A SOFTWARE SIZE ESTIMATION TOOL:

HELLERMAN'S COMPLEXITY MEASURE

TOBY LERMER

(B.A., University of Lethbridge, 1986)
(B.MGT., University of Lethbridge, 1987)

A Thesis
Submitted to the Council on Graduate Studies
of the University of Lethbridge
in Partial Fulfilment of the
Requirements for the Degree

MASTER OF SCIENCE

LETHBRIDGE, ALBERTA
April, 1995

Table of Contents

List of Tables

List of Figures

A Software Size Estimation Tool:
Hellerman's Complexity Metric

1. Introduction

I present in this thesis an analysis in support of using Hellerman's Complexity Metric (HCM) [Hellerman, 1972] as a software size estimation tool. In an information system environment, Paulson and Wand previously applied HCM to identify the least complex system decomposition derivable from a formal system analysis [Paulson and Wand, 1992]. This thesis proposes that HCM could have wider applications. Its advantage over functional metrics[1] is that it is an objective measure and it can be calculated early in the system development lifecycle. Because of these two important qualities, HCM could be useful as a tool for estimating the software development size and forecasting development effort.[2]

Effort estimation is an input into project cost analysis and resource planning. It is standard practice to predict software development effort hours from a prior estimate of software project size. The software project size is measured in lines of code or function points. Clearly an inaccurate, late or nonexistent size estimate can have serious consequences. It may lead to underestimating a project's costs and to poor resource scheduling.

---

[1] For example, the original Function Point Analysis [Albrecht and Gaffney, 1983] and one variation, SPR Function Point Method [Jones, 1991].

[2]This thesis addresses only issues related to development costs and not maintenance costs.

HCM can be a useful software size estimation tool by performing two functions as follows:

1) A metric complementary to the most well employed size estimation model, Function Point Analysis (FPA) and,

2) An independent and raw size estimate of the essential system at the system analysis stage.

An accurate and timely metric of program size is a useful tool for software development firms and in-house development teams. Under or over estimating a project's size can lead to poor resource allocations, invalid cost-benefit analyses and large budget overruns :

> Accurate measures of the complexity-adjusted (FPA) size of the deliverables of a software project early in the lifecycle will permit the estimation of the relationships between the deliverables and the cost and time required to produce them. However, any error in the measurement of the deliverables will add to the errors involved in estimating the required resources. Therefore, a critical first step in software management is the use of a reliable software size measures [Kemerer, 1993, p.87].

For example, software development projects often run 100 to 200 percent over budget [Kemerer, 1993, p. 87]. Apart from the financial repercussions of unanticipated budget overruns, poor resource allocation and scheduling can hurt the relationship between developers and their clients:

> This phenomenon (differences between desired schedules and actual

> schedules) constitutes the most visible source of dissatisfaction between software developers and their clients. Clients almost always wish to have projects finished earlier than development is capable of doing [Jones, 1991, p. 152].

Jones concluded that the major reason for schedule delays was "irrational schedule targets":

> For more than half of all projects, the desired schedule targets were established by essentially irrational means. That is, the schedules were set by decree without regard to the capabilities of the staff or the complexities of the projects [Jones, 1991, p. 152].

Figure 1 (Planned vs. actual schedule duration from initiation to delivery) is from a report based on "partial[3]" historical data derived from 4000 information system, military, and system software projects developed between 1950 and 1990 [Jones, 1991, p. 124]. Almost all the projects were over schedule and the over scheduled time increased with the size of the project.

Clearly, the management of an information system development project would be enhanced by the availability of an accurate size estimation model. This project presents a new size estimation tool, HCM.

