

Rapport de recherche no. 247

28 avril 1995

**A FORMAL NOTATION FOR THE RULES OF
FUNCTION POINT ANALYSIS**

Paton, K. et Abran, A.

A FORMAL NOTATION FOR THE RULES OF FUNCTION POINT ANALYSIS

Keith Paton and Alain Abran

*Paton Conception de Systemes 350 Pine, St.Lambert. Quebec, J4P 2N8 (514) 671-1969
UQAM, Dept. of Computer Science., C.P. 8888, Succ. Centre-Ville, Montreal, Québec H3C 3P8*

April 26, 1995

fprules.doc

Summary

Function Point Analysis (FPA) is a method for measuring the *functional size* of a software system. The rules governing FPA are laid down in the Counting Practices Manual (the CPM) by the International Function Point Users Group. This paper

- ◇ defines a formal notation (N) for software designs,
- ◇ shows that the notation N is consistent with the language of the the CPM
- ◇ shows that FPA can be expressed as a program (P) operating on a design expressed in N
- ◇ shows that the program P can be expressed in 17 steps of which 12 are algorithmic and five require human judgment.

Each step of the argument is justified by reference to the the CPM. The notation N is consistent with that used by many CASE tools.

The paper is designed for two classes of reader, the function points expert and the tool builder. For the function points expert it captures the mechanism of counting function points as defined in the CPM; for the tool builder it provides the detail required to start building the physical design of an interactive tool for counting function points.

Table of Contents

Summary	1
Table of Contents	2
1 Introduction	3
1.1 Problem	3
1.2 Relevance	3
1.3 The Character of a Solution	3
1.4 Review.....	3
1.5 Proposed Approach	4
2 Overview	5
2.1 Applications, Development and Enhancements.....	5
2.2 The Count for an Enhancement	6
2.3 A First Notation for Design	7
2.4 Finding Added, Modified and Deleted Items	9
2.5 Defining the Boundary of the Enhancement	11
2.6 Classifying the Files	12
2.7 Classifying the Processes	12
2.8 Unique Processes and Files	20
2.9 Defining the count	21
2.10 A Second Notation for Design.....	24
3 Formal procedure for Counting Function Points	25
3.1 Activities in Counting Function Points	25
3.2 Precedence Analysis.....	26
3.3 Traceability.....	27
3.4 The notation extended for Function Points	28
4 Conclusion.....	29
5 Future Directions.....	29
5.1 Building an interactive tool based on CPM 4.0.....	30
5.2 Reviewing existing tools based on CPM 4.0	30
5.3 Extending the rules to include uncodified practice used by expert counters.....	30
5.4 Building a tool for a standards body	30
6 References	31
Appendix A.....	32
A1 Geometrical and Set-Theoretical Language for defining a boundary.....	32
A2 The Activities in Capturing a Design	33
Appendix B.....	34
B1 Double process	34
B2 Displays, persistent and transient data	35
B3 Re-use.....	36
B4 Decomposing an inquiry.....	36
A7 Reviewer's Comment Form.....	37

1 Introduction

This introduction defines the problem to which Counting Function Points is a solution, shows why that problem is important, and explains how to recognize a solution. It reviews existing methods, noting the gaps, and proposes an approach that fills these gaps.

1.1 Problem

Managers of Information Systems would like a solution to the following problem

EFFORT ESTIMATION (1) : Given the *detailed design and constraints* for a software system and a description of the *development environment*, estimate the *effort* required to build the system in that environment.

Albrecht (1979) proposed that

1. each software system has a size (in function points)
2. the size of a software system (in function points) can be measured from its detailed design and constraints
3. the effort to build a software system can be estimated from its size (in function points) and a description of the development environment

Abran (1994) expressed this large problem as two smaller problems that can be paraphrased as

SIZE MEASUREMENT : Given the *detailed design and constraints* for a software system measure the *size* (in function points) of that system

and

EFFORT ESTIMATION (2) : Given the *size* of a software system, (in function points) and a description of the *development environment*, estimate the *effort* required to build the system in that environment.

The latter problem — Effort estimation (2) — must involve some model of the productivity of the development environment (Abran, 1994) and will not be further considered. This paper is about the problem of size measurement

1.2 Relevance

Those who choose which projects to attempt based on *cost-benefit analysis* must estimate the cost and the benefit before starting. Measuring the size of the system to be built is part of the task of estimating the cost via the effort.

1.3 The Character of a Solution

Since the method was introduced by Albrecht (1979), the authoritative version has been that defined by the International Function Point Users Group. Throughout its history different experts have obtained different sizes for the same design, differences that have been resolved at meetings of the Users Group. The current definition of the method is version 3.4 of the Counting Practices Manual (IFPUG , 1994); this will be denoted the CPM. The rules and examples now run to over 200 pages and a lengthy period of study is required to master them.

This paper proposes that one way of solving the size measurement problem is to embody the rules of the CPM in a suitable computer program. Thus the solution sought in this paper can be defined as follows:

A *complete* solution is an *algorithm* for counting FP from a detailed design; a *partial* solution is an essential design for FP, showing what is *algorithmic* and what is *interactive*; the greater the algorithmic proportion the closer the partial solution is to a complete solution.

1.4 Review

1.4.1 The Original Formulation

Function Point Analysis was introduced by Albrecht (1979) and is now guided by the International Function Point Users Group which has modified the FP measurement standards over the years, presenting the latest version each time in an updated version of the Counting Practices Manual (the CPM) ; the current release is 4.0 (IFPUG , 1994).

The objectives of FPA (IFPUG , 1994) are

1. to measure what the user requested and received
2. to measure independently of the technology used for implementation
3. to provide a sizing metric to support quality and productivity analysis
4. to provide a vehicle for software estimation
5. to provide a normalization factor for software comparison

The method lacks a formal basis, since the object to be counted is nowhere defined. It is clear from the the CPM that the object to be measured must include such things as files, records, fields, processes and that it must be known which processes read and write which fields in which records in which files.

1.4.2 Re-arrangement of the Rules

St-Pierre (1993) has used set structured hypertext (Meyerhoff and Mullerburg, 1992) to analyze the rules in the CPM 3.4 (IFPUG, 1993) and identify a set of 53 concepts such that each of the 137 rules can be associated with a single concept or a pair of concepts. This makes it possible to present the rules as a conceptual map in the form of structured hypertext, indexed by a graph whose nodes represent concepts and whose edges represent relationships between pairs of concepts. However this merely brings the rule book closer to the user; it still leaves the process of counting entirely to the user.

1.4.3 Computer-aided Counting

Various tools have been devised to help users count function points as shown in table 1.

<i>Tool</i>	<i>Developer</i>
Function Point Mentor r1.0	Angel Group
Metrics Manager	Computer Power Group
CA-FPXpert	Howard Rubin Associates, Inc.
IEF - Information Engineering Facility	Texas Instruments
LINC Logical and Informat Network Compiler	Unisys - Brazil
Via Recap	Viasoft

Table 1: Tools for Counting Function Points

Although a full critical review of these tools has yet to be published, it is worth noting that

- ◇ Some of these are proprietary; they work only for designs expressed in a particular notation defined by the tool and not open to the public.
- ◇ Some of the tools suggest that CFP can be completely automated.

Proprietary tools cannot be used for designs imported from some other place, notably a general CASE tool. This paper shows that CFP can not be completely automated but that several key steps require human judgment. None of the tools named above offers an interactive approach to CFP for a design expressed in a notation open to the public.

1.5 Proposed Approach

In what follows a hybrid approach is proposed, automating the parts that require no judgment and providing as much assistance as possible in the book-keeping for the parts that require judgment. All design decisions are justified by referring to the CPM. The approach proposed, with only five interactive steps, is a partial solution to the problem defined in 1.1

The paper is designed for two classes of reader, the function points expert and the tool builder. For the function points expert it captures the mechanism of counting function points as defined in the CPM; for the tool builder it provides the detail required to start building the physical design of an interactive tool.

2 Overview

As Agresti points out in his preface to Card (1991)

Measurement of physical reality, we learned, involved three elements: an *object*, an *observable characteristic*, and an *apparatus* for performing the measurement.

In the case of counting function points, the *object to be measured* must be some representation of the software system, although the CPM nowhere defines just how the software system is to be represented for the purpose of measuring its size. The *observable characteristic* is the function point itself and the *apparatus for performing the measurement* is a human expert using the rules of the CPM.

2.1 Applications, Development and Enhancements

The Counting Practices Manual recognizes that counting function points takes place on three types of object: a Development Project, an Enhancement Project and an Application. To show how the three types of counts are related, it is convenient to regard an enhancement as the passage from an existing application, $A(n)$ say, to a new "enhanced version" of that application, $A(n+1)$ say. The activity of transforming $A(n)$ into $A(n+1)$ may be called the n 'th enhancement project and denoted $E(n)$, as shown in Figure 1.

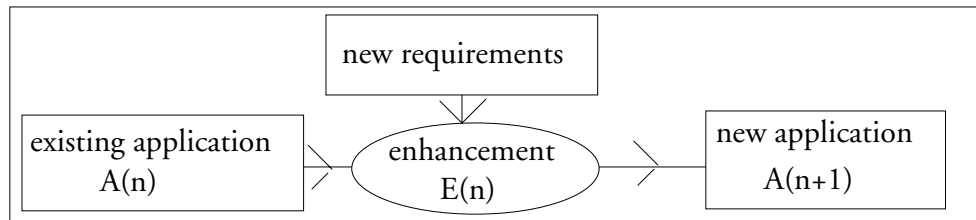


Figure 1: An Enhancement

Software development as a whole may thus be regarded as a development project D that creates an initial application $A(1)$ followed by a sequence of enhancement projects $\{E(1), E(2), \dots\}$ that create a sequence of applications $\{A(2), A(3), \dots\}$ as suggested in Figure 2.

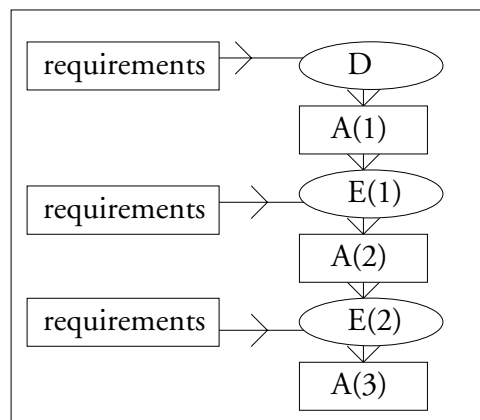


Figure 2: The software development process

The CPM explains that the development process has a size, each enhancement process has a size and each application has a size. If we represent the activity of measuring that size as a bubble labelled "count" then the software development process can be shown as in figure 3.

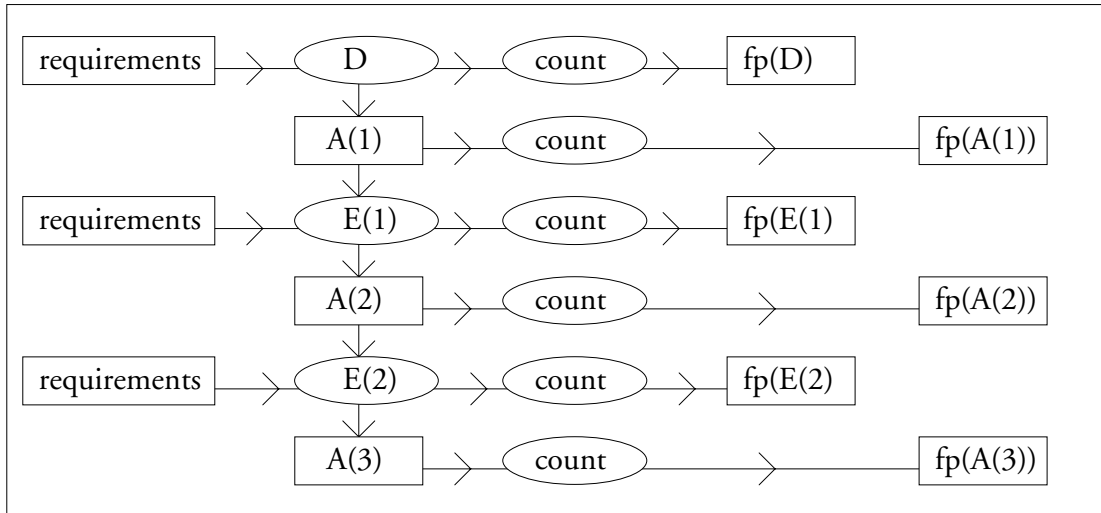


Figure 3: The software development process with measurement of size

Let us refer to an enhancement by naming a pair of applications. Thus the enhancement $(A(1), A(2))$ takes application $A(1)$ into $A(2)$ and so on. Then

- ◇ count applied to the initial development project D yields a real number denoted $fp(D)$
- ◇ count applied to an application $A(i)$ yields a real number denoted $fp(A(i))$
- ◇ count applied to an enhancement project $(A(n), A(n+1))$ yields a real number denoted $fp(A(n), A(n+1))$

From the rules given in the CPM, measuring the size of an enhancement project seems to be more complicated than measuring the size of an application. Therefore, the main text will deal with the enhancement project; the application will be dealt with in 2.2.2.

