0/21/77

[redacted]

This presentation was made in October 1979 at Monterey, Calif., at a Joint Share, Guide, IB Conference on Application Development. Regards [illegible]

Measuring Application Development Productivity

by Allan J. Albrecht
IBM Corporation
White Plains, New York

In this paper I would like to share with you some experiences in measuring application development project productivity in IBM's DP Services organization. I have several objectives in this paper:

- to describe our productivity measure, which has been effective in measuring productivity over all phases of a project including the design phase, and has enabled us to compare the results of projects that use different programming languages and technologies.
- to show how we used that measure to determine the productivity trend in our organization.
- to identify some factors that affected productivity and to show how we determined their relative importance.

At this point I shall describe the organization so you can consider the subject in the context of our management objectives.

The DP Services organization consists of about 450 people engaged in application development for IBM's customers under contract. Both customers and Services people are located throughout the United States. At any given time there are about 150-170 contracts under way. Projects cover all industries. They address the spectrum of data processing functional requirements: order entry and control, insurance claim processing, hospital patient information systems, data communication control systems, etc. The average contract size is 2 or 3 people. A number of projects each year require 15 to 20 people, and several require up to 35 or 40 Services people and customer people, working under our project manager, to develop the application.

Each project is done under the DP Services Application Development Process (Figure 1). Let me not give the impression that on each of these hundreds of contracts we perform complete projects. On the majority of the contracts we perform only specific tasks covering part of the function provided by the project. Our approach is a phased approach to developing applications. The phased approach consists of a system design followed by the system implementation phase. System design is completed and approved before system implementation is committed and started. This graphic represents the distribution of development effort in those phases. It is based on DP Services experience on projects over the last 3 or 4 years.

Our experience shows that the design phase takes about 20% of the work-hours and the implementation phase takes about 80% of the work-hours on a completed project. This distribution has been quite consistent. Our latest projects show the same distribution of work effort as our earliest.

Later I will show you a model of the percentage of effort for each task on this graph. I will also show that we had a significant increase in project productivity during that time. Since the shape of the distribution did not change, the conclusion is that we have been increasing the productivity of the system design phase at the same rate as the implementation phase.

The following disciplined management techniques have helped achieve that productivity improvement.

Before the design project begins we make sure that the project objectives are completely described and approved, including functions to be provided, a statement of work to be accomplished, and estimated schedule and cost of the project.

The system design phase starts with requirements definition, where business requirements are defined for the customer's approval. During external and internal system design we provide the customer with a design of the system they will be getting if they approve. We provide a blueprint and a proposal for further development and implementation. System implementation follows with program development ending with a customer-approved system test and demonstration.

That is the phased approach. Each project goes through those basic phases.

To perform each project we use a project management system (Figure 2). Figure 2 shows the fundamental processes of our project management system.

After the need is identified with the customer, project objectives are completely documented. Using productivity feedback from previous jobs, and a complete task schedule of the project to be done, we estimate the project. Then we go through the first of the feedback loops where we assess the project risk, with a 28-question structured questionnaire. We have an independent systems assurance group review the entire project objectives document to make sure it is accurate, it is not misleading, and it contains all information that must be agreed upon. After we and the customer agree on their objectives, and that we plan to deliver these objectives, we start the project.

The project is then planned in detail. We ask that the project have an average task size of 20-40 work hours. We track the results against the plan once a week, and report the project status to IBM management and to the customer. We continue project reviews by having the systems assurance group review the project about every six months. Above all we control change—not to prevent change, but to make sure each party understands the value and cost, and approves the change, before it is implemented.

A key element of the project management system is, as we complete larger projects over 1000 work-hours, we document the completed project. How many hours were spent on the various tasks? What was delivered? What were the helpful actions or detrimental events? Under what system and environment did development take place? This document is then analyzed along with other project completion reports to update the standards we use to guide future projects. We measure application development productivity via the feedback loop at the top of Figure 2, the documented completed project results.

Now that I have discussed our Application Development Process, how do we measure the success of an application development project? There are three basic criteria: projects should be finished on time, within budget, and satisfy the customer. The customer should have stipulated the desired functional objectives and benefit/cost objectives at the outset; and if the project met those objectives within the schedule and budget, the customer should be satisfied.

Why measure productivity at all? To answer some questions for management and our customers. Are we doing as well as we can? Are we competitive? Are we improving? Productivity measurement is meant to provide some answers. Productivity measurement helps identify and specifically promote those things that improve productivity while avoiding things that hurt productivity. We must continue to identify and select development systems and technologies that deliver more application function with less effort and at lower cost.

We have learned there are things to watch out for in measuring productivity. To start with, you must measure the whole process, including the design phase. Costs can be incurred in the system design phase as well. Next, there are tasks and functions such as project management and system architecture that must be included because they contribute significantly to productivity results. To ignore them gives a false picture of the true costs. Finally, all measures are relative. You can measure contemporary projects against one another and you can measure time trends within your organization, but comparisons between organizations must be handled carefully unless they are using the same definitions.

Productivity measurement can do harm. To avoid this, keep the major project objectives in perspective—on time, within budget, a satisfied customer. Don't permit productivity measurements to divert your attention from those major project objectives.

Productivity measurement avoids a dependency on measures such as lines-of-code that can have widely differing values depending on the technology used. We track lines-of-code, but only as a secondary measure to compare projects using the same language.

The projects that were analyzed were 22 complete DP Services application development projects. 16 were written in COBOL, 4 were written using PL/1, and 2 used DMS/VS.