---

[3] "The word 'partial' is of great significance in this context: although thousands of projects have been examined, it is not the case that each project had a consistent, accurate, and fully detailed set of measurements associated with it. [Jones, 1991, p. 124]"

In the following chapter, I provide background information about HCM. In the third chapter, I present the reasons for using HCM as a complementary metric to FPA. In the fourth chapter, I present a theoretical argument for HCM as a potential estimator of essential

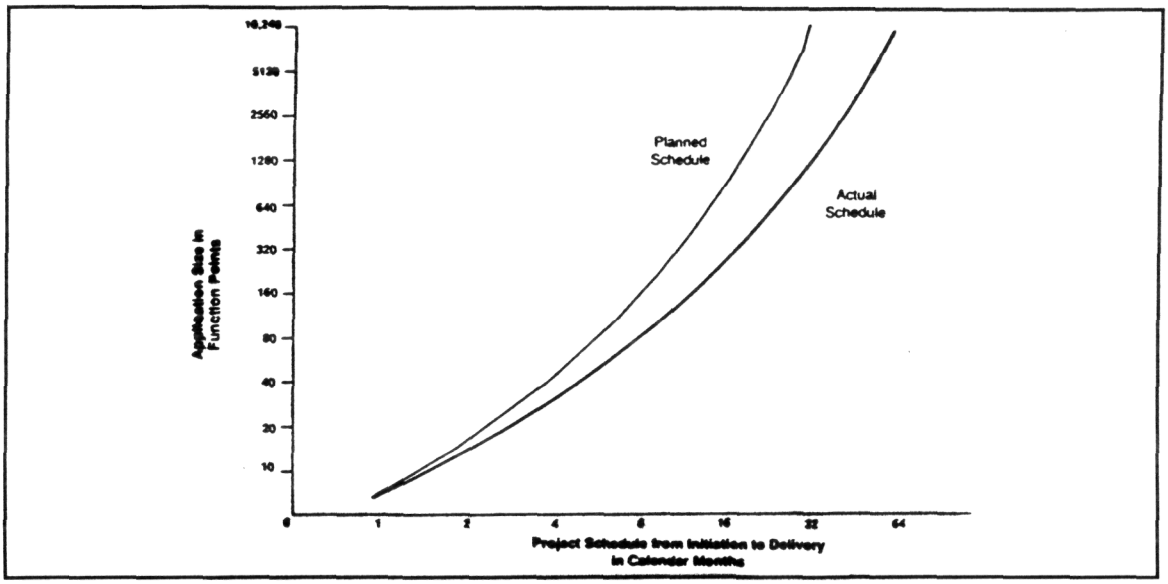

Figure 1 Planned vs. actual schedule duration from initiation to delivery [Jones, 1991, p. 151]

size at the system analysis stage. In the fifth chapter, I describe a research investigation into the relationship between HCM and the completed code. In the final chapter of this thesis, I collect my conclusions and I suggest that the HCM approach is worth further development.

2. Hellerman's Complexity Metric (HCM)

Existing research about software metrics largely involves measuring existing computer code. Cote et al. list the classic software metrics to be Halstead's software science metrics, McCabe's Cyclomatic Complexity, Lines of Code, and FPA [Cote et al, 1988, p. 121]. Except for FPA, the classic software metrics evaluate existing computer code after the system is developed.[4]

"Lines of Code" (LOC) methods include variations of counting the lines of code in the program such as counting only executable lines and data definitions. Halstead's program length and volume [Halstead, 1977] are calculated from counting operators such as (=, *) and counting operands such as variables like (A, B, C). McCabe's Cyclomatic Complexity [McCabe, 1989] is calculated from counting the number of decision statements in the code.

Unlike the classic software metrics, HCM provides a gauge of complexity at the system analysis stage before the code is written. An analyst calculates HCM from the skeletal structure of the proposed system rather than the completed code.

---

[4]On the other hand, Cyclomatic complexity has been extended to the analysis of program module hierarchies [McCabe and Butler, 1989].

Table 1 Computational Work (W) [Hellerman, 1972, p. 441-442]

Notation:

If X is a set of elements, we will let |X| denote the number of elements in the set. Throughout this paper log will mean the **base 2 logarithm**.

