the Time required to code the Program improve over the passage of 10 programs?

[M2 Time Estimation Error]

Q3 Is there any relationship between the Ability to Estimate Size and Ability to Estimate Time?

[M1 Size Estimation Error]
[M2 Time Estimation Error]

G2 Does the Feed-Front from previous programs help the students?

Q4 Does the Defect-Quality of the student for a particular type of defect X increase based on the experience obtained from the previous programs?

[M3 Real Defect Quality]
[M4 Apparent Defect Quality]

Q5 Do the previous programs help the student by offering code which can be reused?

[M5 Reused Code]

G3 **Investigate the behavior of Programmer Productivity and Defect-Quality in the context of Personal Software Process.**

Q6 Does the student get more productive in writing code?

[M6 Real Programmer Productivity]
[M7 Apparent Programmer Productivity]

Q7 Is there any relationship between the Programmer Productivity and the Defect-Quality?

[M3 Real Defect Quality]
[M4 Apparent Defect Quality]
[M6 Real Programmer Productivity]
[M7 Apparent Programmer Productivity]

Q13 Does the students Defect-Removal Efficiency increase?

[M17 Defect Removal Efficiency]

G4 Study the Contingency of Personal Capability Factors on Personal Software Process

---

Q8 If the student makes any improvement, then is it solely due to the Process, PSP, or do the Personal Capability Factors of the student such as Skills, Hardwork, Experience, Intelligence, etc also affect it?

NB: This requires a graph of Student Improvement vs Ranking of Students by Personal Capability level.

Student Improvement can be judged from various factors such as Estimation Ability, Programmer Productivity, Defect-Quality, Defect-Removal Efficiency.

A Ranking of Students Personal Capability level can be made using the students GPA or by using the students performance during this semester (Productivity, Quality, Efficiency).

[M1 Size Estimation Error]
[M2 Time Estimation Error]
[M3 Real Defect Quality]
[M6 Real Programmer Productivity]
[M8 Student GPA]
[M17 Defect Removal Efficiency]

G5 **Are the students generally more optimistic about their capabilities?**

Q9 **How good do the students think they are in Estimation Ability, Productivity, Defect Quality, Defect Removal Efficiency and Reusability versus how well they actually are?**

NB: Before the start of the semester, the students will be asked to estimate what they think is their standard of Productivity, Quality, Efficiency and Estimation Ability. Their estimates will then be compared with the actual values obtained from their performance in the first two programs.

[M1 Size Estimation Error]
[M2 Time Estimation Error]
[M3 Real Defect Quality]
[M4 Apparent Defect Quality]
[M5 Reused Code]
[M6 Real Programmer Productivity]
[M7 Apparent Programmer Productivity]
[M9 Students Self-Appraisal of Defect Quality]
[M10 Students Self-Appraisal of Programmer Productivity]
[M11 Students Self-Appraisal of Size Estimation Error]
[M12 Students Self-Appraisal of Time Estimation Error]
[M13 Students Self-Appraisal of Reusability]
[M17 Defect Removal Efficiency]
[M18 Students Self-Appraisal of Defect Removal Efficiency]

G6 **To study some subjective aspects of Human Interest, Motivation and Satisfaction in the context of Personal Software Process.**

Q10 Does the students interest in writing programs increase or decrease with the passage of time?

Q11 Does the motivation with which a student writes programs increase or decrease with the passage of time?

Q12 Does the satisfaction which a student achieves by writing programs increase or decrease with the passage of time?

[M14 Students Interest]
[M15 Students Motivation]
[M16 Students Satisfaction]

If you feel that we can study some more Goals, then please mention them below. If possible, please also mention the metrics which can be used to measure those Goals/Questions.

Once again, we thank you for filling out this form. Please return it to Khalid Sherdil in MC Rm 334 or if you want it to be picked from your office, then please leave a message at 844-1378, 398-7084 or sher@binkley.

APPENDIX E

Motivation Form

Evaluation Form

In the questions below, 'Program' means the process of designing, coding and running a program satisfactoraly. The term 'Data' implies the process of collecting PSP Data. Please check the appropriate box. You must answer all the questions, though you will NOT be graded on what you answer.

Name _____ Assignment # ☐

"Interest" is defined as the 'willing attention' you took in the assignment. How much interest did you take in this assignment?

	-5	-4	-3	-2	-1	0	+1	+2	+3	+4	+5
	Completely Disinterested	Very Disinterested		Quite Disinterested	Slightly Disinterested	Neutral	Slightly Interested	Quite Interested		Very Interested	Extremely Interested
Program											
Data											

"Motivation" is defined as the desire or incentive given to you by the management in the program. How much motivation, do you think, were you given in this assignment?

	-5	-4	-3	-2	-1	0	+1	+2	+3	+4	+5
	Completely Unmotivated	Very Unmotivated		Quite Unmotivated	Slightly Unmotivated	Neutral	Slightly Motivated	Quite Motivated		Very Motivated	Extremely Motivated
Program											
Data											

"Satisfaction" is the pleasure you receive upon fulfillment of a task. How much satisfaction did you get from this assignment?

	-5	-4	-3	-2	-1	0	+1	+2	+3	+4	+5
	Completely Unsatisfact.	Very Unsatis-factory		Quite Unsatisfact.	Slightly Unsatisfact.	Neutral	Slightly Satisfactory	Quite Satisfactory		Very Satisfactory	Extremely Satisfactory
Program											
Data											

With respect to how much you learnt, did you find this assignment useful?

	-5	-4	-3	-2	-1	0	+1	+2	+3	+4	+5
	Completely Useless		Quite Useless		Not very useful	Neutral	Slightly Useful		Quite Useful	Very Useful	Extremely Useful
Program											
Data											

APPENDIX F

Sample Data Forms

Defect Recording Log

Student ______________________ Date __________
Instructor ______________________ Program # __________

Date | No. | Type | Inj. | Rem. | Fix Time
Description:

Date | No. | Type | Inj. | Rem. | Fix Time
Description:

Date | No. | Type | Inj. | Rem. | Fix Time
Description:

Date | No. | Type | Inj. | Rem. | Fix Time
Description:

Date | No. | Type | Inj. | Rem. | Fix Time
Description:

Date | No. | Type | Inj. | Rem. | Fix Time
Description:

Date | No. | Type | Inj. | Rem. | Fix Time
Description:

Date | No. | Type | Inj. | Rem. | Fix Time
Description:

Date | No. | Type | Inj. | Rem. | Fix Time
Description:

Date | No. | Type | Inj. | Rem. | Fix Time
Description:

Date | No. | Type | Inj. | Rem. | Fix Time
Description:

Time Recording Log

Date ____________

Student ________________________________ Program # ____________

Date	Start	Stop	Interruption Time	Delta Time	Job #	Phase	Activity	Comments

PSP2 Project Plan Summary

Student _____ Date _____
Instructor _____ Program # _____
Methods _____ Language _____

Summary	Plan	Actual	To Date
LOC/Hour			
Planned Time			
Actual Time			
CPI(Cost-Performance Index)			
% Reuse			
% New Reuse			
Test Defects/KLOC			
Total Defects/KLOC			
Yield			

Program Size (LOC):	Plan	Actual	To Date
Base			
+Reused			
-Deleted & Mod			
+Modified (M)			
+Added (A)			
Total LOC			
Total New&Chgd (A+M)			
Total New Reuse			

Time in Phase (min.)	Old %	Plan	Actual	To Date	New %
Planning					
Detailed Design					
Code					
Code review					
Compile					
Test					
Postmortem					
Total					
Total UCL (70%)					
Total LCL (70%)					

(continued)

PSP2 Project Plan Summary (continued)

Student		Date	
Instructor		Program #	
Methods		Language	

Defects Injected	Plan	Actual	To Date	New %
Planning				
Detailed Design				
Code				
Code review				
Compile				
Test				
Total Development				
After Development				

Defects Removed	Plan	Actual	To Date	New %
Planning				
Detailed Design				
Code				
Code review				
Compile				
Test				
Total Development				
After Development				

Defect Removal Efficiency	Plan	Actual	To Date
Defects/Hour - Code Review			
Defects/Hour - Compile			
Defects/Hour - Test			
DRL(CodeRevue/UT)			
DRL(Compile/UT)			

APPENDIX G

Sample PSP Templates

Size Estimating Template

Student _____ Date _____

Instructor _____ Program # _____

OBJECT NAME: NEW OBJECTS:	TYPE[1]	NUMBER OF METHODS	RELATIVE SIZE	ESTIMATED LOC
TOTAL (A)				
NEW REUSE OBJECTS:				
TOTAL (B)				
REUSED OBJECTS:	**BASE:**			
REUSE BASE(C)				
REUSE CHGD(D)				

E - Estimated New and Changed Object LOC:	A+B+D	
F - Regression Factor:	β_1	
G - Regression Factor:	β_2	
H - Estimated New and Changed LOC:	F+E*G	
I - Estimated Total LOC:	H+C	
J - Percent Reuse:	100(C/I)	
K - Percent New Reuse:	100*(B/H)	
L - Confidence Range:		
M - Lower Confidence Limit:	H-L	
N - Upper Confidence Limit:	H+L	
O - Actual Program LOC:		
P - Confidence Percent:		
	R	
	SIGNIFICANCE (Y/N)	

[1] L-Logic, I-I/O, C-Calculation, T-Text, D-Data, S-Set-up

C++ Code Review Guideline and Check List (example)

PROGRAM NAME AND #:

Suggestion	Complete a checklist for one program unit before starting the next. Review one checklist item completely before doing the next.	
Completeness	Ensure that the code covers all the design.	
Standards	Follow the coding, naming, and defect reporting standards.	
Includes	Check that includes are complete	
Initialization	Check variable and parameter initialization: - at program initiation - at start of every loop - at function/procedure entry	
Calls	Check function call formats: - pointers - parameters - use of '&'	
Names	Check name spelling and use: - consistent - within declared scope - structures and classes use '.' reference	
Strings	Check that all strings are: - identified by pointers - terminated in NULL	
Pointers	Check that pointers are: - initialized NULL - only deleted after new - new pointers are always deleted after use - always used within their defined range	
Output Format	Check the output format: - line stepping is proper - spacing is proper	
{} Pairs	Ensure that the {} are proper and matched	
Logic Operators	Verify the proper use of ==, =, ||, etc. Check every logic function for proper ()	
Line by Line Check	Check every line of code: - instruction syntax - proper punctuation	
File Open and Close	Ensure that all files are: - properly declared and used - opened - closed	

TEST REPORT TEMPLATE

Student _____ Date _____
Instructor _____ Program # _____
Methods _____ Language _____

Test Name/Number: _____

Step #	**Data/Action/Result**

Test Name/Number: _____

Step #	**Data/Action/Result**

Test Name/Number: _____

Step #	**Data/Action/Result**

PROCESS IMPROVEMENT PROPOSAL (PIP) FORM

PIP Number _____

Student _____ Date _____
Instructor _____ Program # _____
Methods _____ Language _____

PROBLEM:

Number	**Description:**

PROPOSED IMPROVEMENT:

Priority	**Description**

APPENDIX H

Defect Types Standard

Defect Types Standard

Documentation (101)

- Incorrect Comments 110
- Wrong Headers 120
- Incorrect Coding Standards 130
- Other Documentation 190

Build/Package/System (201)

- Wrong Include Files selected 210
- Incompatible Version of Compiler 230
- Using wrong Editor 240
- Wrong Module Export Import Interface 250
- Other Build/Package/System 290

Design (301)

- High Level Algorithm Design Errors 310
- Medium Level Algorithm Design Errors 320
- Low Level Algorithm Design Errors 330
- Faulty planning in Arguments passing 340
- Faulty planning in Linking Functions 350
- Errors in Flow Chart / Structured Chart 360
- Control Flow Logic Error 370
- Data Flow Logic Error 380
- Other Design 390

Pure Syntax (401)

- Incorrect Spelling 410
- Incorrect Punctuation 420
- Missing Tokens , . ; : ( ) := { } " : 430
- Typos 134
- Case Sensitivity Errors 440
- Other Pure Syntax 490

Compilation/RunTime/Semantic Errors (501)

Name Structures (5101)

- Unknown Identifier 5110
- Duplicate Identifier 5120
- Variable not Declared 5130
- Undefined Variable 5140
- Constant out of Range 5150
- Uninitialized Variables 5160
- Invalid Compiler Directives 5170
- Other Name Structures 5190

Simple Data Types (5201)

- Type mismatch 5210
- Type not defined 5220
- Const/Var confusion 5230
- Float/Integer Confusion 5240
- Error in defining Char type 5250
- Confusion in Local/Global Vars 5260
- Other Simple Data Types 5290

Enumerated Data Types 530

- Boolean T or F Confusion 531
- Wrong UserDefined Enumerated Types 532
- Other Enumerated Data Types 539

Data Structures (5401)