(Although we still use an incidental amount of Assembler (ALC) coding for specialized function on some of our projects, we no longer do entire projects using ALC, because it is not productive enough. Consequently, we have no ALC projects in our measurements. For those who might have the data, the method described should work well in measuring relative functional productivity on ALC projects.)

Project completion dates ranged from mid 1974 to early 1979. The projects ranged in size from 500 work-hours to 105,000 work-hours.
Twenty-two projects may seem to be a small number, considering that DP Services probably completed about 1500 contracts during that time. But, these 22 were all of the projects that passed the selection criteria in that time period.

These were the selection ground rules:

1. Only complete projects that had proceeded through all phases from requirements definition to final system test and demonstration, and had delivered a product to our customer, would be analyzed.

2. The whole project had to be done under our project management with consistent task definitions and management procedures.

3. All work-hours spent by our people and the customer's people had to be known and accounted for carefully.

4. The functional factors had to be known.

We found that about 3 to 7 projects meeting those criteria are completed per year.

To measure productivity we had to define and measure a product and a cost. The product that was analyzed was function value delivered. The number of inputs, inquiries, outputs, and master files delivered were counted, weighted, summed, and adjusted for complexity for each project. The objective was to develop a relative measure of function value delivered to the user that was independent of the particular technology or approach used.

The basis for this method was developed over the last 5 years from the DP Services project estimating experience. As part of that estimating we validated each estimate with a series of weighted questions about the application function and the development environment. We found that the basic value of the application function was consistently proportional to a weighted count of the number of external user inputs, outputs, inquiries and master files.

The general approach is to count the number of external user inputs, inquiries, outputs, and master files delivered by the development project. These factors are the outward manifestations of any application. They cover all the functions in an application.

These counts are weighted by numbers designed to reflect the function value to the customer. The weights used were determined by debate and trial. These weights have given us good results:

- Number of Inputs X 4
- Number of Outputs X 5
- Number of Inquiries X 4
- Number of Master Files X 10

Then we adjust that result for the effect of other factors.

If the inputs, outputs, or files are extra complicated, we add 5%. Complex internal processing can add another 5%. On-line functions and performance are addressed in other questions. The maximum adjustment possible is 50%, expressed as ± 25% so that the weighted summation is the average complexity.

This gives us a dimensionless number defined in function points which we have found to be an effective relative measure of function value delivered to our customer. These definitions are all shown in the Function Value Worksheet (Figure 3).

A recent article by Trevor Crossman in the May 1979 Datamation described a similar functional approach to measuring programmer productivity. His approach defined function based on program structure. Our approach defines function based on external attributes. He concentrated on programmer tasks of design, code, and unit test. As you will see we looked at the whole application development cycle from system design through system test. Both our views on the necessity of a functional approach appear to agree.

I believe that DP Services approach, based on external attributes, will be more effective in determining or proving the productivity advantages of other higher level languages and development technologies.

The cost used was the work-hours contributed by both IBM and the customer people working on the project, through all phases (design and implementation) (Figure 4). These are the elements of cost covered in the analysis. The percentage distributions in the current model have been developed from DP Services project experience over the last 3 or 4 years. The intent is to state the development cost in terms of work-hours used to design, program, and test the application project.

Figure 5 is our productivity time trend from 1974 through 1978. We have plotted hours worked per function point delivered. This means that down is good; down means that fewer hours were spent generating each function point. There are three types of projects plotted here. Each red dot represents a COBOL project, each blue dot represents a PL/1 project, and each white dot represents a DMS/VS project. The line is a linear least squares fit to all of the points.

The productivity improvement shown between 1974 and 1978 is about 3 to 1. PL/1, DMS/VS, and COBOL projects combine to show a significant trend toward improving productivity. Because three languages are represented, we believe we have a measurement that can analyze the relative productivity of different languages and different technologies. The DMS/VS projects, in particular, have continued the increasing productivity trend even though we are still very early on the learning curve with DMS.

Figure 6 shows the relationship between project size and productivity. It is derived from the same projects. It shows that more work hours are required to produce each function point as the projects get larger. This effect has been known for a long time. This chart simply quantifies the result for the 22 projects we analyzed. It shows that project size must be considered in any productivity comparisons. Furthermore, the two curves show that for the whole range of project sizes, on the average, PL/1 is probably about 25% more productive than COBOL. The initial indication on DMS/VS is that it is probably more productive than either PL/1 or COBOL. If this trend is confirmed, DMS/VS could prove to be about 30% more productive than COBOL.

The promise of being able to analyze the relative productivity of techniques as different as COBOL and DMS/VS is the greatest potential of this functional approach. It can provide the means to prove the efficiency of higher level languages and approaches such as DMS/VS and ADF.

Figure 7 uses the same data but removes the effect of project size on our productivity trend line.

Remember that down still means that fewer hours were spent to produce each function point. Down is good. With the project size effect removed, there is still a significantly improving productivity trend from 1974 to 1978.

Figure 7 also allows us to divide our projects into two groups: those below the zero line that exceeded their productivity expectations, and those above the line that did not achieve their productivity expectations. Then we can draw some conclusions.

Factors that were strongly associated with higher than average productivity were:

- The project was completed after 1976, the time when the disciplined application development process, installed in 1974, was beginning to have its effect across all our projects.

- The programming languages and technologies PL/1, DMS/VS, and on-line development were used.
- The improved programming technologies—structured coding, top-down implementation, development library, and HIPO documentation—were used.

Where approved these are now used on all our contracts (naturally such decisions as programming language and the use of DMS/VS are still the prerogative of our customers).