Definitions:

Let f:X->Y be a process defined on a finite number |X| of inputs. The domain X may be partitioned into n domain classes X_i each comprising all the points in the inverse image of some point in the range Y. The work of f is then

$$w(f) = \sum_{i=1}^{n} |X_i| \log (|X|/|X_i|)$$

Example: Let f: X x Y -->Z by z = x + y, where x,y, and z are logical variables, and + is the logic or function. The truth table {logic table} for f is the following:

x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

Each point of the domain consists of a pair of logic values (x,y). There are two domain classes. One class consists of those pairs values mapping into 0, {(0,0)}. The second class consists of those pair values mapping into 1, {(0,1),(1,0),(1,1)}. Thus, the four points of the domain are partitioned into a 1-part and a 3-part. This gives

w(f) = 1 log 4/1 + 3 log 4/3 =3.245

2.1 Adaptation of Hellerman's Computational Work

Paulson and Wand updated Hellerman's computational work measure to rank the complexity of alternative system decompositions. Hellerman's original purpose for developing a computational work measure was to "estimate the amount of work done by a process independent of it's implementation" [Paulson and Wand, 1992, p.183]" Therefore, the computational work measure is independent of the program language and the level of programmer skill. Hellerman's definition of computational work is in Table 1.

Table 2 Paulson Terminology of HCM [Paulson and Wand, 1992, p.183-184]

Let I be the total number of input states. If there are N output states, then the input states are partitioned into N domain classes. Denote by I_i the number of input states leading to the ith output state. According to information theory, if all input states have the same probability of occurrence, recognition of an input provide $\log(I/Ii)$ bits of information. The total information which is associated with input recognition during implementation or maintenance is

$$C = \sum_{i=1}^{N} I_i * \log(I/I_i)$$

Although Hellerman's measure was published in 1977, Paulson and Wand (Paulson & Wand) reemployed the measure in an information system environment. Hellerman's measure suited Paulson & Wand's requirement for a metric to gauge input recognition complexity. Input recognition "reflects the work which must be done in order to select the transformation that has to be applied to a given input"[Paulson and Wand, 1992, p.183]. Paulson and Wand revised the terminology of Hellerman's computational work equation to

an information system framework (See Table 2).

One or many system decompositions are automatically generated from a formal specification model, the States, Events, and Laws Modelling Approach (SELMA) [Paulson and Wand, 1992]. In order to choose which system decomposition to implement further, HCM is used to rank the total complexity for each decomposition. Paulson and Wand assert the premise that "reducing input recognition complexity will reduce the overall work associated with implementing and maintaining the system" [Paulson and Wand, 1992, p. 183]. In other words, HCM assists the analyst to identify the least complex decomposition before designing and implementing a system. For example, in a sample payroll system over 100 decompositions were generated by computer software. Each decomposition has a

Table 3 Sample Payroll System Decomposition

Lowest-Complexity Decomposition for a Payroll System
(Full description of this payroll system is in Appendix J)

2: {base pay, commission, overtime, total pay} 3.9
1: {end, hours, pay rate, base pay} 7.8
{end, employee position, employee type, sales, commission} 5.4
{end, employee position, employee type, hours, overtime}23.0

Total HCM: 23.09 (Sum of the subsystem HCMs)

First calculate the results of the first level subsystems, base pay, commission, and overtime. Then calculate the second level subsystem for the final total pay result. Note the end variable is a flag indicating end of period.

corresponding HCM. The decomposition with the lowest HCM is the most likely candidate to implement further. Table three shows a sample decomposition and how to interpret the decomposition.

In the following section, I summarize the formal specification method from which the decompositions are derived (SELMA), and how HCM is calculated for a whole system and a system decomposition.

2.2 SELMA

SELMA is an operationalized formal specification. It is based on a model of systems developed by Wand and Weber [Wand and Weber, 1990] who employ an ontological model developed by Bunge [Bunge, 1977] [Bunge, 1979].

The specification consists of four parts:

1) state variables,

2) state variable values,

3) external events, and

4) sublaws including (4a) stability conditions and (4b) corrective actions.

The following demonstrates a SELMA specification of a car insurance system for processing a premium based on a client's age and sex and a premium reduction based on a client's accident record. The specification is written in Prolog, a logic based programming language (shown in italics).