2.2 The Count for an Enhancement

The CPM defines the function point count of an enhancement as

$$EFP = [(ADD + CHGA + CFP) * VAFA] + (DEL * VAFB)$$

where:

- ◇ ADD is the unadjusted function point count of those functions that were added by the enhancement project
- ◇ CHGA is the unadjusted function point count of those functions that were modified by the enhancement project. This number reflects the functions *after* the modifications.
- ◇ CFP is the function point count added by the conversion.
- ◇ VAFA is the value adjustment factor of the application *after* the enhancement project
- ◇ DEL is the unadjusted function point count of those functions that were deleted by the enhancement project
- ◇ VAFB is the value adjustment factor of the application *before* the enhancement project

The two value adjustment factors may be dealt with here. The CPM recognizes 14 general characteristics which apply to the system as a whole, rather than to any particular file or process. The user assigns an integer in $[0,5]$ to each one; the sum of all 14 is therefore in $[0,70]$ and a medium pattern of characteristics yields a sum of 35. Then the value adjustment factor before (VAFB) and the value adjustment factor after (VAFA) are defined by

$$VAFB = 0.65 + (\text{sum of general characteristics before enhancement}) / 100.0$$

$$VAFA = 0.65 + (\text{sum of general characteristics after enhancement}) / 100.0$$

This ensures that an average set of general characteristics leads to an adjustment factor of 1.

2.2.1 The Count for a Development Project

The difference between the initial development project and an enhancement project is more symbolic than real. If a null application $A(0)$ is defined, the initial development project can be regarded as the zero'th enhancement project, transforming the null application $A(0)$ into the initially delivered application $A(1)$. The development project may therefore be regarded as a special case of an enhancement project.

2.2.2 The Count for an Application

The difference between an application and an enhancement is a little more significant. If we measure the function points for the enhancement that takes $A(0)$ into $A(n)$ then there is nothing to modify and nothing to delete and the formula becomes

$$fp(A(0), A(n)) = [(ADD + CFP) * VAFA]$$

and if we now subtract the one-time conversion points ($CFP*VAFA$) we obtain

$$[(ADD * VAFA)]$$

which is the function point count of the application $A(n)$. Thus

$$fp(A(n)) = fp(A(0), A(n)) - CFP*VAFA$$

showing that the count of an application can be obtained by slightly varying the steps that obtain the count for an enhancement.

2.3 A First Notation for Design

The paper describes the methodology of counting function points as a computer program. The first requirement is to express all designs in some notation that can be understood by a computer program.

The idea that a design can be expressed as a picture in which boxes represent stored data and bubbles represent the processes that manipulate that stored data dates back at least to DeMarco (1979) and the presence of this notation in many CASE tools suggests that it is a powerful unifying force in system design. Figure 4 shows a design in this notation.

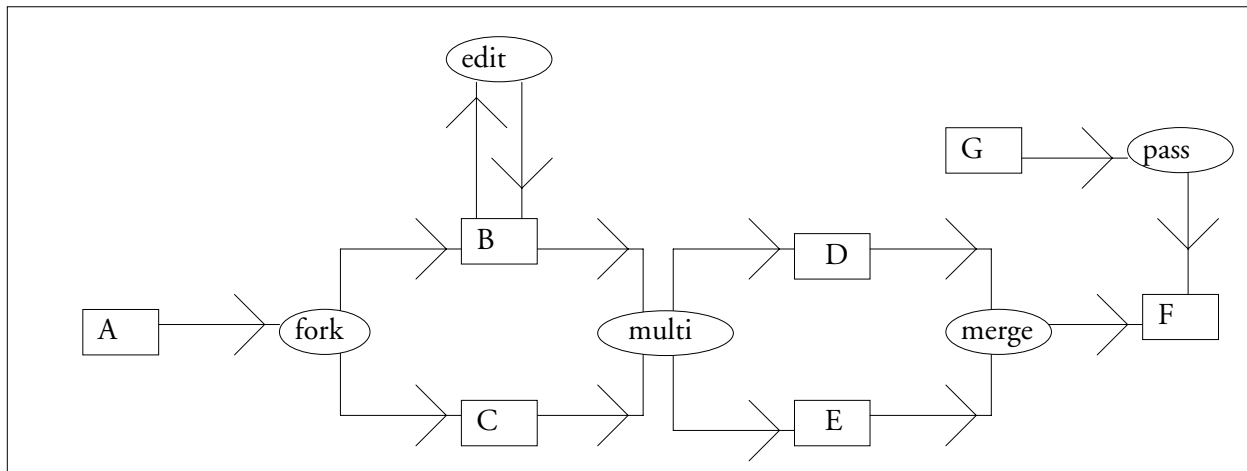


Figure 4: A design in the unconstrained bubble-box notation

The processes have been named to suggest how many inputs and outputs they take. In the general form, any process can have many inputs and many outputs and any stored datum can be read by many processes and written by many processes.

The design of figure 4 can be represented equivalently in tabular form as in table 2.

READS		WRITES	
process	store	process	store

fork	A
multi	B
multi	C
edit	B
merge	D
merge	E
pass	G

fork	B
fork	C
edit	B
multi	D
multi	E
merge	F
pass	F

Table 2: Design of figure 4 as a table

The picture is an example of the schema

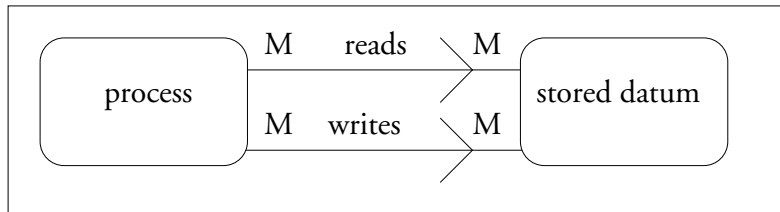


Figure 5: Schema for unconstrained bubble-box notation

Any *design* can thus be summarized as a quadruple (P,S,R,W) where

- ◇ P is a set of processes
- ◇ S is a set of stores
- ◇ R is the reads relation from P to S
- ◇ W is the writes relation from P to S

Although every design can be expressed as a quadruple (P,S,R,W) not every quadruple makes sense as a design. Consider figure 6,

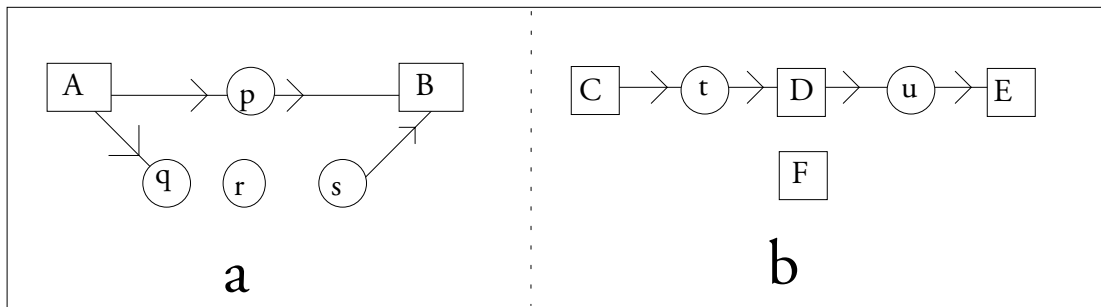


Figure 6: Some pathological processes and stores

In figure 6a, since process q writes to no store, the state of the stores after executing process q is the same as that before executing it; hence q is completely useless. Similar arguments show that of the four processes in figure 6a, only process p can claim to be useful; the others can just as well be omitted.

Likewise in figure 6b, store C is read but not written by the system shown; presumably it is written by some other system. Store E is written but not read by the system shown; presumably it is read by some other system. Since store F is neither read nor written by the system shown it can play no part in the system and can just as well be omitted.

These observations can be summarized in the principle

A quadruple (P,S,R,W) represents a valid design when and only when

1. every process reads at least one store *and* writes at least one store
2. every store is read by at least one process *or* written by at least one process

The description of the notation has followed DeMarco in using the terms "process" and "store"; to be consistent with the CPM, we replace the word "store" by the word "file" from now on.

2.4 Finding Added, Modified and Deleted Items

The first requirement is to find the files and processes added, modified and deleted by the enhancement. Suppose we are dealing with the enhancement $E(n)$ that takes application $A(n)$ into $A(n+1)$. Files and processes that are added or deleted are easy to detect, they occur in one but not both of $A(n)$ and $A(n+1)$. Modified files and processes are rather more complicated.

2.4.1 Finding Modified Files

A file is modified if its properties in $A(n+1)$ differ from its properties in $A(n)$. The properties of a file are the properties of its records and the properties of a record are the properties of its fields.

Consider the situation

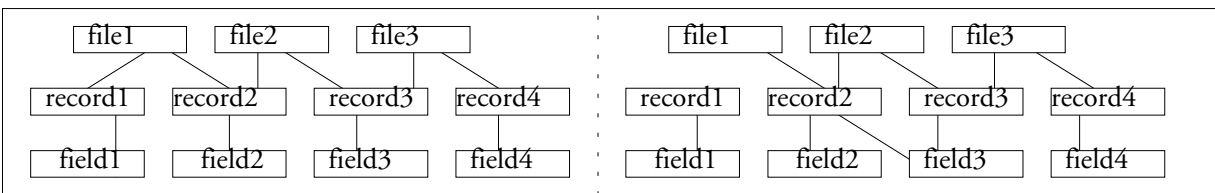


Figure 7: Modified and Unmodified files

- ◇ File 1 is modified; on the left it contains record1; on the right it does not.
- ◇ File 2 is modified; it still contains record2 and record3 but on the left record2 contains just field2 whereas on the right it contains field2 and field3.
- ◇ File 3 is unmodified; in both cases it contains record3 and record4 and both are unmodified.

The logic described above can be summarized in the following table. Consider the existing application, $A(n)$ say, and the enhanced application, $A(n+1)$ say and examine each file f in $A(n) \cup A(n+1)$.