One last chart may give us some clues as to where this productivity comes from. How many lines of designed, written, and tested code must be produced to deliver one point of function value? (Figure 8.) For these projects, DMS/VS averaged 25 lines-of-code per function point delivered, PL/1 averaged 65, and COBOL averaged 110 lines-of-code per one point of function value. You can see why DMS/VS and PL/1 are proving to be more productive than COBOL.

To sum up, measured functionally over the last 5 years, DP Services application development productivity has increased about 3 to 1. This is a compounded productivity increase of about 25% per year.

We have used the function value measure to determine the relative productivity of different languages, technologies, and project sizes. We intend to continue using this functional measure to select and promote technologies that can help us stay competitive.

We found that these improved programming technologies definitely contribute to high productivity. They are an important part of each of our contracts.

We found strong evidence that the disciplined application development process significantly promotes project success. Not only does it consistently yield successful projects by the measures of on-time, within budget, and a satisfied customer, but it is strongly associated with high productivity. In addition it provides the reliable project feedback that we need to make meaningful analysis.

Project phasing, which is an important part of the application development process, has helped define our projects in smaller pieces. In turn these smaller projects have contributed to higher productivity.

Regarding the measurement of project productivity, we found that we had to include the whole process, including the design phase, in the productivity measurement to draw meaningful conclusions.

We found that a disciplined process was an essential ingredient to meaningful productivity measurement. Discipline provide agreed-upon definitions of product and cost consistent across projects being measured.

The function based measurement has proved to be an effective way to compare productivity between projects. Before it was established we could only compare projects that were alike in language and technology, or we had to face the difficult problem of comparing estimates of hypothetical projects against actual results. We intend to continue using and improving the function value measurement.