2.2.1 State variables.

Input, output and event variables required by the system are stated in the specification. The following entries describe all the state variables required for the simple insurance system.

```
/* state variables */
state_variable(driver).        Input variable, client's sex.
state_variable(age).           Input variable, client's age.
state_variable(accident).      Input variable, client had an accident
state_variable(reduction).     Output variable, reduction of premium
state_variable(premium).       Output variable, premium
state_variable(end).           Event variable, batch entry complete
```

2.2.2 State variable values.

Each state variable must be assigned a limited number of values. For example, the driver variable must have the value male or female.

```
/* state variable values */
values(driver,[male,female]).
values(age,[over,under]).          Client over or under age 25
values(accident,[yes,no)].         Client had accident in last 5 years
values(premium,[high,low,blank)].
values(reduction,[yes,no,blank)].
values(end,["0","1"]).             Flag to initiate end of data entry
```

2.2.3 External event(s).

The external event in the car insurance system is that data entry is complete. This is a "batch" approach. [5] The *'end'* flag at **'1'** indicates data entry is complete and initiates processing. The *'end'* flag at **'0'** informs the system that data entry is beginning and that output values should be set to blank.

```
/* events */
event("Begin data entry",[v(end,"0")]).
event("End of data entry",[v(end,"1")]).
```

2.2.4 Sublaws - stability conditions.

Stability conditions describe the state variable values or combinations of state variable values that are stable. The system is in a stable state when the state variable values require no corrective actions. For example, the system is unstable if the batch event is complete (end="1") and the premium amount is blank (premium="blank"). The premium amount must be corrected to high or low for the system to be stable.

---

[5] A sample and comparison of the car insurance premium system "on-line" is in Appendix A.

```
/* stable states */

static("end or beginning",[v(end,"0")]).
static("end or beginning",[v(end,"1")]).

static("male or female",[v(driver,male)]).
static("male or female",[v(driver,female)]).

static("age",[v(age,over)]).
static("age",[v(age,under)]).

static("accident",[v(accident,yes)]).
static("accident",[v(accident,no)]).

static("premium calc",[v(end,"1"),v(driver,male),v(age,under),v(premium,high)]).
static("premium calc",[v(end,"1"),v(driver,female),v(age,under),v(premium,low)]).
static("premium calc",[v(end,"1"),v(age,over),v(premium,low)]).
static("premium calc",[v(end,"0"),v(premium,blank)]).

static("good-driver adjustment",[v(end,"1"),v(accident,yes),v(adjustment,no)]).
static("good-driver adjustment",[v(end,"1"),v(accident,no),v(adjustment,yes)]).
static("good-driver adjustment",[v(end,"0"),v(adjustment,blank)]).
```

2.2.5 Sublaws - corrective actions.

Corrective actions "specify actions to be taken if the system is not in a stable state space, and are used to find all response paths of the system "[Paulson, 1989, p. 31]. For example, at the initial stage of data entry (end=0) in the insurance case, adjustment and premium variables should be adjusted to blank values. Another example at the completed batch entry stage (end=1) is a premium must be calculated. If the driver is male and under aged, the premium would be adjusted to the "high" value.

```
/* corrective actions */
/* beginning of data entry */
dynamic("begin",[v(end,"0")],[v(adjustment,blank),v(premium,blank)]).

/* end of data entry */
/* calculate premium */
dynamic("premium",[v(end,"1"),v(driver,male),v(age,under)],
                  [v(premium,high)]).
```

```
dynamic("premium",[v(end,"1"),v(driver,female),v(age,under)],
                  [v(premium,low)]).
dynamic("premium",[v(end,"1"),v(age,over)],
                  [v(premium,low)]).
dynamic("premium",[v(end,"0")],
                  [v(premium,blank)]).

/* end of data entry */
/* calculate good driver adjustment */
dynamic("good driver adjustment",[v(end,"1"),v(accident,no)],
                                 [v(adjustment,yes)]).

dynamic("good driver adjustment",[v(end,"1"),v(accident,yes)],
                                 [v(adjustment,no)]).

dynamic("good driver adjustment",[v(end,"0")],[v(adjustment,blank)]).
```

From a Prolog specification of state variables, state variable values, external events and sublaws, a software program designed by Paulson is able to check for consistency[6] and completeness[7]. Once the specification is complete and consistent, the software program can decompose the system into sets of subsystems . For example, the program generated a decomposition for the car insurance system which contains two

---

[6] Consistency Definition [Wing, 1990,p.11]
In terms of programs, consistency is important because it means there is some implementation that will satisfy the specification. If you view a specification as a set of facts, consistency implies that you cannot derive anything contradictory from the specification. ... An inconsistent specification which negates on one occasion what it asserts on another, means you have no knowledge at all.