If ...	file f is said to be ...
f is not in A(n) and f is in A(n+1)	ADDED
f is in A(n) and f is not in A(n+1)	DELETED
f contains different records in A(n) and A(n+1) or (f contains a record r in both A(n) and A(n+1) and r contains different fields in A(n) and A(n+1))	MODIFIED
none of the above	UNCHANGED

Table 3: Finding added, deleted and modified files

2.4.2 Finding Modified Processes

A process is completely determined from the point of view of the detailed design by three properties

- ◇ a list of what it reads
- ◇ a list of what it writes
- ◇ a description of what it does, the processing

Hence process p is modified in enhancement E(n) if one or more of these three properties differs from A(n) to A(n+1). Figure 8 shows examples:

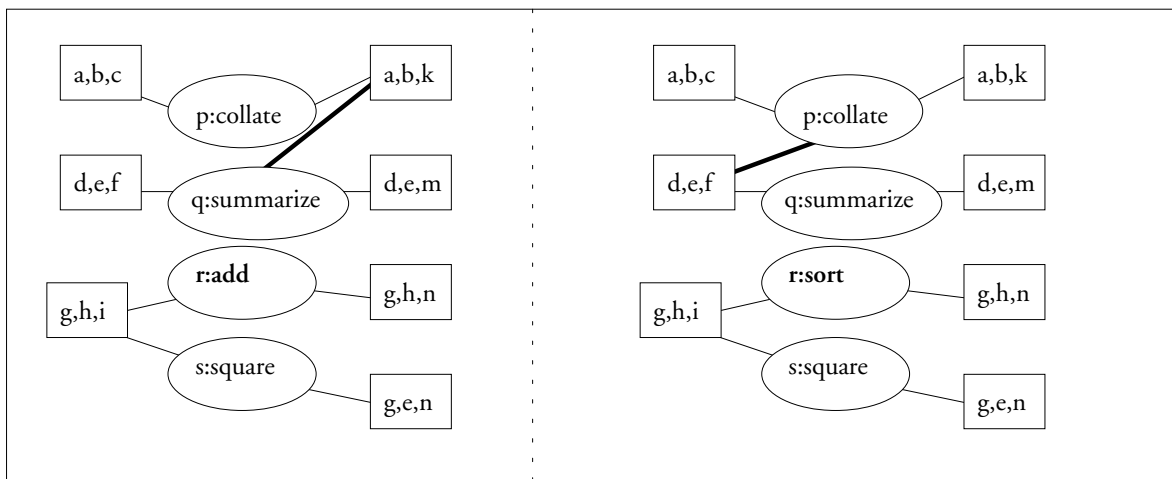


Figure 8: Modified and Unmodified processes

Differences are marked in bold.

- ◇ Process p is modified; it reads one more triple in A(n+1) than in A(n)
- ◇ Process q is modified; it writes one less triple in A(n+1) than in A(n)
- ◇ Process r is modified; it has a different description in A(n+1) from that in A(n)
- ◇ Process s is unmodified

The logic described above can be summarized in the following tables. Consider the existing application, A(n) say, and the enhanced application, A(n+1) say and examine each process p in A(n) ∪ A(n+1).

If ...	process p is said to be ...
p is not in A(n) and p is in A(n+1)	ADDED
p is in A(n) and p is not in A(n+1)	DELETED
p reads a different set of files in A(n) and A(n+1) or p writes a different set of files in A(n) and A(n+1) or p is differently described in A(n) and A(n+1)	MODIFIED
none of the above	UNCHANGED

Table 4: Definition of different kinds of processes

CFP recognizes that in going from an existing application A(n) to a new application, A(n+1) some "one-time" software may be required to transform data from old formats to new formats. Items and processes thus involved are said to be USED IN CONVERSION.

2.5 Defining the Boundary of the Enhancement

The CPM explains that the first step in counting function points is to define a boundary between the items (processes, stores) to be counted and the items not to be counted. It explains this in the language of plane geometry, thus

Use the system external specifications or get a system flow chart and draw a boundary round it to highlight which parts are internal and which parts are external to the application.

Many different boundaries can be drawn in the plane but from the point of view of function points, the only thing that matters is which processes and stores are inside and which are outside the boundary. The language of plane geometry offers valuable help to the user who sees the system as drawn in the plane but for computational purposes the geometrical analogy does no more than cover the task with an extra layer of difficulty. Any concept of *function points* that can be expressed in the language of plane geometry using the notion of processes and stores being inside or outside a *boundary* can be equivalently expressed in the language of set theory using the notions of processes or stores belonging or not belonging to a *distinguished subset* of the processes and stores. For computational purposes, it is considerable simpler to work directly in the language of set theory rather than the language of plane geometry. This is explained in detail in appendix A1. In the main text, we follow the language of the CPM and talk about processes and files being inside and outside a boundary.

The processes in P are placed inside or outside B by following this rule:

If p is MODIFIED or ADDED or DELETED or USED FOR CONVERSION
then p is inside B otherwise it is outside B.

Likewise, the files in F are placed inside or outside B by following this rule

If f is MODIFIED or ADDED or DELETED or USED FOR CONVERSION
then f is inside B; otherwise it is outside B.

The CPM defines the function point count for an enhancement as the sum of the function point count over the items added. modified or deleted or used for conversion. Since the interior of B has been defined to contain exactly the items added. modified, deleted or used for conversion, it follows that the function point count of the enhancement is the sum of the function point count over the interior of B.

2.6 Classifying the Files

Given the boundary B, the files are classified according to the table below

f is inside B	f is outside B	
	<i>f is read by some p inside B</i> or <i>f is written by some p inside B</i>	<i>f is read by no p inside B</i> and <i>f is written by no p inside B</i>
f is an internal logical file ILF	f is an external interface file EIF	f is an ignored file IGF

Table 5: Different types of store

where the notation IGF is introduced here to ensure that every file is classified

2.7 Classifying the Processes

2.7.1 Classifying the Processes by the CPM

The CPM provides rules for recognizing three types of process,

- ◇ the external input,
- ◇ the external output
- ◇ the external inquiry

It is implicit that the rules should allow the user to classify a given process, that is assign it to at most one of these three types. It is clear from the rules of the CPM that the classification has much to do with the answers to these questions

- ◇ does the process read outside B?
- ◇ does the process write outside B?
- ◇ does the process read inside B?
- ◇ does the process write inside B?

There are therefore 16 cases to consider. Each cell in table 6 shows an example of each case and quotes a rule number from the CPM to show the categories into which this case cannot fall. The notation 6.31.7 means chapter 6, page 31, bullet 7 and so on.

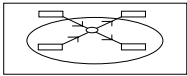
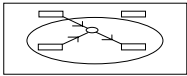
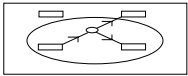
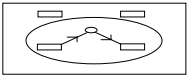
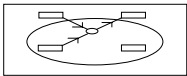
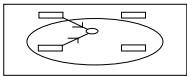
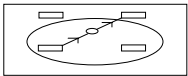
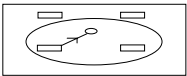
		reads outside B?	yes	yes	no	no
		writes outside B?	yes	no	yes	no
reads inside B?	writes inside B?					
yes	yes					
			not an EQ (6.31.7)	not an EO (6.19.1) not an EQ (6.31.7)	not an EI (6.7.1) not an EQ (6.31.1)	not an EI (6.7.1) not an EO (6.19.1) not an EQ (6.31.1)
yes	no					
			not an EI (6.7.2)	not an EI (6.7.2) not an EO (6.19.1) not an EQ (6.31.7)	not an EI (6.7.1) not an EQ (6.31.1)	not an EI (6.7.1) not an EO (6.19.1) not an EQ (6.31.1)

Table 6: Detailed Classification of Processes

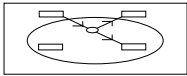
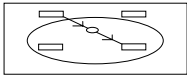
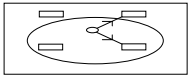
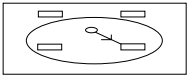
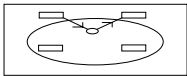
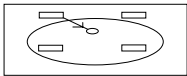
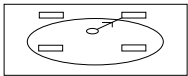
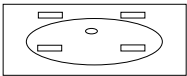
		reads outside B?	yes	yes	no	no
		writes outside B?	yes	no	yes	no
reads inside B?	writes inside B?					
no	yes					
			not an EQ (6.31.7)	not an EO (6.19.1) not an EQ (6.31.7)	not an EI (6.7.1) not an EQ (6.31.7)	not an EI (6.7.1) not an EO (6.19.1) not an EQ (6.31.7)
no	no					
			not an EI (6.7.2)	not an EI (6.7.2) not an EO (6.19.1) not an EQ (6.31.7)	not an EI (6.7.1) not an EQ (6.31.7)	not an EI (6.7.1) not an EO (6.19.1) not an EQ (6.31.7)

Table 6(concluded): Detailed Classification of Processes

Table 7 summarizes the situation so far.

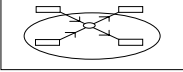
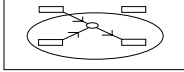
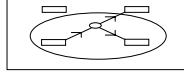
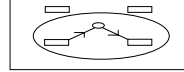
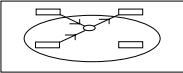
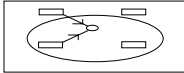
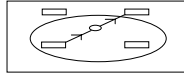
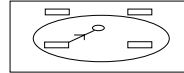
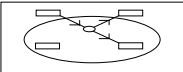
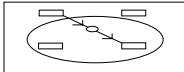
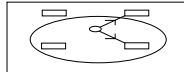
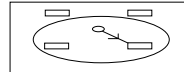
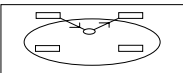
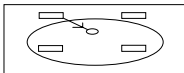
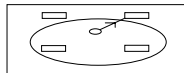
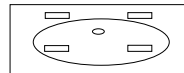
		reads outside B?	yes	yes	no	no
		writes outside B?	yes	no	yes	no
reads inside B?	writes inside B?					
yes	yes		 EI or EO?	 EI?	 EO?	 NIL
yes	no		 EO or EQ?	 NIL	 EO?	 NIL
no	yes		 EI or EO?	 EI?	 EO?	 NIL
no	no		 EO or EQ?	 NIL	 EO?	 NIL

Table 7: Intermediate Classification of Processes

Of the 16 possibilities,

- ◇ six are definitely NIL
- ◇ six have one possibility remaining (four EO and two EI)
- ◇ two are ambiguous (EI or EO)
- ◇ two are ambiguous (EO or EQ)

The four NIL cases in the right hand column can be ignored. Table 8 shows the remaining 12 cases together with a suggested classification.

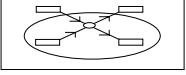
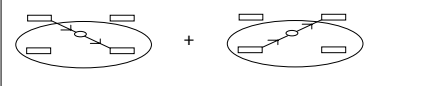
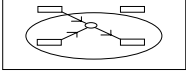
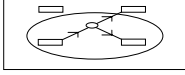
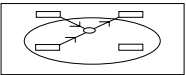
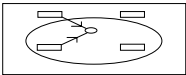
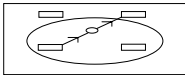
 <p> EI or EO? This is a double process expressed as the sum of an EI and an EO, thus.  </p>	 <p> EI? This is an EI, an update of existing data, as in $\text{new balance} = \text{old balance} + \text{deposit}$ </p>	 <p> EO? This is an EO; it writes to an ILF, an activity not prohibited by the rules </p>
 <p> EO or EQ? This is an EQ (It could also be an EO; it reads from outside and reads from inside, neither being prohibited by the rules) </p>	 <p> NIL </p>	 <p> EO? This is an EO </p>

Table 8: Further Classification of Processes

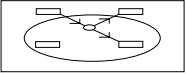
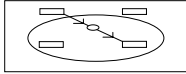
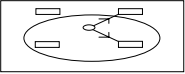
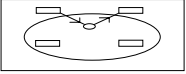
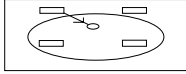
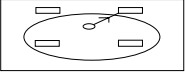
 <p>EI or EO?</p> <p>This is an EI that also writes outside B, an activity not prohibited by the rules.</p>	 <p>EI?</p> <p>This is an EI, an over-write of existing data as in address = new address the old address is destroyed unread</p>	 <p>EO?</p> <p>This is a NIL</p> <p>How can this be an EO; it reads from nowhere? This contravenes the intention but not the letter of the CPM</p>
 <p>EO or EQ?</p> <p>This is a NIL; it does nothing inside the boundary. This contravenes the intention but not the letter of the CPM</p>	 <p>NIL</p>	 <p>EO?</p> <p>This is a NIL</p> <p>How can this be an EO; it reads from nowhere? This contravenes the intention but not the letter of the CPM</p>

Table 8 (continued): Further Classification of Processes

This yields the classification of table 9:

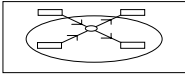
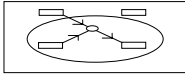
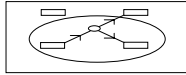
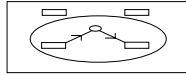
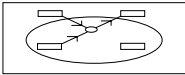
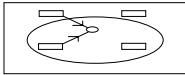
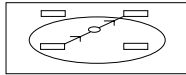
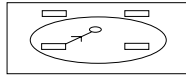
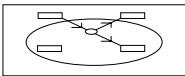
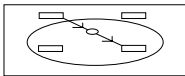
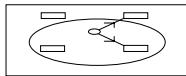
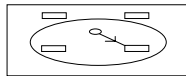
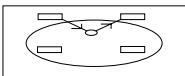
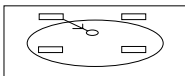
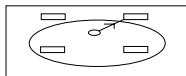
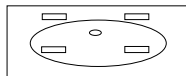
		reads outside B?	yes	yes	no	no
		writes outside B?	yes	no	yes	no
reads inside B?	writes inside B?					
yes	yes		 DOUBLE	 INPUT	 OUTPUT	 NIL
yes	no		 INQUIRY	 NIL	 OUTPUT	 NIL
no	yes		 INPUT	 INPUT	 NIL	 NIL
no	no		 NIL	 NIL	 NIL	 NIL

Table 9: Final Classification by the CPM

Although all cases have now been classified, extra logic not found in the CPM has been required. In particular, in three places we have labelled a case as a NIL because we believe it contravenes the intention but not the letter of the CPM. This suggests that a simpler classification is needed

2.7.2 Classifying the Processes -a unified approach

A simpler classification of the processes can be based on eight rules, four primary and four secondary. Consider the four primary rules in table 10 below.