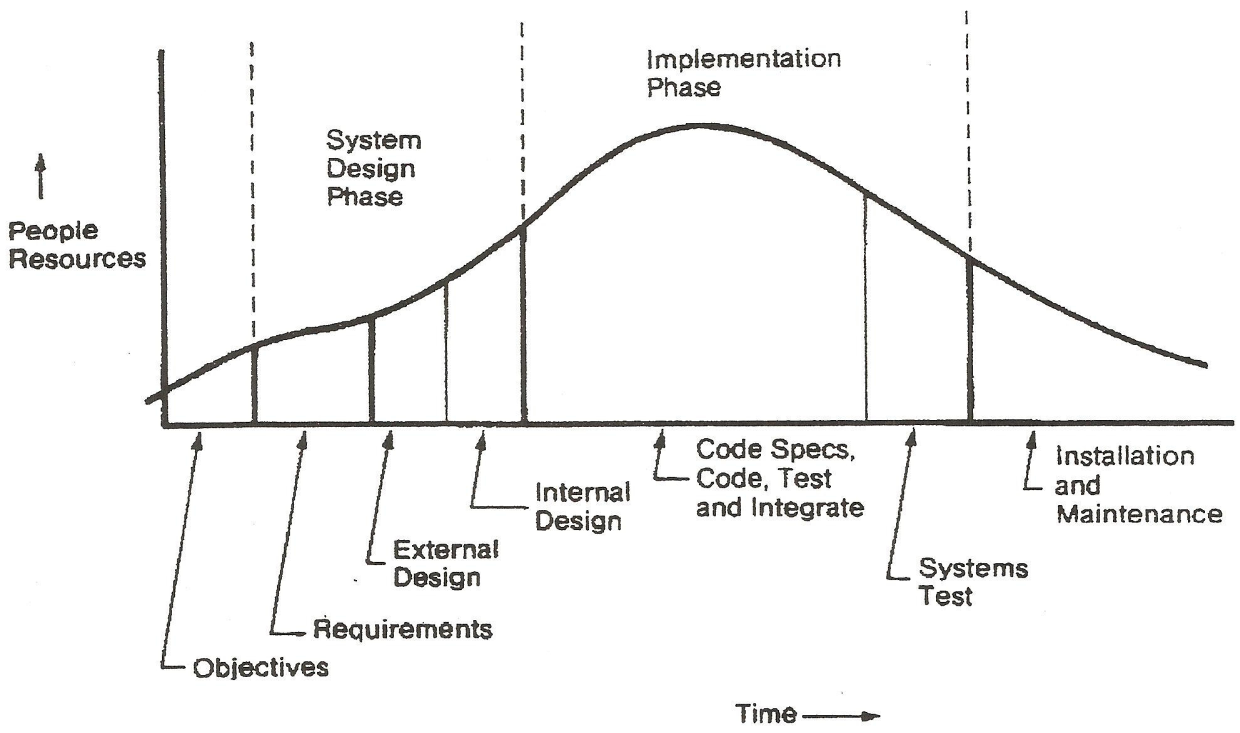


Figure 1

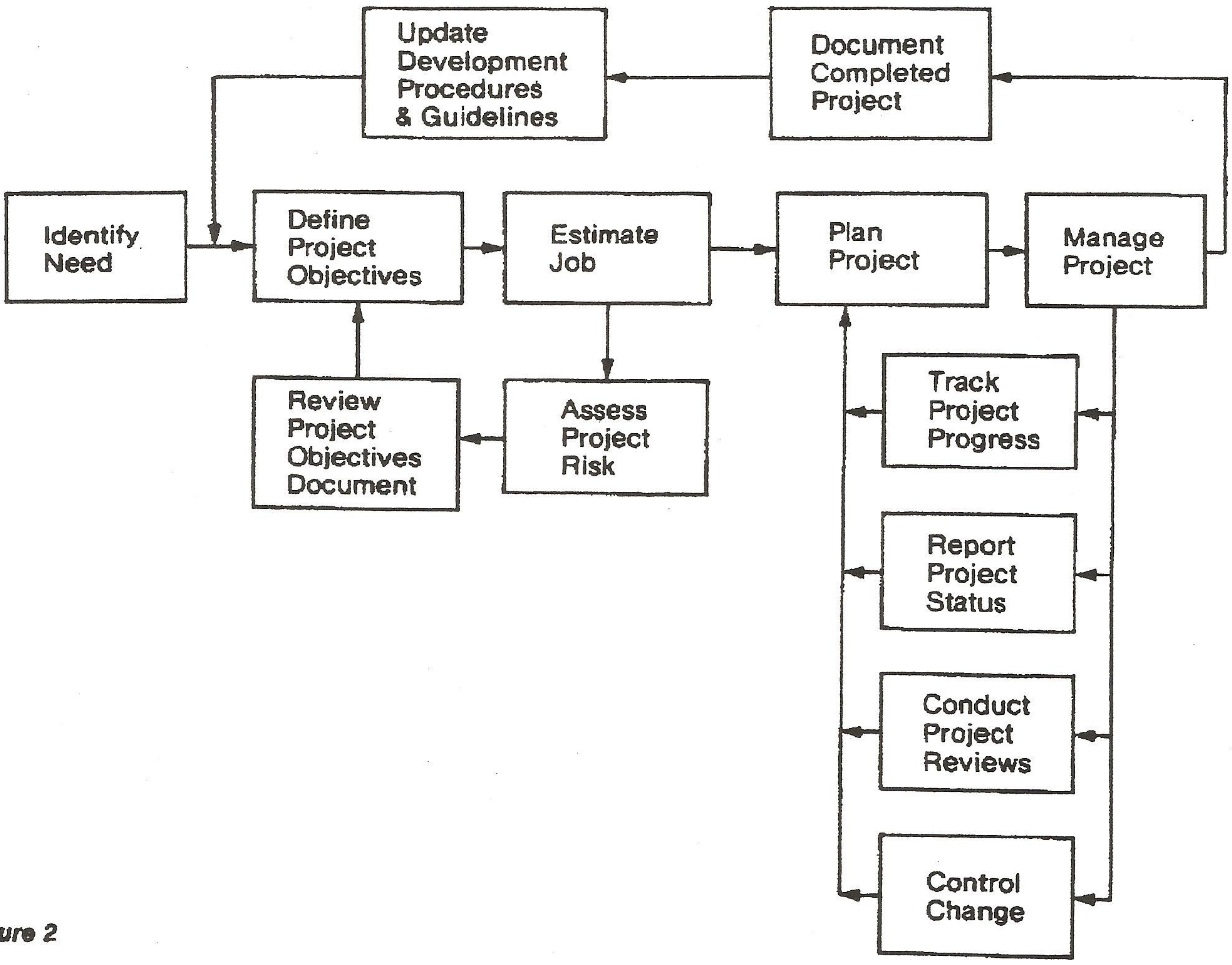
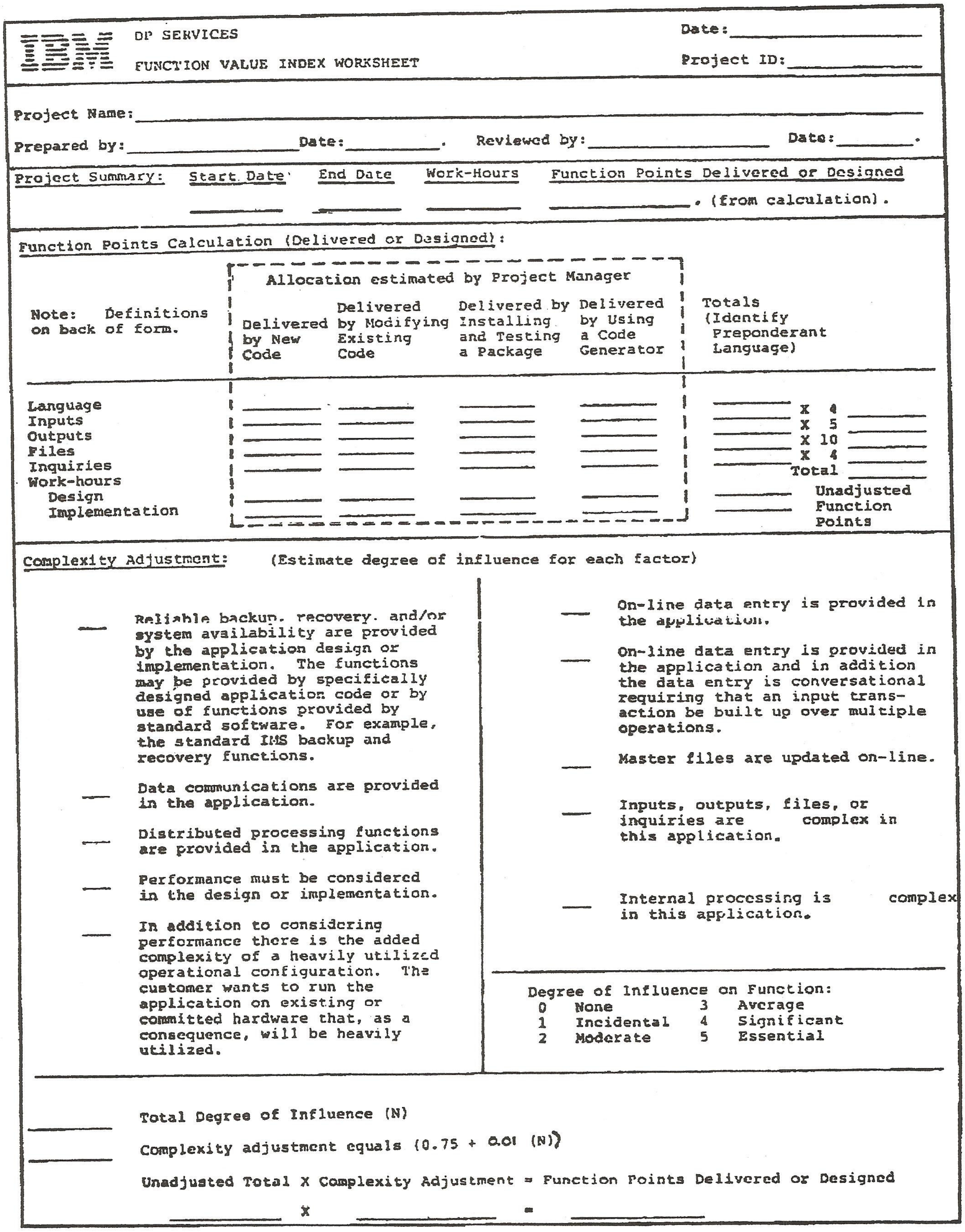