[7] A SELMA specification is internally incomplete if "an external event result(s) in an unstable state that cannot be transformed to a stable state [Paulson and Wand, 1992b]." In other words, there is missing information in the specification of the system.

systems. (See Table 4) The car insurance system is a small application and example of SELMA. For a more in depth description of SELMA and the decomposition process see [Paulson and Wand, 1992].

Table 4 Car Insurance Premium System: SELMA decomposition output

Adjustment=f(accident, end)
Premium=f(age,driver,end)

2.3 Calculation of Hellerman's Complexity Metric

HCM is calculated from the stable state space generated from the formal specification. The stable state space is analogous to Hellerman's "truth table[8]". The stable state space represents all the variations of inputs and corresponding outputs which are stable in the system. It is a table of inputs and outputs. Table 5 shows the stable state space generated for the car insurance system.

[8]See Table One for sample of Hellerman's truth table.

Table 5 Stable State Space for Car Insurance System

	Inputs.....				**Outputs.....**	
	accident	age	driver	end	premium	reduction
	---	---	---	---	---	---
1.	yes	over	male	0	blank	blank
2.	no	over	male	0	blank	blank
3.	yes	under	male	0	blank	blank
4.	no	under	male	0	blank	blank
5.	yes	over	female	0	blank	blank
6.	no	over	female	0	blank	blank
7.	yes	under	female	0	blank	blank
8.	no	under	female	0	blank	blank
9.	yes	over	male	1	low	no
10.	no	over	male	1	low	yes
11.	yes	under	male	1	high	no
12.	no	under	male	1	high	yes
13.	yes	over	female	1	low	no
14.	no	over	female	1	low	yes
15.	yes	under	female	1	low	no
16.	no	under	female	1	low	yes

Table 6 shows the manual calculation of HCM from the stable state space. First, the domain classes are identified. A domain class is a unique output. Second, the frequency of each domain class is counted. From the resulting numbers, HCM is calculated.

Table 6 Manual calculation of HCM for Car Insurance System

Domain Classes		frequency
1.	low premium, no adjust	3
2.	low premium, yes adjust	3
3.	high premium, no adjust	1
4.	high premium, yes adjust	1
5.	blank, blank	8

# of Domain Classes=5
# of Transition States=16

$C = 3 * \log(16/3) + 3 * \log(16/3) + 1 * \log(16/1) + 1 * \log(16/1) + 8 * \log(16/8) = 21$

The HCM for the whole system is 21. The HCM for the decomposition is calculated by adding the HCMs of each subsystem (6 and 11). The manual calculation of HCM for each decomposed subsystem is in Table 7. Summing subsystem HCMs for each decomposition is based on the following premise: "The complexity of a decomposition of a system is the sum of complexities of the subsystems in its decomposition" [Paulson and Wand, 1992, p. 183].

Table 7 Manual calculation of HCM for Car Insurance subsystems

Functional Form of subsystem : Adjustment=*f*(end,accident)

Stable State Space

	accident	end	==>	**adjustment**
	---	---		**---**
1.	yes	1		yes
2.	no	1		no
3.	yes	0		blank
4.	yes	0		blank

Domain Classes		frequency
1.	yes	1
2.	no	1
3.	blank	2

# of Domain Classes=3 # of Input states=4

C=1 * log(4/1) + 1 * log(4/1) + 2 * log(4/2) = 6.0

Functional Form of subsystem : Premium=*f*(age,driver,end)

Stable State Space

	age	driver	end	==> **premium**
	---	---	---	**---**
1.	over	male	1	low
2.	under	male	1	high
3.	over	female	1	low
4.	under	female	1	low
5.	over	male	0	blank
6.	under	male	0	blank
7.	over	female	0	blank
8.	under	female	0	blank

Domain Classes		frequency
1.	low	3
2.	high	1
3.	blank	4

# of Domain Classes=3 # of Input States=8

C = 3 * log(8/3) + 1 * log(8/1) + 4 * log(8/4)=11.2

2.4 Qualities of Hellerman's Complexity Metric

As shown in Tables 6 & 7, HCM is a mathematical calculation derived from a stable state space generated from a SELMA specification. Therefore, it is objective. Another feature of HCM is that it is calculated early in the system development lifecycle. Specifically, a SELMA specification and the derived decompositions are created during the system analysis stage, the first stage of a traditional system development lifecycle.