- *Arrays/Structures* 5410
  - Initialization Errors 5411
  - Array Maximum Capacity Exceeded 5412
  - Incompatible Data Type 5413
  - Not declared properly 5414
  - Out of Range Indices 5415
  - Problems with Hierarchy of Structs 5416
  - Other Arrays/Structures 5419
- *Pointers* 5420
  - Memory Allocation Errors 5421
  - Uninitialized Pointers/NULL values 5422
  - Wrong Assignment 5423
  - Wrong use of & 5424
  - Wrong use of * 5425
  - Wrong use of -> 5426
  - Error in Declaration 5427
  - Errors with New/Dispose 5428
  - Other Pointers 5429

Files 5430
- Invalid File Type 5431
- File not found 5432
- Path not found 5433
- File access denied 5434
- File not open 5435
- File not closed 5436
- Wrong File Pointer 5437
- Confusion in Binary/ASCII 5438
- Other Files 5439

Streams/Memory 5440
- Wrong number of bytes 5441
- Error in Memory Read/Write 5442
- Incorrect Seek,Rewind operations 5443
- Problems with Shared Memory 5444
- Other Streams/Memory 5449

Strings 5450
- String Length mismatch 5451
- Using Char instead of String 5452
- String Constant too big 5453
- String Concatenation problems 5454
- String Comparison problems 5455
- String Copy problems 5456
- Other Strings 5459

Classes/Unions (54601)
- Invalid Declaration of Classes 54610
  - Confusion in Public/Private 54611
  - Errors in Creating Insertors 54612
  - Using wrong Super/Sub Classes 54613
  - Errors in Protecting a Class 54614
  - Inheritance Problems 54615
  - Other Class problems 54619
- Errors in Declaring Objects 54620
- Errors in Messages 54630
- Problems with Methods/Functions 54640
  - Errors in Using Constructors 54641
  - Errors in Using Destructors 54642
  - Wrong Overloading of Functions 54643
  - Error in using Friend Functions 54644
  - Error in Inline/Macro Functions 54645
  - Error in passing Objects to Functions 54646
  - Wrong Overloading of Constructors/Destructors 54647
  - Error in Dynamic Initializations 54648
  - Other Methods/Functions 54649

C++/OOP Errors 54650
Wrong Overloading of Operators 54651
Errors using C++ KeyWords 54652
Errors using Unions 54653
Wrong use of :: operator 54654
Errors in using Polymorphism 54655
Other C++/OOP 54659
Other Classes/Unions 54690
Other Data Structures 5490

Control Structures (5501)

Sequential (55101)
Assignment 55110
Illegal Assignment 55111
Var/Expression not Type Compatible 55112
Division by Zero 55113
Floating Point Overflow 55114
Range Check Error 55115
Invalid Numeric Format 55116
Other Assignment 55119
Print/Scan (551201)
Wrong Arguments 551210
Invalid Format Specified 551220
Incorrect use of & operator 551230
Error Reading EOF EOLn 551240
Error in using cout, cin 551250
Other Print/Scan 551290
Expressions/Operations (55130)
Operand types don't match operator 55131
Cannot Evaluate Expression 55132
Invalid Floating Point operation 55133
Bad Combination of OpCode/Operands 55134
Arithmetic Expression Errors 55135
Boolean/Relational Expression Errors 55136
Logic Expression Errors 55137
Wrong Operator Precedence 55138
Other Expressions/Operations 55139
Other Sequential 55190

Selection (55201)
Wrong Boolean Expression in If-Then 55210
Wrong nesting of If-Then-Else 55220
Errors in IF-Then statement syntax 55230

Errors in SWITCH statement 55240
Errors in BREAKs 55250
Other Selection 55290

Iterative 5530

Incorrect Entering status in Loops 5531
Wrong Boolean Expression in Loops 5532
Error in Incrementing loop counters 5533
Errors in DO-While Syntax 5534
Errors in While-DO Syntax 5535
Errors in FOR loop Syntax 5536
Incorrect loop termination criteria 5537
Other Iterative 5539

Hierarchical (55401)

Recurssion Errors 55410
Wrong No of arguments passed 55420
Wrong Type of arguments passed 55430
Invalid Function Reference 55440
Invalid Function Declaration 55450
Incompatible function return type 55460
Stack/Heap Overflow Error 55470
Other Hierarchical 55490

Other Control Structures 5590

Deep Semantic Errors 560

(these involve modifications in multiple control structures, modules, etc)

Other Compilation/RunTime/Semantic 590

Configuration (601)

Out of Memory 610
Include Files not found 620
Wrong External Variable 630
Problems in Linking 640
Path for Include Files not found 650
Disk Read/Write error 660
Disk Seek Error 670
Problems with Processes 680
Other Configuration 690

Maintenance 7-[Above ErrorNo]

Dictionary of Abbreviations

Phases

1. Requirements and Specification (RS)
2. Design (D)
3. Code (C)
4. Code Review (CR)
5. Compile (CP)
6. Test (T)
7. PostMortem (PM)
8. Maintenance (M)

Activities

1. Problem Understanding (PU)
2. Domain Analysis (DA)
3. Building User Requirements (BR)
4. Requirements Review (RR)
4. Specifications (S)
5. Specifications Review (SR)
6. High Level Design [Key Modules] (HD)
7. Low Level Design [Objects, User Interface] (LD)
8. Pseudocode, FlowCharts (PC)
9. Design Review (DR)
10. Coding (C)
11. Code Review (CR)
12. Structured Walk Through (SW)
13. Compiling (CP)
14. Debugging (DB)
15. Test Case Generation (TG)
16. Testing (T)
17. PostMortem (PM)
18. Maintenance (M)

APPENDIX J

Sample Complexity Results

5/31/1994
PC-METRIC (C++) Version 4.05
Summary Complexity Report for: A:\STD1.RP2

Software Science Length (N): 229
Estimated Software Science Length (N^): 276
Software Science Volume (V): 1336
Software Science Effort (E): 69278

Estimated Errors using Software Science (B^): 0
Estimated Time to Develop, in hours (T^): 1

Cyclomatic Complexity (VG1): 6
Extended Cyclomatic Complexity (VG2): 6
Average Cyclomatic Complexity: 0
Average Extended Cyclomatic Complexity: 0
Average of Nesting Depth: 1
Average of Average Nesting Depth: 0

Lines of Code (LOC): 141
Physical Source Stmts (PSS): 114
Logical Source Stmts (LSS): 29
Nonexecutable Statements: 23
Compiler Directives: 5
Number of Comment Lines: 23
Number of Comment Words: 155
Number of Blank Lines: 27
Number of Procedures/Functions: 7

5/31/1994
PC-METRIC (C++) Version 4.05
Summary Complexity Report for: A:\STD2.RP2

Software Science Length (N): 684
Estimated Software Science Length (N^): 402
Software Science Volume (V): 4274
Software Science Effort (E): 282147