Figure 2



IBM DP SERVICES

FUNCTION VALUE INDEX WORKSHEET

Date: ____________

Project ID: ____________

Project Name: ____

Prepared by: ______________ Date: ________. Reviewed by: ______________ Date: ________.

Project Summary:	Start Date	End Date	Work-Hours	Function Points Delivered or Designed
	________	________	________	________. (from calculation).

Function Points Calculation (Delivered or Designed):

Allocation estimated by Project Manager

Note: Definitions on back of form.	Delivered by New Code	Delivered by Modifying Existing Code	Delivered by Installing and Testing a Package	Delivered by Using a Code Generator	Totals (Identify Preponderant Language)		
Language	____	____	____	____	____		
Inputs	____	____	____	____	____	X 4	____
Outputs	____	____	____	____	____	X 5	____
Files	____	____	____	____	____	X 10	____
Inquiries	____	____	____	____	____	X 4	____
Work-hours						Total	____
Design	____	____	____	____	____	Unadjusted Function Points	
Implementation	____	____	____	____	____		

Complexity Adjustment: (Estimate degree of influence for each factor)

___ Reliable backup, recovery, and/or system availability are provided by the application design or implementation. The functions may be provided by specifically designed application code or by use of functions provided by standard software. For example, the standard IMS backup and recovery functions.

___ Data communications are provided in the application.

___ Distributed processing functions are provided in the application.

___ Performance must be considered in the design or implementation.

___ In addition to considering performance there is the added complexity of a heavily utilized operational configuration. The customer wants to run the application on existing or committed hardware that, as a consequence, will be heavily utilized.

___ On-line data entry is provided in the application.

___ On-line data entry is provided in the application and in addition the data entry is conversational requiring that an input transaction be built up over multiple operations.

___ Master files are updated on-line.

___ Inputs, outputs, files, or inquiries are complex in this application.

___ Internal processing is complex in this application.

Degree of Influence on Function:

0	None	3	Average
1	Incidental	4	Significant
2	Moderate	5	Essential

______ Total Degree of Influence (N)

______ Complexity adjustment equals (0.75 + 0.01 (N))

Unadjusted Total X Complexity Adjustment = Function Points Delivered or Designed

__________ X __________ = __________

Figure 3

Definitions:

General Instruction:

Count all inputs, outputs, master files, inquiries, and functions that are made available to the customer through the project's design, programming, or testing efforts. For example, count the functions provided by an IUP, FDP, or Program product if the package was modified, integrated, tested, and thus provided to the customer through the project's efforts.

Work-hours:

The work-hours recorded should be the IBM and customer hours spent on the DP Services standard tasks applicable to the project phase. The customer hours should be adjusted to IBM equivalent hours considering experience, training, and work effectiveness.

Input Count:

Count each system input that provides business function communication from the users to the computer system For example:

- data forms
- terminal screens
- scanner forms or cards
- keyed transactions

Do not double count the inputs. For example, consider a manual operation that takes data from an input form, to form two input screens, using a keyboard to form each screen before the entry key is pressed. This should be counted as two (2) inputs not five (5).

Count all unique inputs. An input transaction should be counted as unique if it required different processing logic than other inputs. For example, transactions such as add, delete, or change may have exactly the same screen format but they should be counted as unique inputs if they require different processing logic.

Do not count input or output terminal screens that are needed by the system only because of the specific technical implementation of the function. For example, DMS/VS screens, that are provided only to get to the next screen and do not provide a business function for the user, should not be counted.

Do not count input and output tape and file data sets. These are included in the count of files.

Do not count inquiry transactions. These are covered in a subsequent question.

Output Count:

Count each system output that provides business function communication from the computer system to the users. For example:

- printed reports
- terminal screens
- terminal printed output
- operator messages

Count all unique external outputs. An output is considered to be unique if it has a format that differs from other external outputs and inputs, or, if it requires unique processing logic to provide or calculate the output data.

Do not include output terminal screens that provide only a simple error message or acknowledgement of the entry transaction, unless significant unique processing logic is required in addition to the editing associated with the input, which was counted.

Do not include on-line inquiry transaction outputs where the response occurs immediately. These are included in a later question.