3. HCM and Function Point Analysis

Presently, Albrecht's FPA is the most popular software size estimation model among practitioners and academics [Kemerer, 1993, p. 87]. With FPA, the complexity of the system is incorporated into Function Points (FPs) and Function Counts (FCs) by subjective means and by arbitrary scales. Although a proponent of the Function Point method, Jones, author of Applied Software Management, admits that complexity in all functional metrics are inadequately treated:

> It has been pointed out many times that the possible Achilles heel of functional metrics in general and function points in particular is the way complexity is treated. ... In the 1984 revision, the range of adjustments was extended and the rigor of complexity analysis was improved, but much subjectivity remains. This assertion is also true of the other flavours of functional metrics, such as the SPR function... [Jones, 1991, p. 105].

Because of the theoretical weaknesses in the calculation of complexity, I propose in this thesis that HCM be used by analysts as a complementary tool with functional metrics. In this chapter, I describe two functional metrics, the original FPA by Albrecht and a variation on FPA, the Jones SPR function. For both techniques, I summarize the problems in calculating complexity. As well, I recommend using HCM as a tool to overcome the weaknesses of functional metrics.

3.1 Function Point Analysis (FPA)

Albrecht was the first to develop a software size estimation model based on system design components. Albrecht's design variables are external inputs, external outputs, logical internal files, external interface files, and external inquiries. (See Appendix D for descriptions of these concepts) The output of Albrecht's model is a number called FP. The following pages include an example of a function point calculation worksheet and the steps to calculate FP.

Table 8 Function-point Worksheet [Albrecht and Gaffney, 1983, p. 647]

- Function Count:

type ID	Description	complexity			Total
		simple	average	complex	
IT	External Input	__ * 3 = __	__ * 4 = __	__ * 6 = __	___
QT	External Output	__ * 4 = __	__ * 5 = __	__ * 7 = __	___
FT	Logical Internal File	__ * 7 = __	__ * 10 = __	__ * 15 = __	___
EI	Ext Interface File	__ * 5 = __	__ * 7 = __	__ * 10 = __	___
QT	External Inquiry	__ * 3 = __	__ * 4 = __	__ * 4 = __	_____
FC	Total Unadjusted Function Points				_____

- Processing Complexity:

ID	Characteristic	DI	ID	Characteristic	DI
C1	Data Communications	__	C5	Online Update	__
C2	Distributed Functions	__	C9	Complex Processing	__
C3	Performance	__	C10	Reuseability	__
C4	Heavily Used Configuration	__	C11	Installation Ease	__
C5	Transaction Rate	__	C12	Operational Ease	__
C6	Online Data Entry	__	C13	Multiple Sites	__
C7	End User Efficiency	__	C14	Facilitate Change	__
PC	Total Degree of Influence				___

- DI Values:

- Not present, or no influence = 0
- Insignificant influence = 1
- Moderate inlfuence = 2
- Average influence = 3
- Significant influence = 4
- Strong influence, throughout = 5

PCA Processing Complexity Adjustment = 0.65 + (0.01 * PC) = _____

FP Function Points Measure = FC * PCA = _____

The steps to calculate FPs from Table 8 are as follows:

Step One: Calculate the FUNCTION COUNT (FC)

Count and classify functions based on complexity. The sum of the unadjusted FPs is the function count (FC).

Step Two: Calculate the PROCESSING COMPLEXITY ADJUSTMENT FORMULA (PCA)

Estimate the degree of influence (DI) of fourteen processing complexity characteristics. The total degree of influence (PC, processing complexity) is an input for a Processing Complexity Adjustment formula {PCA = .65 + (.01 * PC)}.

Step Three: Calculate FUNCTION POINTS

Calculate a Function Points measure (FP) by multiplying FC by the PCA.

As shown in the FPA worksheet above, functions are classified into low, medium, and high complexity. In order to reduce user subjectivity, factors for determining complexity and guidelines for interpretation are published {ie. [Albrecht, 1984]}. For example, Table 9 assists the analyst to determine the complexity of external inputs. In the calculation of the PCA (Step 2), a user follows and interprets rules to determine the degree of influence of fourteen "complexity characteristics" as not present (0), a strong influence (5) or in between (1-4). For example, the interpretation rule for C1, data communications is in Table 10.