1	A process must read or write outside B
2	A process must read or write inside B
3	A process must read somewhere
4	A process must write somewhere

Table 10: Four Primary Rules for Classifying Processes

These four rules reject nine of the 16 candidates as shown in the table.

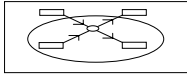
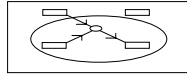
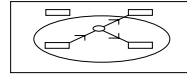
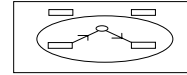
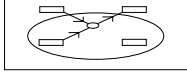
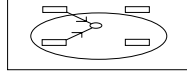
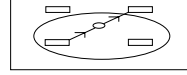
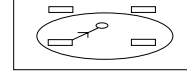
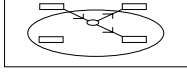
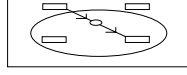
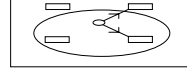
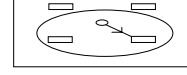
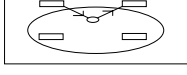
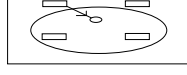
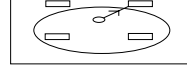
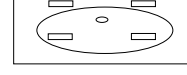
		reads outside B?	yes	yes	no	no
		writes outside B?	yes	no	yes	no
reads inside B?	writes inside B?					
yes	yes					 NIL (1)
yes	no			 NIL (4)		 NIL (1)
no	yes				 NIL (3)	 NIL (1)
no	no		 NIL (2)	 NIL (2)	 NIL (2)	 NIL(1)

Table 11: Classification of Processes using Unified Rules 1-4

There are seven surviving cases. Now consider the four secondary rules in table 12 which lead to the final classification of table 13.:

5	A surviving process that reads outside B but does not write outside B is an INPUT
6	A surviving process that writes outside B but does not read outside B is an OUTPUT
7	A surviving process that reads and writes outside B is an INQUIRY if it reads inside B and an INPUT if it writes inside B
8	A surviving process that carries on all four activities is a DOUBLE PROCESS and must be decomposed as an input plus an output.

Table 12: Four Secondary Rules for Classifying Processes

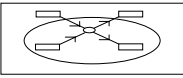
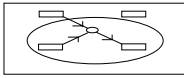
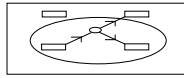
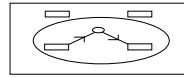
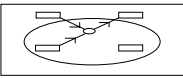
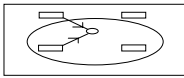
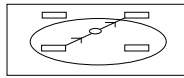
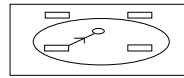
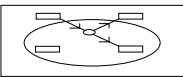
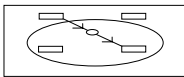
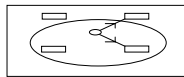
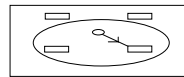
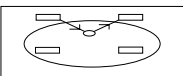
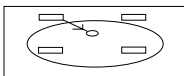
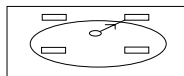
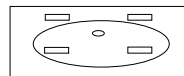
		reads outside B?	yes	yes	no	no
		writes outside B?	yes	no	yes	no
reads inside B?	writes inside B?					
yes	yes					
			DOUBLE (8)	INPUT (5)	OUTPUT (6)	NIL (1)
yes	no					
			INQUIRY (7)	NIL (4)	OUTPUT (6)	NIL (1)
no	yes					
			INPUT (7)	INPUT (5)	NIL (3)	NIL (1)
no	no					
			NIL (2)	NIL (2)	NIL (2)	NIL(1)

Table 13: Final Classification of Processes using Unified Rules

Thus 8 rules cover all cases. Rules 1-6 are very straightforward. This leaves just rule 7 which says that an inquiry must not write an ILF and rule 8 on the double process.

Table 9 shows the classifications by the CPM directly, table 13 that by the rules 1-8 of the *unified* scheme defined in tables 10 and 12. The two classifications are identical but the unified scheme follows fewer, simpler rules that allow the reader to see the reasons behind most of the rules. It thus lays extra stress on the two exceptions, namely the double process and the rule that an inquiry is not allowed to write to an ILF.

2.8 Unique Processes and Files

2.8.1 Unique Processes

The CPM explains (6.7.5) how to discover if an input process is unique.

For the identified process one of the following two rules must apply

- ◇ processing logic is unique from other external inputs for the application
- ◇ the data elements identified are different from other external inputs to the application

This may be formalized for computation as follows. Every external input process is described by a triple (N,L,P) where

- ◇ N is the name of the process
- ◇ L is a sorted list of its input triples
- ◇ P is a description of the processing it carries out

Now suppose there are two processes $(N1, L1, P1)$ and $(N2, L2, P2)$ with the same list $(L1=L2)$ and the same description $(P1=P2)$. We say that $N1$ and $N2$ are equivalent processes and count only one of them.

Figure 13 shows an example.

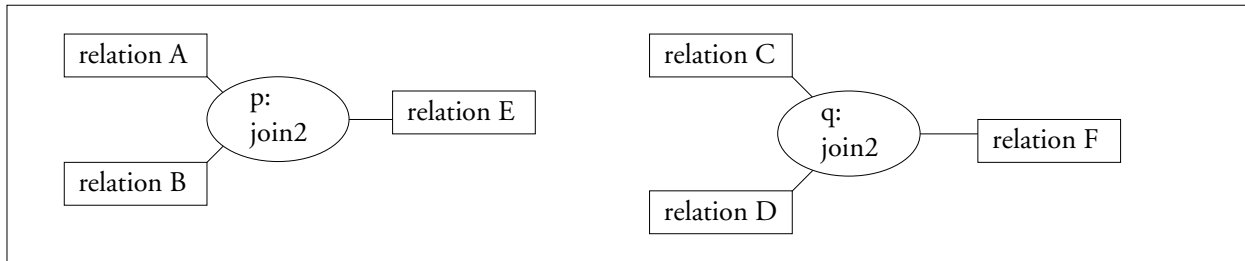


Figure 13: Equivalent processes

Although processes p and q have different names and operate on differently named inputs to yield differently named outputs, they are "really" the same process. Each one is described by the triple (I, P, O) where

$I = (\text{relation}, \text{relation})$
 $P = \text{join2}$
 $O = \text{relation}$

This example shows that to determine if two processes are equivalent demands in general that we find whether two text strings describe the same processing. This is an unsolved problem in computer science, which is why it is left to the user to resolve.

2.8.2 Unique Files

Although the CPM does not talk about *unique files* it explains that the user should count *types of file* rather than files themselves; thus if there are 12 files with names "Jan", "Feb" and so on for the monthly sales figures, these should be counted as one file type.

This approach can be formalized as follows. Every file is described by a pair (N, L) where

- ◊ N is its name
- ◊ L is a sorted list of its record element types

Suppose that there are two files $(N1, L1)$ and $(N2, L2)$ with the same list $(L1=L2)$. We say $N1$ and $N2$ are equivalent files and count only one of them. Thus in the example above, if the file for the annual sales figures uses *exactly* the same record element types as the file for the monthly sales figures then only one of them will be counted. This is reasonable; presumably the reason for counting a file is *eventually* to ensure that the productivity model (Abran 1994) allows for the effort used in defining the structure of the file itself; if two files have the same structure then only one piece of work is involved in such a definition.

2.9 Defining the count

2.9.1 Raw Material for counting at a file

Each file has a count, obtained by arithmetical operations on the number of data element types and the number of record element types as defined below:

A data element type (DET) is a unique user-recognizable, nonrecursive field on the ILF or EIF.

A record element type (RET) is a user-recognizable subgroup of data elements within an ILF or EIF.

These will be carried forward as $dets(f)$ and $rets(f)$ for file f .

2.9.2 Raw Material for counting at a process

Each process has a count, obtained by arithmetical operations on the number of file type references (FTRs) and the number of data element types (DETs) as defined below. Unfortunately

- ◇ the DET as defined for a process is different from the DET as defined for a file
- ◇ the DET is defined differently for each different type of process

Here are the three definitions of DET, laid out to emphasize differences.

<i>For an External Input a DET is.</i>	<i>For an External Output a DET is</i>	<i>For an External Inquiry a DET is</i>
a unique user recognizable nonrecursive field maintained on an internal logical file by the external input	a user recognizable nonrecursive field that appears on the external output	a user recognizable nonrecursive field that appears in the external inquiry

Table 14: Data Element Type definitions for Processes

1. Note that for an external input the field has to be unique, whereas for the other two it does not. Does this have any significance?
2. For the external input the word "maintains" is used whereas for the other two the word is "appears". Is "appears" being used as a synonym for "referred to"?

There is a difference between a DET for a file and a DET for a process; a DET for a file is what is implied by the words whereas a DET for a process is really a data element type that **appeared** (in the case of an output or as an inquiry) or that **was maintained** (in the case of an input). We shall therefore use the word *involved* and call this quantity a *data element type involved* or DETI, with the understanding that the definition of a DETI is contained in table 14 above.

A *file type referenced* is also a different thing depending on the type of the process. Here are the definitions:

<i>For an External Input a file type referenced is one of the two ...</i>		<i>For an External Output a file type referenced is ...</i>	<i>For an External Inquiry a file type referenced is ...</i>
an internal logical file read or maintained by a process	an external interface file read by a process	a file read when the external output is processed	a file read when the external inquiry is processed

Table 15: File Type Reference definitions

Notice that every *read* operation (reference to) is counted but the only *write* operation counted is the one that takes place when an external input writes to an ILF.

For a process we carry forward the number of file types referenced as $ftrs(p)$ and the number of data element types involved as $deti(p)$.

2.9.3 The count at an item

The count at any individual process or store is defined in terms of a table of the form shown in figure 10 below.

	$0 \leq x < x_1$	$x_1 \leq x < x_2$	$x_2 \leq x$
$0 \leq y < y_1$	L	L	M
$y_1 \leq y < y_2$	L	M	H
$y_2 \leq y$	M	H	H

literal	numerical
L	<i>low</i>
M	<i>medium</i>
H	<i>high</i>

Figure 14: Look up tables

where

- ◇ the non-negative integers x and y may be called the *inputs*
- ◇ the positive integers x_1, x_2, y_1 and y_2 may be called the *thresholds*
- ◇ the positive integers *low, medium, high* may be called the *weights*

Thus, the input x and the thresholds x_1 and x_2 together determine one column of the left hand table; likewise the input y and the thresholds y_1 and y_2 together determine one row of that table; the cell selected yields a literal value of L or M or H . This literal value determines one row of the right hand table and hence determines one of the three integral weights {*low, medium high*}. The whole operation can be written as the function

$$count = lookUp(x, y, x_1, x_2, y_1, y_2, low, medium, high)$$

with nine integral arguments and one integral value. By varying the identity of the arguments x, y sent in, the thresholds (x_1, x_2, y_1 and y_2) and the weights (*low, medium, high*), the CPM arranges specialist treatment for the two types of file and the three types of process.