File Count:

Count each unique machine readable logical file, or logical grouping of data from the viewpoint of the user, that is generated, used, or maintained by the system. For example:

- input card files
- disk files
- tape files

Count major user data groups within a data base. Count logical files, not physical data sets. For example, a customer file requiring a separate index file because of the access method would be counted as one logical file not two. However, an alphabetical index file to aid in establishing customer identity would be counted.

Count all machine readable interfaces to other system as files.

Inquiry Count:

Count each input/response couplet where an on-line input generates and directly causes an immediate on-line output. Data is not entered except for control purposes and therefore only transaction logs are altered.

Count each uniquely formatted or uniquely processed inquiry which results in a file search for specific information or summaries to be presented as response to that inquiry.

Do not also count inquiries as inputs or outputs.

Figure 3 (Continued)

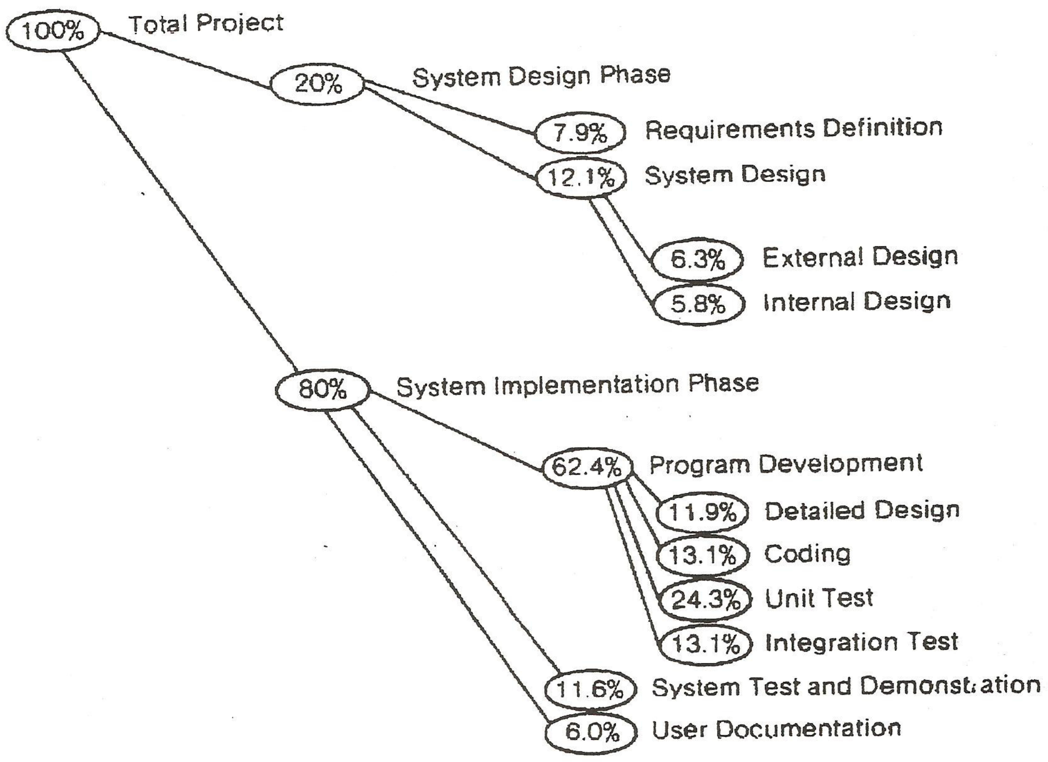


Figure 4

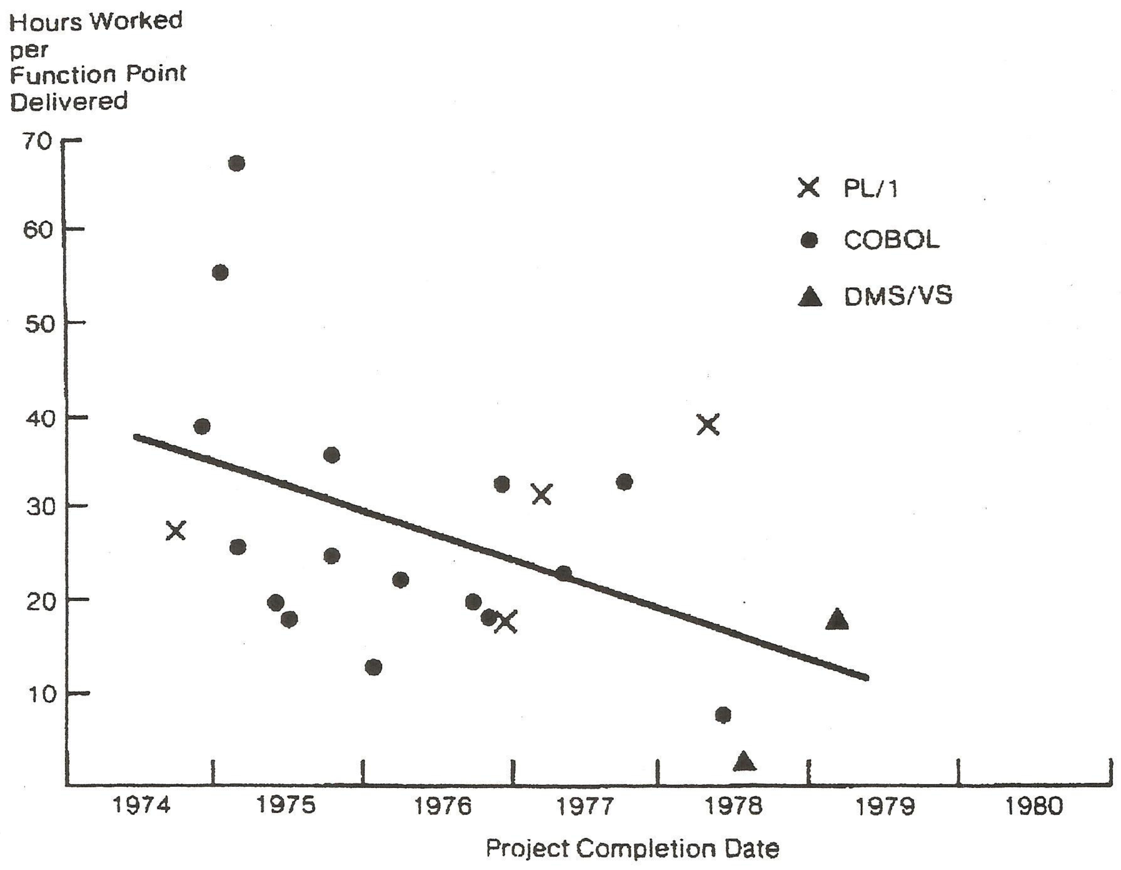


Figure 5

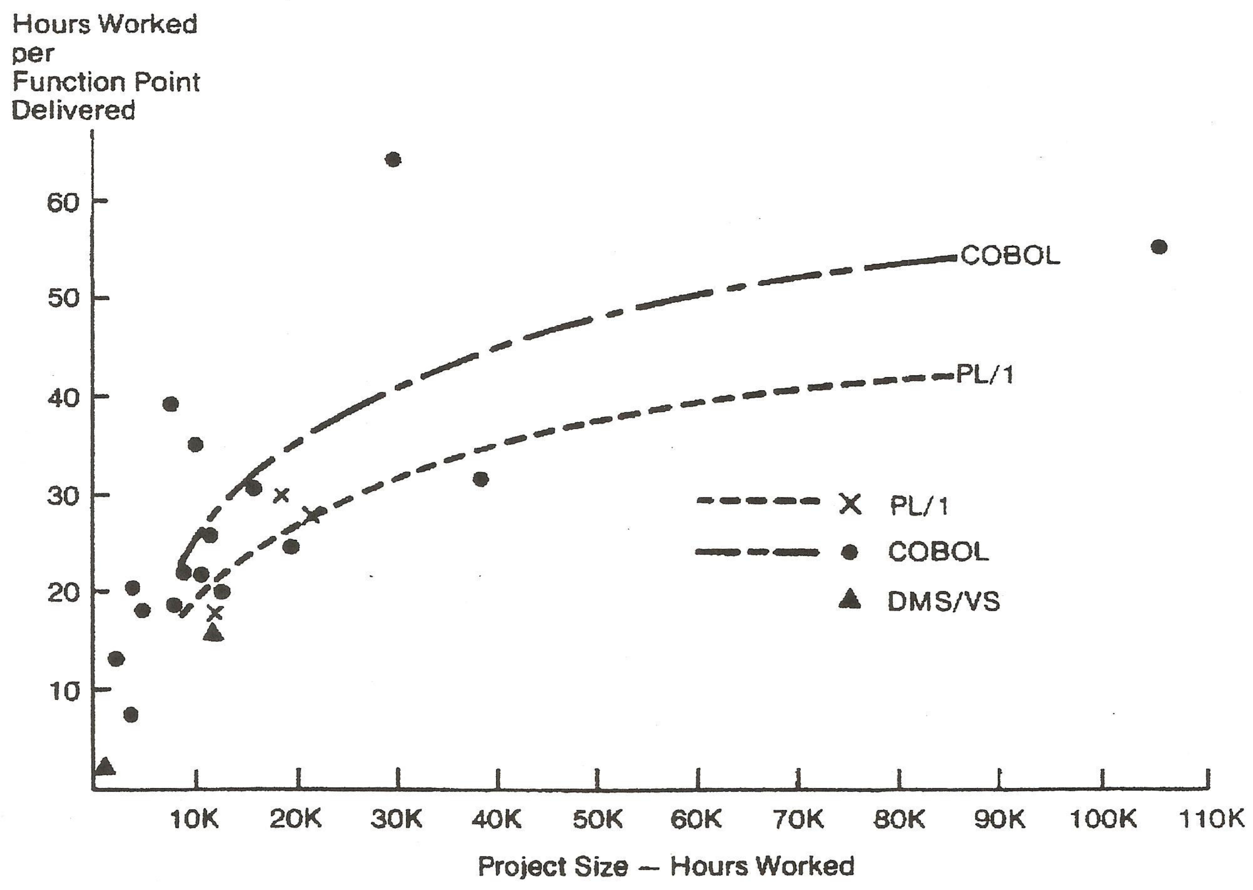


Figure 6

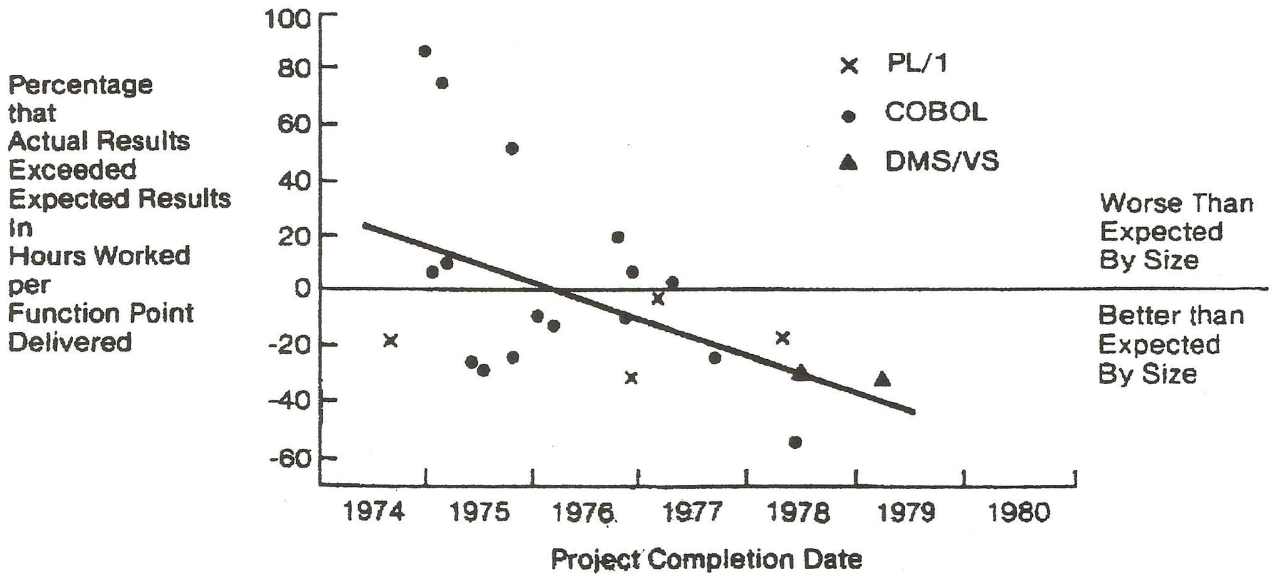
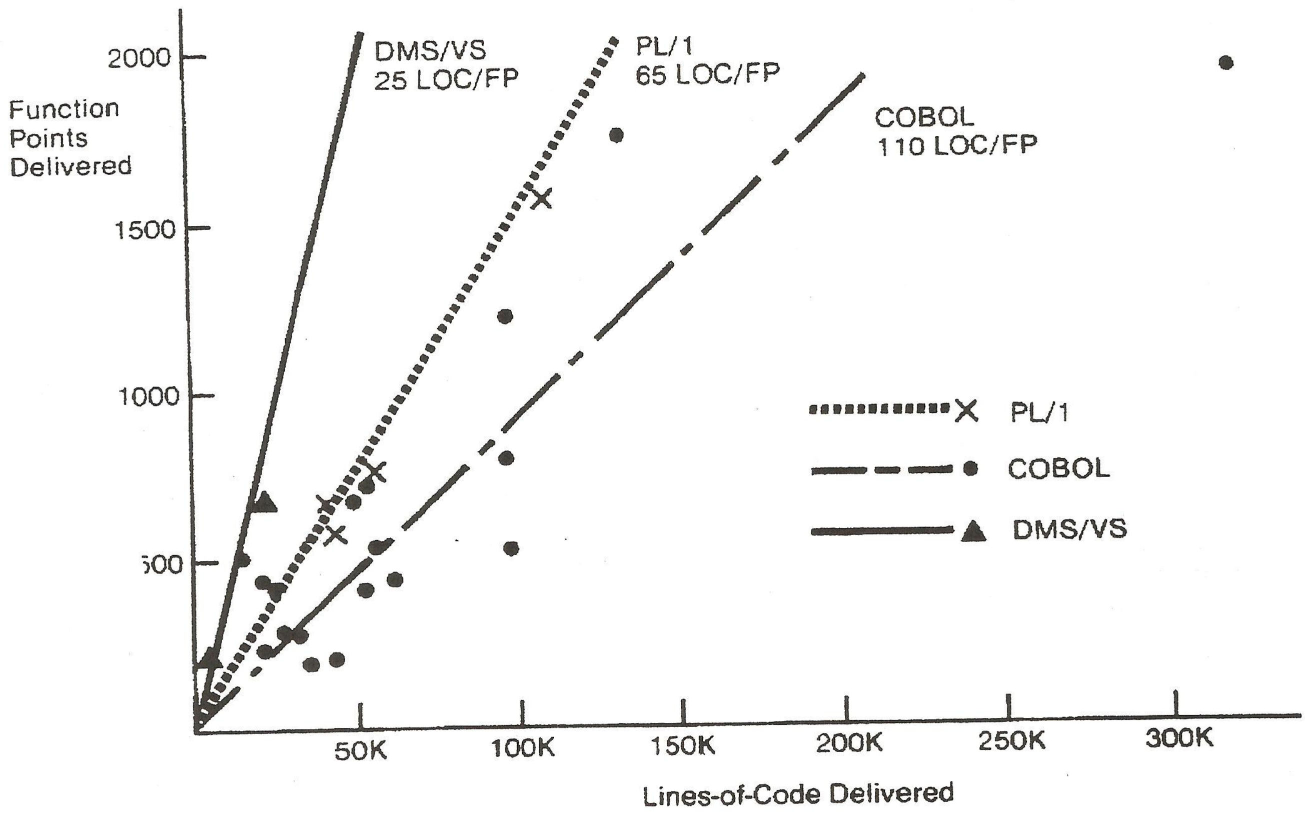


Figure 7



DMS/VS
25 LOC/FP
PL/1
65 LOC/FP
COBOL
110 LOC/FP
Function Points Delivered
2000
1500
1000
500
50K
100K
150K
200K
250K
300K
Lines-of-Code Delivered
PL/1
COBOL
DMS/VS

Figure 8