Estimated Errors using Software Science (B^): 1
Estimated Time to Develop, in hours (T^): 4

Cyclomatic Complexity (VG1): 27
Extended Cyclomatic Complexity (VG2): 42
Average Cyclomatic Complexity: 5
Average Extended Cyclomatic Complexity: 8
Average of Nesting Depth: 3
Average of Average Nesting Depth: 1

Lines of Code (LOC): 243

Physical Source Stmts (PSS): 215
Logical Source Stmts (LSS): 111
Nonexecutable Statements: 14
Compiler Directives: 4
Number of Comment Lines: 22
Number of Comment Words: 160
Number of Blank Lines: 28
Number of Procedures/Functions: 5

5/31/1994
PC-METRIC (C++) Version 4.05
Summary Complexity Report for: A:\STD3.RP2

Software Science Length (N): 1109
Estimated Software Science Length (N^): 546
Software Science Volume (V): 7319
Software Science Effort (E): 895684

Estimated Errors using Software Science (B^): 2
Estimated Time to Develop, in hours (T^): 14

Cyclomatic Complexity (VG1): 46
Extended Cyclomatic Complexity (VG2): 64
Average Cyclomatic Complexity: 6
Average Extended Cyclomatic Complexity: 9
Average of Nesting Depth: 3
Average of Average Nesting Depth: 2

Lines of Code (LOC): 366
Physical Source Stmts (PSS): 326
Logical Source Stmts (LSS): 174
Nonexecutable Statements: 29
Compiler Directives: 5
Number of Comment Lines: 27
Number of Comment Words: 208
Number of Blank Lines: 40
Number of Procedures/Functions: 7

5/31/1994
PC-METRIC (C++) Version 4.05
Summary Complexity Report for: A:\STD4.RP2

Software Science Length (N): 954
Estimated Software Science Length (N^): 660
Software Science Volume (V): 6506
Software Science Effort (E): 886479

Estimated Errors using Software Science (B^): 2
Estimated Time to Develop, in hours (T^): 14

PC-METRIC (C++) Version 4.05
Complexity Report by Function for: A:\C631004.CPP

Function	FT	N	N^	V	E	VG1	VG2	Dpth	AvgDph	PSS	LSS	Nonex	CCDir	Blk	Cmt	CmtWrd	SP	VL
PgmId::PgmId()		9	10	23	47	1	1	0	0	5	2	0	0	0	0	0	1	0
PgmId::~PgmId()		2	1	1	1	1	1	0	0	4	0	0	0	0	1	1	1	0
FileIO::FileIO()		2	1	1	1	1	1	0	0	4	0	0	0	0	1	1	1	0
FileIO::FileOpen(char)		56	103	266	3621	3	3	2	1	24	11	1	0	0	7	29	4	5
FileIO::FileRead(FILE,int)		31	76	136	1961	2	2	1	0	13	5	0	0	0	3	11	2	4
FileIO::FileRead2(FILE,int)		39	74	171	2784	2	2	1	0	13	5	0	0	0	3	13	2	5
FileIO::FileClose(FILE)		5	8	12	23	1	1	0	0	7	1	0	0	0	3	8	0	2
FileIO::~FileIO()		2	1	1	1	1	1	0	0	4	0	0	0	0	1	1	1	0
MeanStdDist::MeanStdDist()		2	1	1	1	1	1	0	0	4	0	0	0	0	1	0	1	0
MeanStdDist::CalcMean(int,int)		42	72	184	2548	2	2	1	0	15	6	1	0	0	3	10	2	6
MeanStdDist::CalcStdDev(int,int)		63	113	306	5420	2	2	1	1	17	8	1	0	0	3	9	2	7
MeanStdDist::CalcDist(int)		44	92	204	2605	2	2	1	1	12	4	0	0	0	3	7	0	9
MeanStdDist::DispObjSizesData(int,i		91	140	463	5185	3	3	1	0	23	13	2	0	3	4	19	1	8
MeanStdDist::~MeanStdDist()		2	1	1	1	1	1	0	0	4	0	0	0	0	1	1	1	0
SlopeIntercept::SlopeIntercept()		2	1	1	1	1	1	0	0	4	0	0	0	0	1	0	1	0
SlopeIntercept::CalcB1()		12	28	42	145	1	1	0	0	9	2	0	0	0	4	16	0	3
SlopeIntercept::CalcB2(int,int)		107	112	520	11644	2	2	1	1	25	12	1	0	0	6	22	6	5
SlopeIntercept::DispB1B2(int,int)		63	101	300	3549	2	2	1	0	17	9	1	0	0	3	8	0	5
SlopeIntercept::EstTotLOC()		36	73	158	1739	3	3	2	1	16	6	1	0	0	3	9	1	4
SlopeIntercept::~SlopeIntercept()		2	1	1	1	1	1	0	0	4	0	0	0	0	1	1	1	0
main()		141	169	745	28617	3	3	1	0	41	25	8	0	8	26	77	4	9