Thus

$$\begin{aligned}
 c(f;ILF) &= lookUp(dets(f), rets(f), 19, 51, 1, 6, 7, 10, 15) \\
 c(f;EIF) &= lookUp(dets(f), rets(f), 19, 51, 1, 6, 5, 7, 10) \\
 c(p;EI) &= lookUp(deti(p), ftrs(p), 4, 16, 1, 3, 3, 4, 6) \\
 c(p;EO) &= lookUp(deti(p), ftrs(p), 5, 20, 1, 4, 4, 5, 7)
 \end{aligned}$$

The treatment of an external inquiry is slightly more involved. If q is an EQ then the CPM suggests that

- ◇ q has an input side, $q(1)$ say,
- ◇ q has an output side, $q(2)$ say
- ◇ $q(1)$ is an EI and as such has a count as an EI; let this be denoted v_1
- ◇ $q(2)$ is an EO and as such has a count as an EO; let this be denoted v_2
- ◇ the count for q is defined as $\max(v_1, v_2)$

The CPM does not state how to define the input side and the output side of an external inquiry and this issue remains to be resolved.

2.9.4 The count for the enhancement

As explained in 2.6, every changed process or file is in one of the sets ADDED, MODIFIED, USED FOR CONVERSION or DELETED, abbreviated to A,M,C,D respectively; as explained in 2.7 every process or file is also in one of the sets (ILF, EIF, IGF, EI, EO, EQ, NIL). Table 16 arranges those items in an appropriate spreadsheet notation. The upper left hand cell is for the items to be added of the type ILF.

The contribution to the function point count is therefore $\sum_{ILF \cap A} c(f;ILF)$ as shown. Adding the entries

down the columns leads to the partial sums, ADD, CHGA, CFP and DEL.

	to be added	to be modified	to be used for conversion	to be deleted	to be unchanged
ILF	$\sum_{ILF \cap A} c(f; ILF)$	$\sum_{ILF \cap M} c(f; ILF)$	$\sum_{ILF \cap C} c(f; ILF)$	$\sum_{ILF \cap D} c(f; ILF)$	0
EIF	$\sum_{EIF \cap A} c(f; EIF)$	$\sum_{EIF \cap M} c(f; EIF)$	$\sum_{EIF \cap C} c(f; EIF)$	$\sum_{EIF \cap D} c(f; EIF)$	0
IGF	0	0	0	0	0
EI	$\sum_{EI \cap A} c(p; EI)$	$\sum_{EI \cap M} c(p; EI)$	$\sum_{EI \cap C} c(p; EI)$	$\sum_{EI \cap D} c(p; EI)$	0
EO	$\sum_{EO \cap A} c(p; EO)$	$\sum_{EO \cap M} c(p; EO)$	$\sum_{EO \cap C} c(p; EO)$	$\sum_{EO \cap D} c(p; EO)$	0
EQ	$\sum_{EQ \cap A} c(p; EQ)$	$\sum_{EQ \cap M} c(p; EQ)$	$\sum_{EQ \cap C} c(p; EQ)$	$\sum_{EQ \cap D} c(p; EQ)$	0
NIL	0	0	0	0	0
	ADD	CHGA	CFP	DEL	0

Table 16: Interim Function Point Counts

and these lead to the function point count for the enhancement as

$$EFP = [(ADD + CHGA + CFP) * VAFA] + (DEL * VAFB)$$

where the factors VAFA and VAFB will be derived from values supplied by the user (see steps 15, 16 in section 3).

2.10 A Second Notation for Design

Since these counting functions refer not only to files but also to records within files and fields within records, the notation of section 2.3 must be extended. One possibility is that of figure 15.

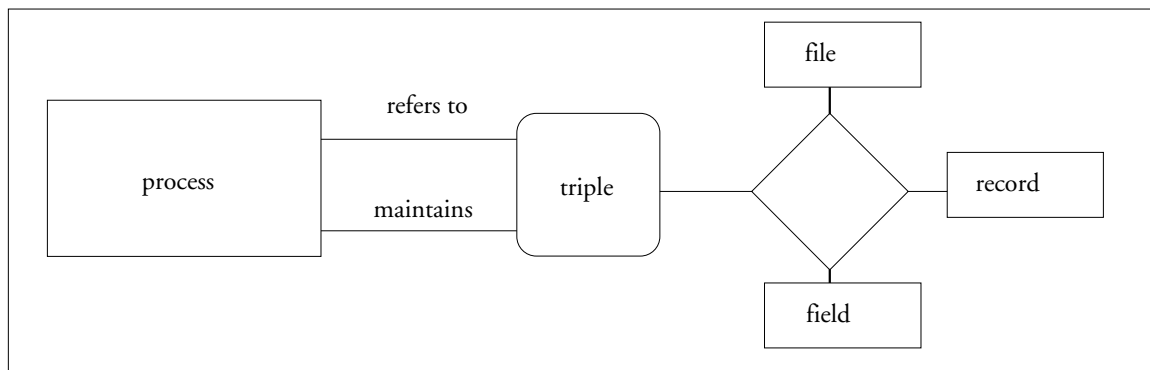


Figure 15: Schema for design suitable for counting function points

File	this is a logical file as defined by the CPM;
Record	this is a record element type as defined by the CPM
Field	this is a data element type as defined by the CPM
Triple	(file, record, field)
Process	this is a process as defined by the CPM

Table 17: Entities for figure 15

refers to	this is reads in common parlance
maintains	this is writes in common parlance

Table 18: Relations for figure 15

3 Formal procedure for Counting Function Points

The whole task of measuring function points for an enhancement can now be expressed as two major tasks

1. Capture the Designs in the notation of section 2.9.
2. Count the Function Points

Since this paper is about the methodology of FPA, task 1 is treated in appendix A2. The activities in major task 2 are as follows:

3.1 Activities in Counting Function Points

1	Name the original system, the enhanced system and the conversion system	The user must specify which enhancement is to be measured.
2	Compute the status for each file	Use the logic of 2.4.1 to select one of (added, modified, deleted, unchanged); then use 2.5 to select one of (inside B, outside B)
3	Compute the status for each process	Use the logic of 2.4.2 to select one of (added, modified, deleted, unchanged); then use 2.5 to select one of (inside B, outside B)
4	Decide which processes are required by this application	the CPM states that the processes to be considered are those recognized by the user as required by the application. This step offers the user the opportunity to discard a process because it is not required ; this must be the user's decision but the principles of consistency and good record-keeping suggest that the user write down the reason why the process is not considered a requirement of the application.
5	Decide which files are required by this application	The same logic applies to files as to processes.
6	Compute the attribute type for each relevant file	Use the logic of 2.6 to select one of (ILF, EIF, IGF)
7	Compute the attribute type for each relevant process	Use the logic of 2.7 to select one of (EI, EO, EQ, NIL)
8	Decide which relevant processes are unique	Use the logic of 2.8.1 to select one of (unique, duplicate)
9	Decide which relevant files are unique	Use the logic of 2.8.2 to select one of (unique, duplicate)

10	Compute the attributes frs and dets for each unique relevant process	Use the logic described in 2.9.1 to derive these attributes from the design itself
11	Compute the attributes rets and dets for each unique relevant, file	Use the logic described in 2.9.1 to derive these attributes from the design itself
12	Compute the attribute count for each unique, relevant process	Use the logic described in 2.9.2
13	Compute the attribute count for each unique relevant file	Use the logic described in 2.9.2
14	Compute the unadjusted function points ADD, CHGA, CFP, DEL for the parts added, modified, used for conversion and deleted	Use the logic described in 2.9.3
15	Decide on the values of the 14 attributes in the group general characteristic for the systems both before and after the enhancement	This information lies in the constraints, not the design, and must therefore be entered by the user.
16	Compute the value adjustment factors VAFB, VAFA before and after the enhancement	Use the logic described in 2.2
17	Compute the attribute adjusted count for the enhancement	Use the logic described in 2.2, namely $EFP = [(ADD + CHGA + CFP) * VAFA] + (DEL * VAFB)$

Table 19: Activities in Counting Function Points

An activity starting "Decide" indicates one where the decision must be taken by the user; one starting "Compute" indicates an algorithmic activity carried out by the tool. In the detailed analysis it has been shown *why* the "decide" activities can not in principle be carried out by the tool and *how* the "Compute" activities can be carried out by the tool. At this stage it is sufficient to note that with five "Decide" activities and 11 "Compute" activities the task is clearly one that calls for a co-operation between user and machine. If the tool were not available the user would have to do the 11 "compute" activities manually; if the user were not available the five "Decide" activities would be impossible to carry out.

3.2 Precedence Analysis

The list above assigns values to the attributes shown in table 11 and does so in a feasible order; i.e. each attribute can be assigned a value because the attributes on which it depends are already available.

The table below shows which attributes are read and written by the 17 tasks. Each row represents an attribute and each column a task; the minus symbol indicates that the attribute is an input to the task, the plus symbol that the attribute is an output from the task

Attribute	Task																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
name of original system	+	-	-	-	-	-	-	-	-	-	-						
name of enhanced system	+	-	-	-	-	-	-	-	-	-	-						
name of conversion system	+	-	-	-	-	-	-	-	-	-	-						
Status(file)		+		-	-	-	-	-	-								
Status(process)			+	-	-	-	-	-	-								
Requiredness(process)				+													
Requiredness(file)					+												
type(file)						+											
type(process)							+										
uniqueness(process)								+									
uniqueness(file)									+								
fters(process) dets(process)										+		-					
rets(file) dets(file)											+		-				
count(process)												+		-			
count(file)													+	-			
ADD, CHGA CFP, DEL														+			-
general characteristics before and after															+	-	
VAFA, VAFB																+	-
EFP																	+

Table 20: Precedence matrix for tasks and attributes

The tasks are ordered from left to right in order of execution; the attributes are ordered from top to bottom in order of creation. Note that every row must have a plus on the left to indicate that the attribute is written before being read. Every column must have all the minuses above all the plusses to indicate that the input data is available when the task is executed. Tasks 1 and 15 have no minus signs; this indicates that the user supplies this information by consulting information in some place not specified here.

3.3 Traceability

The table below provides a two-way index between the 17 tasks given above and the sections of the CPM. Thus the row "identify boundary" indicates that identify boundary supports tasks 2 and 3; the column 4 indicates that task 4 is supported by the CPM sections EI definition, EO definition and EQ definition.

the CPM page theme	Task as given in this text																
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
3 type of count	√																
4 identify boundary		√	√														
5-6 ILF identification					√	√											
5-6 EIF identification					√	√											
5-7 ILF counting rules									√		√		√				
5-7 EIF counting rules									√		√		√				
6-4 EI definition				√			√										
6-6 EI counting rules								√		√		√					
6-16 EO definition				√			√										
6-6 EO counting rules								√		√		√					
6-28 EQ definition				√			√										
6-30 EQ counting rules								√		√		√					
7-3 VAF calculation															√	√	
8-10 Enhancement Project FP calculation														√			√

Table 21: Traceability between the CPM and this text.

3.4 The notation extended for Function Points

To count function points for an enhancement expressed in the notation of figure 15, it is sufficient to introduce one new entity the enhancement and to add attributes to the process and the file defined in figure 15. The new attributes are shown in the tables below.

Status	indicates whether the process is to be in ADD, CHGA, MOD, DEL or UNCHANGED
Insideness	indicates whether the process is to be inside or outside the boundary B
Requiredness	indicates whether the process is required by the user
Type	indicates whether the process is in EI, EO, EQ or NIL
Uniqueness	indicates whether the process is unique in the enhancement
Ftrs	the number of file transfers made by the process
Deti	the number of data element transfers made by the process
Count	the count for this process

Table 22: Process Attributes

Status	indicates whether the file is to be in ADD, CHGA, MOD, DEL or UNCHANGED
Insideness	indicates whether the file is to be inside or outside the boundary B
Requiredness	indicates whether the file is required by the user

type	indicates whether the file is in ILF, EIF or IGF
uniqueness	indicates whether the file is unique in the enhancement
rets	the number of record element types contained in this file
dets	the number of data element types contained in this file
count	the count for this file

Table 23: File Attributes

original application	
enhanced application	
conversion application	
ADD	raw function points added
CHGA	raw function points for modified items
CFP	raw function points for conversion
DEL	raw function points for parts deleted
general characteristics before	14 values each in [0,5]
general characteristics after	14 values each in [0,5]
VAFB	value adjustment factor before enhancement
VAFA	value adjustment factor after enhancement
EFP	enhancement function point count

Table 24: Enhancement Attributes

Accordingly the task of calculating function points can be expressed as that of assigning values to the attributes in the tables above.