Table 9 Matrix to classify the complexity of External Inputs [Jones, 1991, p. 62]

File Types Reference	Data Elements 1-4	Data Elements 5-15	Data Elements =>16
0-1	Low	Low	Average
2	Low	Average	High
=>3	Average	High	High

Table 10 Guideline Scoring for the Data Communication Factor [Jones, 1991, p. 65]

0	Batch applications
1	Remote printing or data entry
2	Remote printing and data entry
3	A teleprocessing front end to the application
4	Applications with significant teleprocessing
5	Applications that are dominantly teleprocessing

In a 1984 paper [Albrecht, 1984] Albrecht presents the basis of the "IBM function point methodology [Jones, 1991, p. 60]." After 1984, others published variations on the original FPA . To promote consistent practices a group called the International Function Point Users Group (IFPUG) was established in 1986 [Jones, 1991, p. 99]. Within the IFPUG, a counting practices committee published the first counting practices manual in April 1990 [Garmus, 1990]. According to Jones, the 1990 manual "generally follow the IBM 1984 standards, although a number of variations and extensions have occurred" [Jones, 1991, p. 99].

3.1.1 Function Points and LOC

Linear regression is frequently used to translate FPs into a LOC estimate. Albrecht and Gaffney (A&G) used a simplified version of their model for empirical testing. Their simplification involved using applied average weights instead of the low, medium and high complexity weights shown in their worksheet (See Table 8). They also ignored the processing complexity adjustment, "PCA", factor. A&G generated estimating formulas from regression analysis data of twenty-four IBM applications. The resulting formulas were tested on seventeen other IBM applications. For example, the estimation results of one equation, $SLOC_{estimated} = 12773 + 53.2\ FC$, was tested on seventeen other applications. The results showed a correlation between estimated and actual SLOC of greater than ninety percent (see Table 11). Kemerer was the first to test the FP method outside of IBM. Table 11 presents results from Kemerer's study. He used fifteen applications developed at a national consulting and services firm specializing in design and development of business data processing software [Kemerer, 1987, p.419]. Kemerer found a better correlation of LOC to function counts rather than LOC to FPs. Function counts are calculated before Function points in Step One shown in Table 8.

Table 11 Function-point model

Albrecht & Gaffney: Example - SLOC estimating formula based on twenty-four cases - combination of PL/1 and Cobol applications
[Albrecht and Gaffney, 1983]
$SLOC_{estimate}$=12773 + 53.2FC (based on all 24 cases)
Sample correlation between $SLOC_{estimate}$ and $SLOC_{actual}$: .9367
Relative error Standard deviation: .5174
Relative error average: .2406
Albrecht & Gaffney: Results of above formula tested on seventeen applications
[Albrecht and Gaffney, 1983]
Sample correlation between $SLOC_{estimate}$ and $SLOC_{actual}$: .938
Relative error Standard deviation: .480
Relative error average: .186
Prediction results of estimated SLOC from FP and FC [Kemerer, 1987]
ABC Cobol KSLOC= -5 + .20 Function points (FP) R^2=65.6%
ABC Cobol KSLOC=-13.2 + .207 Function count (FC) R^2=75.1%
KSLOC = thousands of source lines of code

In 1983, Albrecht & Gaffney concluded:

> The observations suggest a two-step estimate validation process, which uses "function pionts" or "I/O count" (unweighed function points) to estimate, early in the development lifecycle, the "SLOC" to be produced. The work-effort would then be estimated from the estimated "SLOC" [Albrecht & Gaffney, 1983, p.644].

Albrecht & Gaffney, although they calibrated their model to IBM data, did not explicitly recommend calibrating FPs to a site's historical data: "... the excellent degree of fit obtained would tend to support the view that these (and the other) formulas not validated here have some degree of universality"[Albrecht & Gaffney, 1983, p.643].