Number of functions in this file: 21

APPENDIX K

Sample Section from Spread Sheet

Attribute	Subject 1	Subject 2	Subject 3	Subject 4	Subject 5	Subject 6	Subject 7	Subject 8	Subject 9	Subject 10	Subject 11	Subject 12	Attribute	Sum	Mean	SumSq
Program No 1													Program No 1			
Program Size (LOC)													Program Size (LOC)			
Planned													Planned			
Planned Reused	0	0	0	0	0	0	0	0	0	0	0	0	Planned Reused	0	0	
Planned Del & Mod	0	0	0	0	0	0	0	0	0	0	0	0	Planned Del & Mod	0	0	
Planned Modified (M)	0	0	0	0	0	0	0	0	0	0	0	0	Planned Modified (M)	0	0	
Planned Added (A)	100	75	60	100	70	100	80	80	500	45	200	150	Planned Added (A)	1560	130	
Planned Total LOC	100	75	60	100	70	100	80	80	500	45	200	150	Planned Total LOC	1560	130	
Planned Tot New & Chgd	100	75	60	100	70	100	80	80	500	45	200	150	Planned Tot New & Chgd	1560	130	
Planned Tot New Reuse	0	0	0	0	0	0	0	0	0	0	0	0	Planned Tot New Reuse	0	0	
Actual													Actual			
Actual Reused	0	0	0	0	0	0	0	0	0	0	0	0	Actual Reused	0	0	
Actual Del & Mod	0	0	0	0	0	0	0	0	0	0	0	0	Actual Del & Mod	0	0	
Actual Modified (M)	0	0	0	0	0	0	0	0	0	0	0	0	Actual Modified (M)	0	0	
Actual Added (A)	81	115	57	53	59	68	94	94	193	43	94	161	Actual Added (A)	1112	92.6666667	
Actual Total LOC	81	115	57	53	59	68	94	94	193	43	94	161	Actual Total LOC	1112	92.6666667	
Actual Tot New & Chgd	81	115	57	53	59	68	94	94	193	43	94	161	Actual Tot New & Chgd	1112	92.6666667	
Actual Tot New Reuse	0	0	0	0	0	0	0	0	0	0	0	0	Actual Tot New Reuse	0	0	
Productivity (LOC/hr)													Productivity (LOC/hr)			
Guessed Productivity													Guessed Productivity	0	0	
Estimated Productivity	23.5294118	47.3684211	32.7272727	75	26.25	27.2727273	14.5454545	36.9230769	93.75	20	52.173913	17.6470588	Estimated Productivity	467.187336	38.932278	
Actual Productivity	8.42376238	92	25.9223881	42.972973	12.918708	19.902439	16.4431487	45.483871	42.3945946	20.64	53.7142857	12.627451	Actual Productivity	414.444621	34.5370518	7031.94382
Total Productivity	8.83636364	76.6666667	23.75	33.8297872	11.9594595	19.1549296	14.6875	36.6233766	56.4878049	19.1111111	43.0534351	11.1034483	Total Productivity	355.263883	29.6053235	
Null Productivity	10.125	100	27.1428571	43.5616438	13.3082707	21.7021277	24.3163448	50.3571429	64.3333333	23.4545455	55.8415842	13.1607629	Null Productivity	447.297613	37.2748011	
Reuse Productivity	19.6588235	72.6315789	31.8909091	39.75	22.125	18.5454545	17.0909091	43.3846154	36.1875	19.1111111	24.5217391	18.9411765	Reuse Productivity	362.438817	30.2032348	
Estimation Error													Estimation Error			
Size	23.4567901	34.7826087	5.26315789	88.6792453	18.6440678	47.0588235	14.893617	14.893617	159.067358	4.65116279	112.765957	6.83229814	Size	530.988703	44.2490586	27748.6828
Time	42.7272727	18.6666667	20.1388889	17.0212766	39.1891892	12.6760563	1.5625	14.2857143	75.6097561	7.40740741	98.4732824	27.5862069	Time	373.344218	31.1120181	9315.89331
Productivity	144.492059	48.5125858	28.2296651	74.5283019	103.177966	37.8320856	11.5409413	18.8216839	49.7733161	3.10077519	2.86771508	39.7515528	Productivity	561.829368	46.819114	19858.6075
Guessed Productivity	100	100	100	100	100	100	100	100	100	100	100	100	Guessed Productivity	1200	100	0
Time (min)													Time (min)			
Planned													Planned			
Planned Planning Time	45	10	10	5	5	20	60	10	20	10	5	60	Planned Planning Time	260	21.6666667	
Planned Design Time	60	5	10	20	20	20	60	20	20	15	15	150	Planned Design Time	415	34.5833333	
Planned Code Time	80	60	50	20	45	60	30	50	180	60	120	60	Planned Code Time	815	67.9166667	
Planned CR Time	0	0	0	0	0	0	0	0	0	0	0	0	Planned CR Time	0	0	
Planned Compile Time	60	10	30	5	40	60	60	20	40	20	30	120	Planned Compile Time	495	41.25	
Planned Test Time	30	10	10	30	50	40	120	30	60	30	60	120	Planned Test Time	590	49.1666667	
Planned PM Time	60	10	5	30	20	20	60	2	40	10	30	120	Planned PM Time	407	33.9166667	
Planned Total Time	315	105	115	110	180	240	390	132	360	145	260	630	Planned Total Time	2982	248.5	
Actual													Actual			
Actual Planning Time	25	6	8	1	8	17	111	12	5	15	4	31	Actual Planning Time	243	20.25	
Actual Design Time	90	4	13	7	22	18	57	13	10	15	8	73	Actual Design Time	330	27.5	
Actual Code Time	180	40	61	34	88	82	43	46	120	73	41	167	Actual Code Time	977	81.4166667	
Actual CR Time	0	0	0	0	0	0	0	0	0	0	0	0	Actual CR Time	0	0	
Actual Compile Time	90	19	32	4	52	63	88	46	20	19	9	379	Actual Compile Time	813	67.75	
Actual Test Time	120	10	20	26	104	25	44	7	30	5	43	115	Actual Test Time	551	45.9166667	
Actual PM Time	45	15	10	20	22	8	41	30	20	10	26	105	Actual PM Time	352	29.3333333	
Actual Total Time	550	90	144	94	296	213	384	154	205	135	131	870	Actual Total Time	3266	272.166667	
Appraisal%	0	0	0	0	0	0	0	0	0	0	0	0	Appraisal%	0	0	
A/F Ratio	0	0	0	0	0	0	0	0	0	0	0	0	A/F Ratio	0	0	
Defects													Defects			
Planned Injected													Planned Injected			
Planning Pln Defs Inj													Planning Pln Defs Inj	0	0	
Design Pln Defs Inj													Design Pln Defs Inj	0	0	
Code Pln Defs Inj													Code Pln Defs Inj	0	0	
CR Pln Defs Inj													CR Pln Defs Inj	0	0	
Compile Pln Defs Inj													Compile Pln Defs Inj	0	0	
Test Pln Defs Inj													Test Pln Defs Inj	0	0	
Total Pln Defs Inj	0	0	0	0	0	0	0	0	0	0	0	0	Total Pln Defs Inj	0	0	
Actual Injected													Actual Injected			
Planning Act Defs Inj	0	0	0	0	0	0	0	0	0	0	0	3	Planning Act Defs Inj	3	0.25	