4 Conclusion

Section 2 transformed the relevant parts of the CPM into processing logic suitable for the parts of a tool to count function points. Section 3.1 showed how to break down the activity of counting function points into 17 tasks, each labelled as algorithmic or requiring judgment. Section 3.2 showed that these tasks could be carried out in a feasible order, i.e. when a task is required to run its input data is available to it. Section 3.3 traced the 17 tasks back to the CPM and section 3.4 defined the extension to the data model needed to support function point counting.

It has therefore been shown that the activity of counting Function Points can be regarded as 17 sub-activities, six of which require human judgment and 11 of which can be automated. It is therefore concluded that no automatic tool for counting Function Points is at present possible. However there is scope for an interactive tool that will carry out book-keeping and arithmetic while allowing the user to exercise the human judgment required.

5 Future Directions

The analysis of this paper suggests the following four applications:

1. Building an interactive tool based on CPM 4.0
2. Reviewing existing tools based on CPM 4.0
3. Extending the rules to include uncodified practice used by expert counters.
4. Building a tool with which a standards body can investigate the effect of changing the rules

5.1 Building an interactive tool based on CPM 4.0

The formalism defined in this paper can be used as the basis for an interactive tool based on CPM 4.0. Such a tool would carry out the 12 algorithmic steps of FPA and thus allow its user to concentrate on the five interactive steps that require human judgment.

5.2 Reviewing existing tools based on CPM 4.0

This paper breaks FPA into 17 steps of which 12 are algorithmic and five are interactive. This ratio can guide us when we review existing tools for FPA; it prompts us to wonder why some tools can claim to do the whole process automatically whereas others rely on the user for much of the arithmetic.

Suppose that we want to review some existing tool, T say, that is based on CPM 4.0. If we find that T claims to do the whole process of FPA automatically, we shall be tempted to ask

How can the tool T provide fully automatic FPA when the analysis of this paper shows that five out of 17 sets remain subject to human judgment? Is not the tool T promising too much?

If we find that T expects the user to carry out three-quarters of the work himself, we shall be tempted to ask

Why does the tool T leave so much to the user when the analysis of this paper shows that twelve of the 17 steps can be executed by algorithms? Is not the tool T delivering too little?

Neither question would have occurred without an analysis such as that of this paper.

5.3 Extending the rules to include uncodified practice used by expert counters.

Expert counters believe that counting Function Points depends not only on the rules of the CPM but also on what we may call "uncodified practice", pieces of expertise developed over the years and not written down anywhere. The analysis of this paper makes it possible to search more easily for such uncodified practice, with the aim of eventually codifying it for inclusion in a future version of the CPM.

5.4 Building a tool for a standards body

We may distinguish between the *rule-users*, those who count function points to measure the size of a software system and the *rule-makers*, those who devise the rules on which FPA is based.

The rule-users count with the most recent CPM, supplemented by some version of the "uncodified practice" referred to in 5.3 above.

The rule-makers count with any version of the rules that they consider to be worthy of study. This is because the rules of Function Points are subject to change and those responsible for changing them wish to know the effect of a change before proposing that change for wider discussion.

At present, it is hard work to discover the effect of a change in the rules; this suggests that a tool should be built especially for the rule-makers; such a tool would have the rules of Function Points coded as an external rule base (Davis and Lenat, 1982), easily accessible to the rule-maker and easily changeable. The rule-maker could then investigate the effect of any proposed change in the rules by altering the rule base and running the tool to measure a standard set of designs.

A tool such as this needs a disciplined expression of the rules, such as presented in this paper.

6 References

- ◇ Abran, A. (1994) *Analyse du Processus de Mesure des Points de Fonction*, (Ph D thesis) University of Montreal.
- ◇ Albrecht A.J. (1979) *Measuring Application Development Productivity*, Proceedings IBM Applications Development Symposium, Monterey, CA.
- ◇ Card DN and Glass RL (1991) *Measuring Software Design Quality*, Prentice Hall, Englewood Cliffs.
- ◇ Davis R and Lenat DB (1982) *Knowledge-based Systems in Artificial Intelligence*, McGraw-Hill, New York.
- ◇ DeMarco T. (1979) *Structured System Specification* Yourdon Press, New York
- ◇ IFPUG (1994) *Counting Practices Manual 4.0*, International Function Points Users Group
- ◇ Kemerer C.F. (1987) *An Empirical Validation of Software Cost Estimation Models*, CACM, **30**, 5, 416-429.
- ◇ Low G.C., Jeffery, DR. (1990) *Function Points in the Estimation and Evaluation of the Software Process*, IEEE Transactions on Software Engineering, **SE-16**, 1, 64-71.
- ◇ McMenamin S.J. and Palmer J.F. (1981) *Essential System Design*, Yourdon Press, New York
- ◇ Meyerhoff D and Mullerburg M. (1992) *Set Structured Hypertext Applied to Software Measurement*, IFIP Congress 1992, Madrid.
- ◇ Rudolph E.E. (1989) *Precision of Function Point Counts*, IFPUG Spring Conference, San Diego, CA.
- ◇ St-Pierre D (1993) *Hypertexte structure applique aux points de fonction*, Centre d'interets sur les metriques, Montreal.

Appendix A

A1 Geometrical and Set-Theoretical Language for defining a boundary

The following table shows some of the correspondences between the geometrical language of boundary, inside and outside and the set-theoretical language of distinguished set, belongs and does not belong.

Let B denote the boundary used by the CPM, I a distinguished subset and E the complement of I . The correspondence is as follows:

<i>Notion in geometrical language</i>	<i>Same notion in set-theoretical language</i>
p is inside B	p is in I
f is inside B	f is in I
file f is maintained by a process inside B	there is a p in I such that p writes f
file f is maintained by a process outside B	there is a p in E such that p writes f
file f is referred to by a process inside B	there is a p in I such that p reads f
file f is referred to by a process outside B	there is a p in E such that p reads f
file f is an ILF for B	f is in I
file f is an EIF for B	f is in E <i>and</i> some p in I reads f <i>or</i> some p in I writes f
p is an EQ for B when and only when 1. p refers to an f outside B 2. p maintains an f outside B 3. p refers to some f inside B 4. p maintains no f inside B	p is an EQ for I when and only when 1. p reads some f in E 2. p writes some f in E 3. p reads some f in I 4. p writes no f in I
p is an EO for B when and only when 1. p refers to no f outside B 2. p maintains some f outside B 3. p refers to some f inside B	p is an EO for I when and only when 1. p reads no f in E 2. p writes some f in E 3. p reads some f in I
p is an EI for B when and only when 1. p refers to some f outside B 2. p maintains no f outside B 3. p maintains some f inside B	p is an EI for I when and only when 1. p reads some f in E 2. p writes no f in E 3. p writes some f in I

Table 25 : Correspondence between geometrical and set-theoretical language

A2 The Activities in Capturing a Design

The activities in capturing the design are as follows:

1. Enter *files* by name
2. Enter *records* by name
3. Enter *fields* by name
4. Enter *processes* by name
5. Enter the relation RISIN(records, files)
6. Enter the relation FISIN(fields, records)
7. Enter *triples*
8. Enter *questions* by name
9. Enter *answers* by name
10. Enter the relation READS(processes, triples)
11. Enter the relation WRITES(processes, triples)
12. Enter the relation RECEIVES(processes, questions)
13. Enter the relation RECEIVES(processes, answers)
14. Enter the relation SENDS(processes, questions)
15. Enter the relation SENDS(processes, answers)

Each activity starts with the word "enter" indicating that this is a matter of entering data. Although the tool can make it a little easier to enter the data the responsibility for entering the design is primarily that of the user of the tool. For example every "enter ... by name" activity demands that the user think of a name because there is at this point no repertoire of names from which to select. Every "enter the relation ..." activity is a little easier because the sets on which the relation is to be entered already exist; hence the tool can supply these sets and invite the user to enter the relation by checking and unchecking boxes in a manner familiar to users of windows of all varieties.

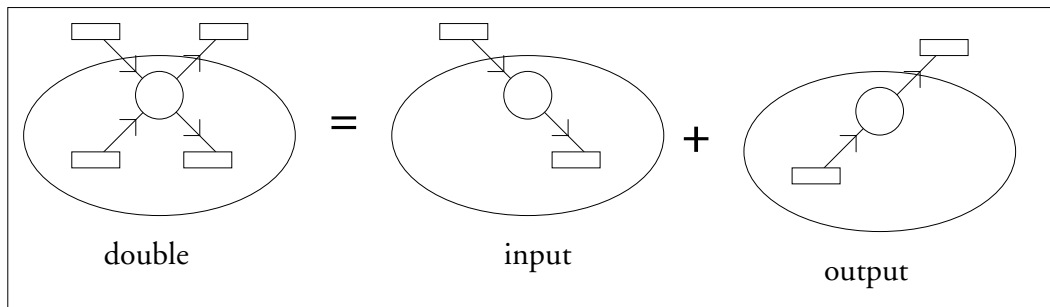
Appendix B

The main text of the paper explains FP as defined in the CPM. These appendices describe several points that do not appear in the CPM. Topics discussed include

- ◇ Double processes
- ◇ Displays and transient data
- ◇ Re-use
- ◇ Decomposing an inquiry

B1 Double process

A process that reads and writes both inside B and outside B has been explained as a double process, with this interpretation:



B2 Displays, persistent and transient data

When the CPM describes the input process and the output process it does so in terms of maintaining files and referring to them; when it describes the inquiry it does so in terms of an input request and an output, thus

EQ	Comment
An input request enters the application boundary	p reads a request is this a file?
Output results exit the application boundary	p writes some output is this to a file?
Data is retrieved	p reads an ILF of A
The retrieved data does not contain derived data	The data retrieved was planted there by an EI, rather than by some internal process?
The input request and output results together make up a process that is the smallest unit of activity that is meaningful to the end user of the business	The process is atomic, unlike a double process (q.v.)
The process is self-contained and leaves the business of the application being counted in a consistent state	Does it? It leaves an answer on some screen but not in a file?
The processing does not update an ILF	p writes no ILF of A
For the identified process one of the following two rules must apply <ul style="list-style-type: none"> ◇ processing logic on the input or output side is unique from other external inquiries in the application ◇ the data elements making up the input or output side are different from other external inquiries in the application 	p is described by the pair (I,P,O) <ul style="list-style-type: none"> ◇ I=a sorted list of its input triples ◇ O=a sorted list of its output triples ◇ P=a description of its processing If there is a p' also in the application that also has the description (I,P,O) then p and p' are said to be duplicates and only one is counted.

This suggests that the inquiry works like this.

There is a *questioning* process that generates a *question* as transient data; there is an *answering* process, namely the external inquiry itself, that reads the question from this transient data, reads an ILF and displays the *answer* as transient data, presumably on a screen. When the *questioning* process has read the answer from the screen the answer vanishes from the screen.

This raises these questions.

- ◇ Is the question a file?
- ◇ Is the answer a file?
- ◇ When the inquiry has taken place, what files have changed?
- ◇ If no files have changed has anything taken place at all?