Design Act Defe Inj	5	0	0	1	2	0	1	0	0	0	3	8	Design Act Defe Inj	20	1.66666667	
Code Act Defe Inj	8	0	8	4	7	14	7	7	10	5	6	36	Code Act Defe Inj	112	9.33333333	
CR Act Defe Inj	0	0	0	0	0	0	0	0	0	0	0	0	CR Act Defe Inj	0	0	
Compile Act Defe Inj	3	3	0	0	0	2	0	1	0	0	0	0	Compile Act Defe Inj	9	0.75	
Test Act Defe Inj	0	1	0	0	0	1	0	0	4	0	4	4	Test Act Defe Inj	14	1.16666667	
Total Act Defe Inj	16	4	8	5	9	17	8	8	14	5	13	51	Total Act Defe Inj	158	13.1666667	
Planed Removed													Planed Removed			
Planning Pln Defe Rem													Planning Pln Defe Rem	0	0	
Design Pln Defe Rem													Design Pln Defe Rem	0	0	
Code Pln Defe Rem													Code Pln Defe Rem	0	0	
CR Pln Defe Rem													CR Pln Defe Rem	0	0	
Compile Pln Defe Rem													Compile Pln Defe Rem	0	0	
Test Pln Defe Rem													Test Pln Defe Rem	0	0	
Total Pln Defe Rem	0	0	0	0	0	0	0	0	0	0	0	0	Total Pln Defe Rem	0	0	
Actual Removed													Actual Removed			
Planning Act Defe Rem	0	0	0	0	0	0	0	0	0	0	0	3	Planning Act Defe Rem	3	0.25	
Design Act Defe Rem	0	0	0	0	0	0	0	0	0	0	0	5	Design Act Defe Rem	5	0.41666667	
Code Act Defe Rem	0	0	0	0	0	0	0	0	0	0	2	23	Code Act Defe Rem	25	2.08333333	
CR Act Defe Rem	0	0	0	0	0	0	0	0	0	0	0	0	CR Act Defe Rem	0	0	
Compile Act Defe Rem	15	3	6	1	8	16	6	8	8	5	2	13	Compile Act Defe Rem	91	7.58333333	
Test Act Defe Rem	1	1	2	4	1	1	2	0	6	0	9	7	Test Act Defe Rem	34	2.83333333	
Total Act Defe Rem	16	4	8	5	9	17	8	8	14	5	13	51	Total Act Defe Rem	158	13.1666667	
Total Fix Time													Total Fix Time			
Defect Removal Efficiency(Def/hr)	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	#DIV/0!	Defect Removal Efficiency(Def/hr)	#DIV/0!	#DIV/0!	
Total Act Defe Inj / KLOC	197.530864	34.7826087	140.350877	94.3396226	182.542373	250	85.106383	85.106383	72.5388601	116.27907	138.297872	316.770186	Total Act Defe Inj / KLOC	1683.6451	140.303758	
CR Defe Rem / KLOC	0	0	0	0	0	0	0	0	0	0	0	0	CR Defe Rem / KLOC	0	0	
Net Defe Rem / KLOC	197.530864	34.7826087	140.350877	94.3396226	182.542373	250	85.106383	85.106383	72.5388601	116.27907	138.297872	316.770186	Net Defe Rem / KLOC	1683.6451	140.303758	71112.8992
Cyclomatic Complexity	6	6	9	6	6	13	10	6	10	5	7	10	Cyclomatic Complexity	89	7.41666667	
Attribute	Subject 1	Subject 2	Subject 3	Subject 4	Subject 5	Subject 6	Subject 7	Subject 8	Subject 9	Subject 10	Subject 11	Subject 12	Attribute	Sum	Mean	SumSq
Program No 2													Program No 2			
Program Size (LOC)													Program Size (LOC)			
Planed													Planed			
Planed Reused	0	0	0	0	20	0	140	0	30	0	0	0	Planed Reused	190	15.8333333	
Planed Del & Mod	0	0	0	0	10	0	120	0	20	0	0	0	Planed Del & Mod	150	12.5	
Planed Modified(M)	0	0	0	0	10	0	0	0	5	0	0	0	Planed Modified(M)	15	1.25	
Planed Added (A)	200	80	50	100	40	120	80	50	225	140	150	230	Planed Added (A)	1485	123.75	
Planed Total LOC	200	80	50	100	60	120	100	50	240	140	150	250	Planed Total LOC	1540	128.333333	
Planed Tot New & Chgd	200	80	50	100	50	120	86	50	230	140	150	230	Planed Tot New & Chgd	1500	125	
Planed Tot New Reuse	0	0	0	0	10	0	20	0	10	0	0	0	Planed Tot New Reuse	40	3.33333333	
Actual													Actual			
Actual Reused	0	0	0	0	15	0	112	0	20	0	0	0	Actual Reused	147	12.25	
Actual Del & Mod	0	0	0	0	9	0	92	0	15	0	0	0	Actual Del & Mod	116	9.66666667	
Actual Modified(M)	0	0	0	0	2	0	0	0	5	0	0	0	Actual Modified(M)	7	0.58333333	
Actual Added (A)	193	41	62	73	88	177	93	52	160	140	97	176	Actual Added (A)	1372	114.333333	
Actual Total LOC	193	41	62	73	106	177	113	52	170	140	97	176	Actual Total LOC	1410	117.5	
Actual Tot New & Chgd	193	41	62	73	100	177	93	52	165	140	97	176	Actual Tot New & Chgd	1379	114.916667	
Actual Tot New Reuse	0	0	0	0	4	0	20	0	5	0	0	0	Actual Tot New Reuse	31	2.58333333	
Productivity (LOC/hr)													Productivity (LOC/hr)			
Guessed Productivity													Guessed Productivity	0	0	
Estimated Productivity	38.7692308	27.3255814	33.3333333	50	20	38.9189189	13.3333333	22.2222222	78.0571429	35.7446809	28.5714286	16.1298323	Estimated Productivity	390.264905	32.5170754	
Actual Productivity	26.0224719	11.3364055	29.5744681	33.15625	27.3972603	30	14.4835165	34.2857143	59.6369542	31.7142857	24.4537815	17.9124378	Actual Productivity	331.591138	25.2987613	1810.18230
Total Productivity	21.8448598	10.1852093	32.6315789	29.8613245	24.291450	28.3957219	12.7801631	21.5172414	51.0324607	30.1016949	19.3335467	14.6858031	Total Productivity	297.121152	24.7600993	
Null Productivity	28.2439024	14.6571429	42.7586207	37.8151261	29.8564502	30.872093	17.6532278	40.5194805	63.4615385	34.0229885	24.7635574	19.027027	Null Productivity	382.758755	31.8965629	
Reuse Productivity	29.6923077	11.4410685	41.3333333	37.5	42.4	57.4054054	18.8333333	23.1111111	58.2857143	37.787234	18.4761905	11.3548387	Reuse Productivity	387.621329	32.3017774	
Estimation Error													Estimation Error			
Size	3.67694301	85.1219512	18.3548387	33.3333333	43.3962264	32.2033898	11.5044248	3.84615385	41.1764706	5.40540541	54.6391753	42.0454545	Size	381.653767	32.1378139	7898.46733
Time	4.67289072	1.23966942	7.89473684	8.64229166	27.1253061	42.513369	10.5564587	13.7531634	7.32964283	15.2142373	34.551499	42.461964	Time	208.049332	17.337441	7647.07162
Productivity	18.2469891	96.9376391	15.7706093	42.2222222	27	29.7297297	8.00477897	35.1051852	32.2251082	12.7684531	18.838488	7.89956812	Productivity	342.762173	28.5635144	8431.79818
Guessed Productivity	100	100	100	100	100	100	100	100	100	100	100	100	Guessed Productivity	1200	100	0
Time (min)													Time (min)			
Planed													Planed			
Planed Planning Time	30	66	5	28	28	15	96	15	16	20	15	46	Planed Planning Time	366	30	
Planed Design Time	60	45	70	20	40	70	60	30	15	35	20	190	Planed Design Time	555	46.25	

APPENDIX L

Calculations for Personal Capability Index

Personal Capability Index (Scale: 0 to 100)

This index is divided into two categories:

- Experience (60)
- Performance (40)

The details are as follows:

(i) Experience (60)

1. Total Experience in Programming (15)

- 0.1 point/month for each programming language

e.g, Experience with C++ for 2 years and Pascal for 6 months would yield 2.4 + 0.6 = 3.0 points

2. Diversity in Programming Languages (10)

- 1 point/programming language in which subject has over 1 semester of programming experience
- 0.5 points/programming language in which subject has less than 1 semester of programming experience

e.g, an experience of 2 years of C, 1 year of Pascal and 2 months of Ada would give 1 + 1 + 0.5 = 2.5 points

3. Level of Education(10)

- 3 points for each completed B.S or M.S degree
- 2 points for each degree which has not been completed yet but the subject is enrolled in
- 1 point for each additional Major/Minor besides Computer Science

e.g, a subject enrolled in the M.S Computer Science program, with a completed B.S in Computer Science and a minor in mathematics would get 2 + 3 + 1 = 6 points

4. Job experience in computer software/hardware related field (15)

- 2 points/year for full time experience
- 1 point/year for part time experience

e.g, 2 years of full time and 6 months of part time experience would yield 4 + 0.5 = 4.5 points

5. Experience with Software Packages (5)

- 0.5 points for familiarity with each software package related to Databases, Spreadsheets, Graphing tools, etc.

e.g, a subject with some working experience of Excel, Cricket Graph and DBase IV would get 1.5 points.

6. Experience in Software Engineering and Object-Oriented Design (5)

- 1 point/course in software engineering or OO-Design
- 1 point/tool used related to software engineering or OO-Design

(ii) Performance (40)

7. Productivity (15)

- 1 point for every 5 LOC/hr value of productivity. This productivity is the average of the productivities in the 9 projects.

e.g, a subject with a productivity of 45 LOC/hr gets 9 points

8. Defects/KLOC (10)

- 0 points for a defect-rate of 100 defects / KLOC or more
- 1 point for a defect-rate of every 5 defects/KLOC below the 100 / KLOC level

e.g, a defect-rate of 85 / KLOC means 15 /KLOC below the base level, yielding 3 points. The defect rate is the average of the 9 defect rates for the projects.

9. Defect Removal Rate (5)

- 1 point for each 6 defects removed/hour value.

e.g, a defect removal rate of 15 defects removed per hour gives 2.5 points. This rate is calculated from the averages of the nine projects.

10. Grade assigned to the subject in the PSP course (10)

This is the letter grade assigned to the subject in the PSP course, transformed to a scale of 0 - 10.

VITA

Khalid Sherdil

Education:

M.S. in Computer Science (expected Feb. 1995)
McGill University, Montreal, Canada

B.S. in Electrical/Computer Engineering, 1993
Washington University in St. Louis, MO, USA

B.S. in Computer Science (Honors), 1993
Washington University in St. Louis, MO, USA

B.A. in Physics, 1991
College of Wooster, OH, USA

Honors/Awards:

Tau Beta Pi
Eta Kappa Nu
Golden Key National Scholastic Society
Elliot Honors Student, Washington Univ., '93
Graduated with Distinction, Washington Univ., '93
Academic Achievement Award, Coll. of Wooster

Societies:

A.C.M
I.E.E.E
American Physical Society

BIBLIOGRAPHY

[1] Adler, P. "Shared Learning," Management Science, vol. 36 (Aug. 1990), no 8, pp 938-957

[2] Adler, P. and Clark, K., "Behind the Learning Curve: A Sketch of the Learning Process," Management Science, vol. 37, no 3 (May 1991), pp 267-281

[3] Albrecht, A., Gaffney J. "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," IEEE Transactions on Software Engineering, vol. SE-9, no 6, Nov. 1983, pp 639-648

[4] Argote, L., Beckman, S. and Epple, D. "The Persistence and Transfer of Learning in Industrial Settings," Management Science, vol. 36, no 2, (Feb. 1992), pp 140-154

[5] Arrow, K. "The Economic Implications of Learning by Doing," Review of Economic Studies, vol. 29 (April 1962a), pp 166-170

[6] Baker, A., et al.. "A Philosophy for Software Measurement," J. Systems Software, vol. 12 (1990), pp 277-281

[7] Baloff, N. "Extension of the Learning Curve - Some Empirical Results," Operations Research Quarterly, vol. 22, no 4, 1971, pp 329-340

[8] Barnes, B., Bollinger T. "Making Reuse Cost-Effective," IEEE Software, Jan 1991, pp 13-24

[9] Basili, V. "Quantitative Evaluation of Software Methodology," Technical Report TR-1519, Dept. of CS, University of Maryland, July 1985

[10] Basili, V. and Weiss, D. "A Methodology for Collecting Valid Software Engineering Data," IEEE Transactions on Software Engineering, vol. SE-10, no 6 (Nov. 1984) pp 728-738

[11] Basili, V., Rombach H., "Goal Question Metric Paradigm," Encyclopedia of Software Engineering, vol. 2, 1994, John Wiley & Sons, Inc.

[12] Basili, V., Rombach H., "Measurement," Encyclopedia of Software Engineering, 1994, John Wiley & Sons, Inc.