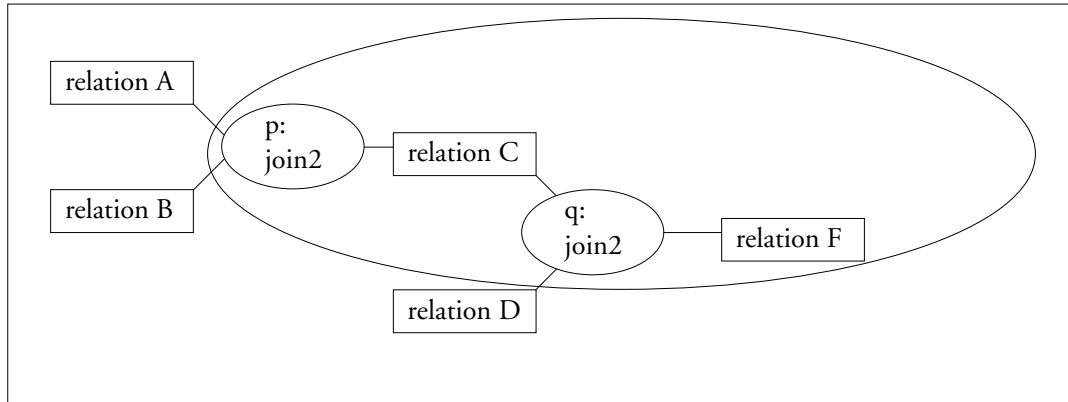
This can be modelled with the mechanism in the paper with just one extension. Data is now of two sorts

- persistent data** This data stays until it is over-written with a fresh value; it is written once and can be read many times.
- transient data** This data stays until it has been read once; it is written once and can be read once; if you want to save such data you have to move it immediately into a file of permanent data.

When an inquiry reads outside B it reads transient data and when it writes data outside B it writes transient data. This mechanism puts the distinction in the data rather than the read; it also means that a display is just a write of transient data.

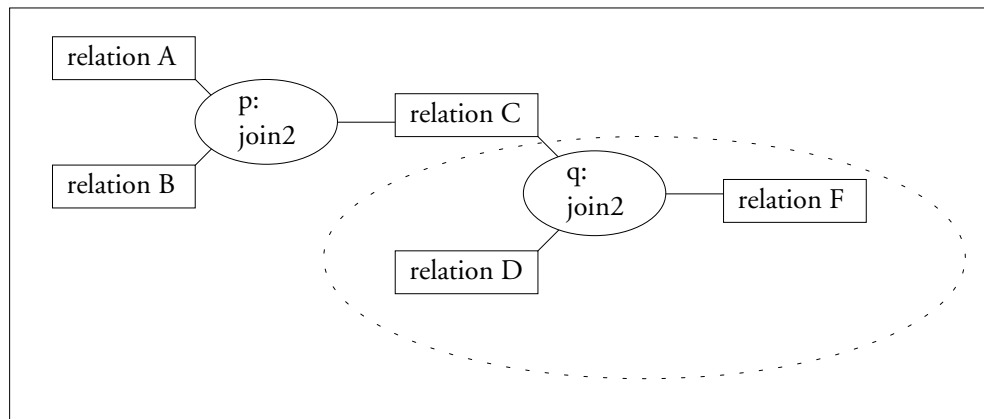
B3 Re-use

FP handles uniqueness and re-use as two sides of the same coin. If a process is unique, then it has to be built; if it is not unique then we can re-use the duplicate we just found. Unfortunately FP does not handle re-use effectively. Here is how it handles re-use within an enhancement.



The processes p and q are both within the enhancement; FP will detect that they are two versions of the same process and count one process.

Now suppose that the join2 process had already been written in the existing system thus:



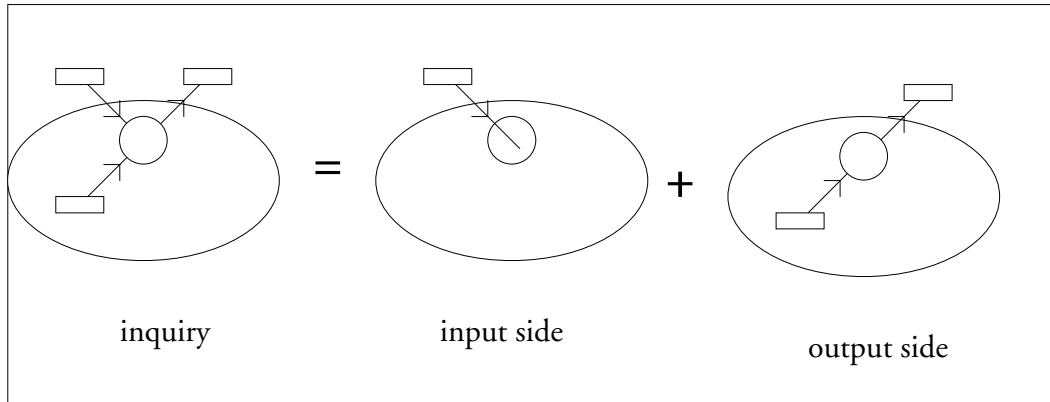
The process p is in the existing system and the process q is in the enhancement. FP will not look in the existing application and it will therefore count q as one process. Hence the measured size will be too big by the size of process q.

This is a serious flaw; when judging that a process is unique FP should be forced to look at all processes in all applications available to the builder.

The solution is to put all the processes of the existing application in a repository; each process is described by its (I,P,O) triple where I is a sorted list of inputs. P describes the processing and O a sorted list of outputs. The repository must be stored in such a way that if you have a new process (I',P',O') you can quickly see all the processes (I,P,O) for which I=I' and O=O'. You can then judge whether P' is equivalent to P, a decision that cannot be undertaken by computer.

B4 Decomposing an inquiry

The CPM talks about the input side and the output side of an inquiry. Is this what is meant?



A7 Reviewer's Comment Form

Dear Reviewer,

Please use this form to communicate comments about the paper. The section below is meant only as a guide; please feel free to add comments on any other aspects that occur to you.

A7.1 Summary of the Paper:

Is the summary clear? Does the summary accurately describe the content? if not, in what way is it inaccurate?

A7.2 Accuracy

Do the main text and the appendices jointly cover the aim as stated in the summary? If not, what would you like to see added?

A7.3 Completeness:

Is there any part of counting function points not covered in the 17 points?

A7.4 Subdivision into Tasks

Is the subdivision of FP into 17 tasks as in the paper appropriate? If not, how would you subdivide it?

A7.5 Classification of Tasks:

The table below reproduces the paper's classification of tasks as interactive or algorithmic. Do you agree with this classification? If not, where and why not?

	Task	class	agree ✓ or disagree × if you disagree please say why
2	Compute the status for each file	algorithmic	
3	Compute the status for each process	algorithmic	
4	Decide which processes are required by this application	interactive	
5	Decide which files are required by this application	interactive	
6	Compute the attribute type for each relevant file	algorithmic	
7	Compute the attribute type for each relevant process	algorithmic	
8	Decide which relevant processes are unique	interactive	
9	Decide which relevant files are unique	interactive	
10	Compute the attributes ftrs and dets for each unique relevant process	algorithmic	
11	Compute the attributes rets and dets for each unique, relevant, file	algorithmic	
12	Compute the attribute count for each unique, relevant process	algorithmic	
13	Compute the attribute count for each unique, relevant file	algorithmic	
14	Compute the unadjusted function points ADD, CHGA, CFP, DEL for the parts added, modified, used for conversion and deleted	algorithmic	
15	Decide on the values of the 14 attributes in the group general characteristic for the systems both before and after the enhancement	interactive	
16	Compute the value adjustment factors VAFB, VAFA before and after the enhancement	algorithmic	
17	Compute the attribute adjusted count for the enhancement	algorithmic	

A7.6 Other

Please use this space for any other comments.

Thank you for your help.

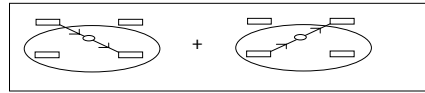
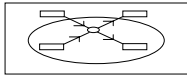
Keith Paton and Alain Abran

the end

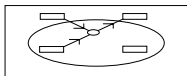
- A process must read or write outside B
- A process must read or write inside B
- A process must read somewhere
- A process must write somewhere
- A process must not read and write both inside and outside

The ones ambiguous so far are those that read outside B and write outside B. Figure xx suggests one way of resolving the ambiguities.

Case Suggested Resolution
This is a *double process* and can be written as the sum of two processes

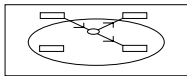


EI or EO?

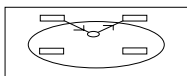


This is an EQ

EO or EQ?



EI or EO?



This is strange; the process neither reads from nor writes to an ILF. In this case, what is the purpose of surrounding the process by a boundary? This case will be called a NIL.

EO or EQ?

It is possible that these ambiguities can be resolved using the rules

The process is the smallest unit of activity that is meaningful to the end user of the business.

This suggests that an ambiguous process can be expressed as the union of two processes that are themselves elementary in the sense above.

a process that reads outside B and writes outside B

can be

he rules of based on the rules of the the CPM.

If the processes are classified From the point of view of computation we
 We now show that the rules allow a proces to be recognized as both an external input and an external output. Consider the rules for identifying external inputs and external outputs reproduced in table xx below.

<i>External Input</i>	<i>External Output</i>
The data is received from outside the application boundary	The process sends data or control information external to the application's boundary
The data in an ILF is maintained through an elementary process of the application	The data or control information is sent through an elementary process of the application
The process is the smallest unit of activity that is meaningful to the end user in the business	
The process is self-contained and leaves the business in a consistent state	

Table xx : Rules for identifying an external input or an external output. A process atsofoes all rules if it

1. receives data from outside B
2. sends data outside B
3. maintains data in an ILF
4. is an elementary process, i.e. cannot be subdivided into processes
5. is self-contained.

Now consider the process in figure xx.

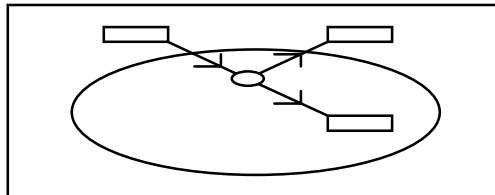


Figure xx: A process that is both an external input and an external output by the the CPM.

Obviously it satisfies 1-3 above. It cannot be subdivided into two processes, each of which is an EI, anEO or an EQ. Figure xx shows all possible attempts to subdivide it.

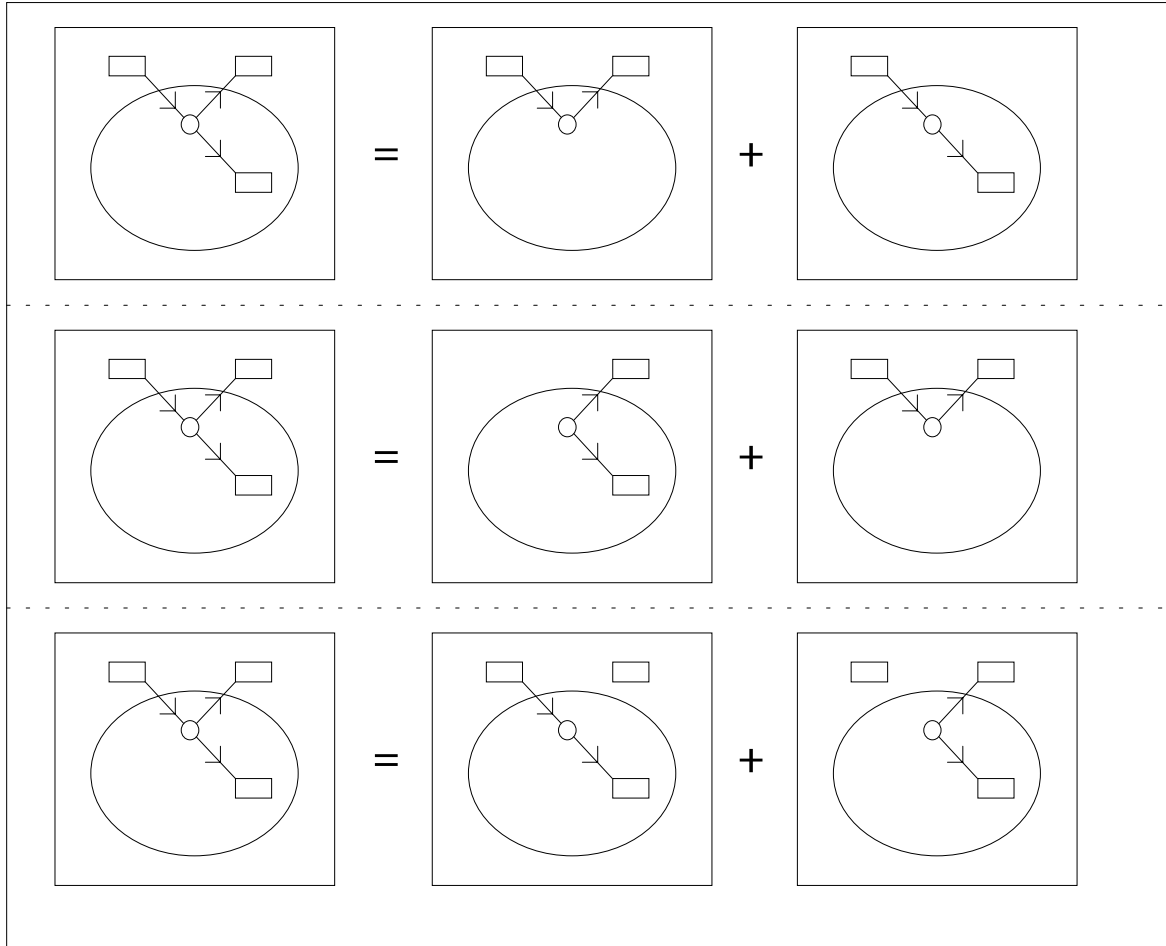


Figure xx: Attempted subdivisions of figure xx into elementary processes.

In attempt 1,

In attempt 2,

In attempt 3,

processes that are them; figure xx shows all possible ways of trying to subdivided

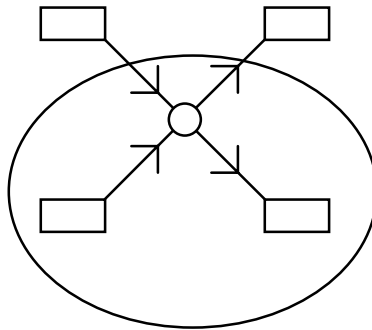
It receives data

It sends data outside

It maintains data in an ILF

ntary process

Consider the process in figure xx



We now show that it satisfies the rules for both an external input and an external output.

fail this test We show that the rules fail the r These rules are analuzdd below and prohere are two function types, the file and the transaction. The transaction corresponds to what we have called a process.

The the CPM deined two function types, files and transactions. It firther sdefi transaction type, which we are calling

Given the boundary B, the processes are classified by the CPM according to these rules.

We start by identifying 4 main cases. A process may read outside B or not and independently it may write outside B or not. We have these four cases:

	p reads outside B	p does NOT read outside B
p writes outside B	<p>p might be an inquiry</p>	<p>p might be an output</p>
p does not write outside B	<p>p might be an input</p>	<p>p can not be an external process</p>

Figure 9: Initial Classification of Processes

The the CPM ignores any process for which no data crosses the application boundary; thus the lower right pattern is not considered further. We take the three remaining cases one by one for simplicity and consider in each case whether the process reads inside B and whether it writes inside B

2.7.1 Possible External Inquiry

(The process reads outside B and writes outside B.)

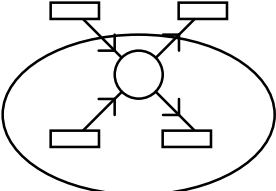
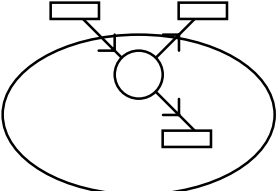
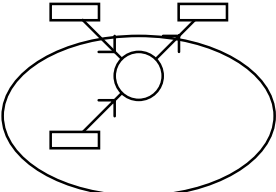
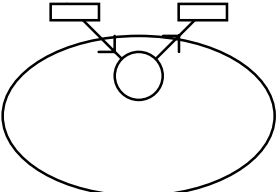
	p reads inside B	p does NOT read inside B
p writes inside B	 <p>not allowed (the processing does not update an ILF 6-31(7))</p>	 <p>not allowed (the processing does not update an ILF 6-31(7))</p>
p does NOT write inside B	 <p>legal</p>	 <p>not allowed (data must be retrieved 6-31(3))</p>

Figure 10: Classification of Possible External Inquiries

2.7.2 Possible External Output

(The process writes outside B but does not read outside B.)

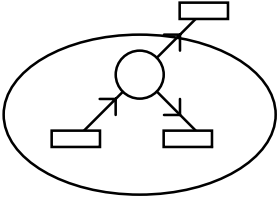
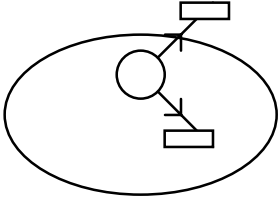
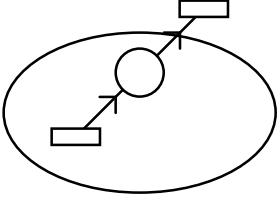
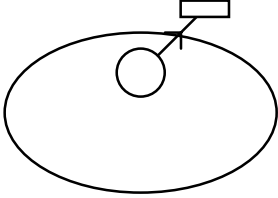
	p reads inside B	p does NOT read inside B
p writes inside B	 <p>legal</p>	 <p>not allowed (must read from somewhere)</p>
p does NOT write inside B	 <p>legal</p>	 <p>not allowed (must read from somewhere)</p>

Figure 11: Classification of Possible External Outputs

2.7.3 Possible External Input

(The process reads outside B but does not write outside B.)

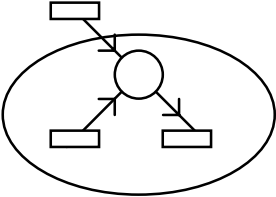
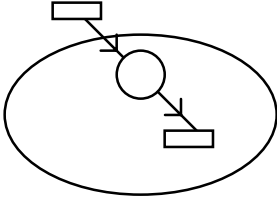
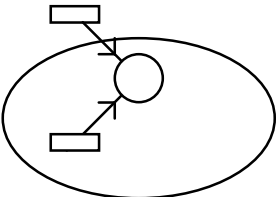
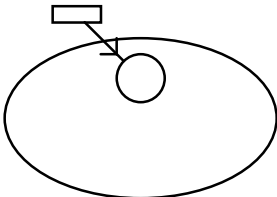
	p reads inside B	p does NOT read inside B
p writes inside B	 <p>legal</p>	 <p>legal</p>
p does NOT write inside B	 <p>not allowed (data in an ILF is maintained.. 6-7◇ 2)</p>	 <p>not allowed (data in an ILF is maintained.. 6-7◇ 2)</p>

Figure 12: Classification of Possible External Inputs

2.7.4 Overall classification of the processes

The the CPM starts with the notion of three types of external processes and provides three separate sets of rules for these three types. This has two major disadvantages

- ◇ in principle the processes defined by the rules might overlap; in this case the result would depend on the order in which the rules were tested
- ◇ from the point of view of computation we would like to ask "what is the type of this process?" rather than ask the three questions
 1. is process p an external input?
 2. is process p an external output?
 3. is process p an external inquiry?

The rules classifying the processes can be unified as follows: Suppose we have a candidate process and we want to know whether it is an input, an output an inquiry or none of the above. We can start by asking

- ◇ does it read outside the boundary of the application
- ◇ does it read inside the boundary of the application
- ◇ does it write outside the boundary of the application
- ◇ does it write inside the boundary of the application

We want to classify the candidate by applying rules to these answers. Such rules are suggested by the following arguments.

1. Suppose a process neither reads nor writes outside B. Then no data crosses B and the process loses its claim to be an "external" process.
2. Suppose a process neither reads nor writes inside B. Then at the end of the process the application has neither stored any new information nor retrieved any old information; it has therefore carried out no useful function.
3. Suppose a process reads neither inside nor outside B. Then it can receive no data to process and can do nothing.
4. Suppose a process writes neither inside nor outside B. Then at the end of the process no information has been sent anywhere so the processing was completely wasted.

These points suggest the following rules

1	A process must read or write outside B
2	A process must read or write inside B
3	A process must read somewhere
4	A process must read somewhere

Table 6: Four New Rules fro Classifying Processes

These four rules reject nine of the 16 candidates as shown in the table.

	reads outside writes outside	reads outside does not write outside	does not read outside writes outside	does not read outside does not write outside
reads inside writes inside				NO by rule 1
reads inside does not write inside		NO by rule 4		NO by rule 1
does not read inside writes inside			NO by rule 3	NO by rule 1
does not read inside does not write inside	NO by rule 2	NO by rule 2	NO by rule 2	NO by rule 1

Table 7: Classification using new rules 1-4

Now classify the six surviving cases using these rules:

5. A surviving process that reads outside B but does not write outside B is an INPUT
6. A surviving process that writes outside B but does not read outside B is an OUTPUT
7. A surviving process that reads and writes outside B is an INQUIRY if it reads inside B and NOT AN INQUIRY if it writes inside B.
8. A process that carries on all four activities is a DOUVLE PROCESS and must be decomposed.

	reads outside writes outside	reads outside does not write outside	does not read outside writes outside
Reads inside writes inside	(DOUBLE PROCESS rule 8	INPUT rule 5	OUTPUT rule 6
Reads inside does not write inside	INQUIRY rule 7	NO rule 4	OUTPUT rule 6
does not read inside writes inside	NOT AN INQUIRY rule 7	INPUT rule 5	NO rule 3

Table 8: Classification using new rules 1-8

Thus 8 rules cover all cases. Rules 1-6 are very straightforward. This leaves just and rule 7 which says that an inquiry must not write an ILF and rule 8 on the double process.

In this unified classification scheme we obtain the same classification as defined in the CPM by a route that allows the reader to see the reasons behind most of the rules. It thus lays extra stress on the two exceptions, namely the double process and the rule that an inquiry is not allowed to write to an ILF

6 Original Set of References

- ◇ Abran, A. (1994) *Analyse du Processus de Mesure des Points de Fonction*, (Ph D thesis) University of Montreal.
- ◇ Albrecht A.J. (1979) *Measuring Application Development Productivity*, Proceedings IBM Applications Development Symposium, Monterey, CA.
- ◇ Bachman (1994), Function Point Tool (proprietary to Bachman International)
- ◇ Card DN and Glass RL (1991) *Measuring Software Design Quality*, Prentice Hall, Englewood Cliffs.
- ◇ Davis R and Lenat DB (1982) *Knowledge-based Systems in Artificial Intelligence*, McGraw-Hill, New York.
- ◇ DeMarco T. (1979) *Structured System Specification* Yourdon Press, New York
- ◇ IFPUG (1994) *Counting Practices Manual 4.0*, International Function Points Users Group
- ◇ Kemerer C.F. (1987) *An Empirical Validation of Software Cost Estimation Models*, CACM, 30, 5, 416-429.
- ◇ Koch (1994) Function Point Tool (proprietary to Koch Productivity Consulting)
- ◇ Low G.C., Jeffery, DR. (1990) *Function Points in the Estimation and Evaluation of the Software Process*, IEEE Transactions on Software Engineering, SE-16, 1, 64-71.
- ◇ Mazzucco (1994) Function Point Tool (proprietary to Texas Instruments and IEF)
- ◇ McMenamin S.J. and Palmer J.F. (1981) *Essential System Design*, Yourdon Press, New York
- ◇ Meyerhoff D and Mullerburg M. (1992) *Set Structured Hypertext Applied to Software Measurement*, IFIP Congress 1992, Madrid.
- ◇ QIP (1994) Function Point Tool (advertized in IFPUG news as automatic)
- ◇ Rudolph E.E. (1989) *Precision of Function Point Counts*, IFPUG Spring Conference, San Diego, CA.
- ◇ St-Pierre D (1993) *Hypertexte structure applique aux points de fonction*, Centre d'interets sur les metriques, Montreal.
- ◇ SST (1994) Function Point Tool (advertized in IFPUG news as automatic)
- ◇ Wessels (1994) Function Point Tool (advertized in IFPUG news as automatic)