[13] Basili, V., Selby R. "Comparing the Effectiveness of Software Testing Strategies," IEEE Transactions on Software Engineering, vol. SE-13, no 12, Dec. 1987, pp 1278-1296

[14] Basili, V., Selby R., Phillips T. "Metric Analysis and Data Validation Across Fortran Projects," IEEE Transactions on Software Engineering, vol. SE-9, no 6, Nov. 1983, pp 652-663

[15] Basili, V., Selby, R. and Hutchens, D. "Experimentation in Software Engineering," IEEE Transactions on Software Engineering, vol. SE-12, no 7 (July 1986), pp 733-743

[16] Boehm, B. "Software Engineering Economics," 1981, Prentice-Hall, Englewood Cliffs, NJ

[17] Card, D., Agresti W. "Measuring Software Design Complexity," The Journal of Systems and Software, vol. 8, 1988, pp 185-197

[18] Chillarege, R., et al. "Orthogonal Defect Classification - A Concept for In-Process Measurements," IEEE Transactions on Software Engineering, vol. SE-18, no 11, Nov. 1992, pp 943-955

[19] Coulter, N. "Software Science and Cognitive Psychology," IEEE Transactions on Software Engineering, vol. SE-9, no 2, Mar 1983, pp 166-171

[20] Curtis, B., et al. "Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics," IEEE Transactions on Software Engineering, vol. SE-5, no 2, Mar 1979, pp 96-104

[21] Curtis, B., et al. "Productivity Factors and Programming Environments," Proceedings of the Seventh International Conference on Software Engineering, Washington DC, IEEE Computer Society, pp 143-152

[22] Curtis, B., Sheppard S., Milliman P. "Third Time Charm: Stronger Prediction of Programmer Performance by Software Complexity Metrics," Proceedings of the 4th International Conference on Software Engineering, 1979, pp 356-360

[23] Dunham, J., Kruesi, E. "The Measurement Task Area," IEEE Computer, Nov. 1983, pp 47-54

[24] Dutton, J., Thomas, A. and Butler, J. "The History of Progress Functions as a Managerial Technology," Business History Review, vol. 58 (Summer 1984), pp 204-233

[25] Emam, K., Moukheiber, N. and Madhavji, N. "The Empirical Evaluation of the G/Q/M," Proceedings CASCON 1993, vol. 2, pp 265-289

[26] Fagan, M. "Advances in Software Inspections," IEEE Transactions on Software Engineering, vol. SE-12, no 7, Jul. 1986, pp 744-751

[27] Gill, G., Kemerer C. "Cyclomatic Complexity Density and Software Maintenance Productivity," IEEE Transactions on Software Engineering, vol. SE-17, no 12, Dec. 1991, pp 1284-1288

[28] Herbsleb, J., et al., "Benefits of CMM-Based Software Process Improvement: Initial Results," Technical Report, CMU/SEI-94-13, Aug. 1994, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA 15213

[29] Hirsch, W. "Manufacturing Progress Functions," The Review of Economics and Statistics, vol. 34 (May 1952), pp 143-155

[30] Humphrey, W. "Managing the Software Process," Reading, MA:Addison-Wesley, 1989

[31] Humphrey, W. "A Discipline for Software Engineering," (currently being published)

[32] Irwin, P. H and Laugham, F. W. "The Change Seekers," Harvard Business Review, vol. 35 (Jan-Feb. 1966), pp 81-92

[33] Jones, T. "Measuring programming quality and productivity," IBM Systems Journal, Vol. 17, No 1, 1978, pp 39-63

[34] Jones, T. "Reusability in Programming: A Survey of the State of the Art," IEEE Transactions on Software Engineering, Vol. SE-10, No 5, Sept. 84, pp 488-494

[35] Kemerer, C. F. "How the Learning Curve Affects Case Tool Adoption," IEEE Software, May 1992, pp 23-28

[36] Kemerer, C., Porter B. "Improving the Reliability of Function Point Measurement: An Empirical Study," IEEE Transactions on Software Engineering, Vol. 18, No 11, Nov. 92, pp 1011-1024

[37] Kerlinger, F. "Foundations of Behavioral Research," 3rd ed., 1986, Holt, Rinehart and Winston, New York, NY

[38] Kidder, L. "Research Methods in Social Relations," 5th ed., 1986, Hold, Rinehart and Winston, NY.

[39] Kleinbaum, D. "Applied Regression Analysis and other Multivariable Methods," 2nd ed., 1988, PWS-Kent Pub. Co., Boston, MA

[40] Koffman, E. "Turbo Pascal," 3rd ed., Addison-Wesley Publishing Company, Inc., Reading MA, p AP-42

[41] Leblanc, R., Fischer C., "A Case Study of Run-Time Errors in Pascal Programs," Software-Practice and Experience, Vol. 12, 1982, pp 825-834

[42] Levy, F. K. "Adaptation in the Production Process," Management Science, vol. 11, no 6, (April 1965), pp B136-B154

[43] McCabe, T. "A Complexity Measure," IEEE Transactions on Software Engineering, Vol. SE-2, No 4, Dec. 76, pp 308-320

[44] PC-Metric, SET Laboratories, P.O. Box 868, Mulino, OR 97042

[45] Putnam, P. and Myers, W. "Measures for Excellence," Englewood Cliffs, NJ: Yourdon Press, 1992

[46] Rapping, L. "Learning and World War II Production Functions," Review of Economics and Statistics, vol. 47, 1965, pp 81-86

[47] Remus, H., Zilles, S. "Prediction and Management of Program Quality," Proceedings of the 4th International Conference on Software Engineering, Munich, Germany, 1979, p341-350.

[48] Ripley, D., Druseikis F. "A Statistical Analysis of Syntax Errors," Computer Languages, Vol. 3, 1978, pp 227-240

[49] Rosenthal, R. "Essentials of Behavioral Research: Methods and Data Analysis," 1984, McGraw-Hill, New York, NY

[50] Russell, G. "Experience with Inspection in Ultra-Scale Developments," IEEE Software, Jan 91, pp 25-31

[51] Sheil, B "The Psychological Study of Programming," Computing Surveys, vol. 13, no 1, Mar 1981, pp 101-120

[52] Shepperd, M. "An Evaluation of Software Product Metrics," Information and Software Technology, Vol. 30, No 3, April 1988, pp 177-188

[53] Sherdil, K., Madhavji N. "Personal Progress Functions in the Software Process," Proceedings of Ninth International Software Process Workshop

[54] Standish, T. "An Essay on Software Reuse," IEEE Transactions on Software Engineering, Vol. SE-10, No 5, Sept. 84, pp 494-497

[55] Strait, P. "Probability and Statistics with Applications," Harcourt Brace Jovanovich, New York, NY

[56] Straub, D. "Validating Instruments in MIS Research," MIS Quarterly, June 1989

[57] Sunohara, T., et al. "Program Complexity Measure for Software Development Management," Proceedings of 5th International Conference on Software Engineering, 1981, pp 100-106

[58] Turban, E. "Incentives during Learning - an Application of the Learning Curve Theory and a Survey of Other Methods," Journal of Industrial Engineering, vol. 19, no 12, 1968, pp 600-607

[59] Walrad, C. and Moss, E. "Measurement: Key to Applications Development and Quality," IBM Systems Journal, vol. 32 (1993), no. 3, pp 445-461

[60] Yelle, L. "The Learning Curve: Historical Review and Comprehensive Survey," Decision Sciences, vol. 10 (Feb. 1979), pp 302-328

[61] Zultner, R. "The Deming Approach to Software Quality," Quality Progress, v 21, no 11, pp 58-64