On the other hand, Albrecht & Gaffney acknowledged that different models should be used

Table 12 SLOC estimation based on Function points [Albrecht and Gaffney, 1983, p.642]

$SLOC_{Cobol\ estimate}$= 118.7 (FP) - 6,490 $SLOC_{PL/1\ estimate}$= 73.1 (FP) - 4,600

for different programming languages: "Significantly more Cobol 'SLOC' are required to deliver the same amount of 'function points' than are required with PL/1 'SLOC'"[Albrecht and Gaffney, 1983, p.642].

Albrecht & Gaffney believed the formulas to have "universal" application, a proposition later rebuked by researchers. Research has shown that it is necessary to calibrate the FP and LOC relationship to site historical data. For example, Jeffrey and Low collected data from 112 projects developed in six large Australian MIS departments [Jeffery and Low, 1990, p.220]. They concluded that SLOC/fp ratios are significantly different between organizations: "The variance using either size measure investigated here is such that organizational calibration appears **mandatory**"[Jeffrey and Low, 1990, p. 221]. Jones has published a table of LOC per FP for several computer languages (See Appendix E). For example, the database language default is 40 LOC per FP [Jones, 1991, p. 76]. Jones warns about the ratios' likely inaccuracy:

> Because of the individual programming styles and variations in the dialects of many languages, the relationship between function points and source code size often fluctuates widely, sometimes for reasons that are not currently understood [Jones, 1991, p. 75].

Because generic ratios may exhibit wide fluctuations, collection of site project data is recommended for calibration of the FP-LOC relationship for the purposes of estimating LOC from FPs.

3.1.2 Function Point Analysis and Complexity.

In the Function Point literature, the form of the complexity captured by the model is not specified. Complexity is simply incorporated into the FPA model by classifying each function in Step One (see Table 8) into low, medium, or high complex categories. There is no underlying theory explaining why one should divide the functions into three categories and why each category should have a different weight. Moreover, in Step Two, fourteen

"influential complexity factors" are evaluated on a scale of 1 to 5. Again, there is no underlying theory to explain why one should scale the influence of the particular fourteen complexity factors. Symons has written about the weaknesses of FPA [Symons, 1988]. He criticised Step One, calculation of Function Counts, for using the arbitrary weights:

> The choice of "weights" {ie. points per component type} has been justified by Albrecht as reflecting "the relative value of the function to the user/customer" and "was determined by debate and trial." It seems a reasonable question to ask if the weights obtained by Albrecht from his users in IBM will be valid in all circumstances [Symons, 1988, p.3].

In addition, Symons criticized Step Two, the method of calculating the Processing Complexity Factor, for the selection of factors and the weights for each factor:

> The restriction of 14 factors seems unlikely to be satisfactory for all time. Other factors may be suggested now, and others will surely arise in the future. A more open-ended approach seems desirable. ... The weights ("degree of influence") of each of the 14 factors are restricted to the 0-5 range, which is simple, but unlikely to be always valid. ... A re-examination of the TCF weights is therefore also desirable [Symons, 1988, p.4].

In both steps, although there are rules and guidelines, too much space remains for subjectivity. Jones recommends training analysts in function points to control the subjectivity factor:

> Counting function points by using the current IBM (1984 Albrecht FPA) method ...requires trained function point specialists to ensure consistency of the counts. Both IBM and consulting companies ... are now providing both function point training and assistance in getting started with function points. [Jones, 1991, p. 69].

Jones speculates that it is possible to automate the counts of the functional components but the complexity adjustments "may still require some form of human intervention" [Jones, 1991, p. 104].

3.2 Software Production Research Functional Metric

In 1985, in order to simplify the treatment of complexity and make it easier to calculate FPs , Software Productivity Research (SPR) introduced a variation on FPA. The SPR variation as shown in Table 13 does not classify each component into complexity levels. There is instead only one set of weights and the calculation of the Complexity Adjustment uses two scales, one for problem complexity and another for data complexity. If one is retrofitting FPs to existing software, one adds a third complexity scale to code complexity (See Appendix C). Problem or algorithmic complexity is defined as:

> This form of complexity is one of the classic topics of software engineering. The basic concept is the length and structure of algorithms intended to solve various computable problems. Some algorithms are quite simple, such as one that finds the circumference of a circle, C= pi * diameter. Other problems, such as those involving random or nonlinear phenomena, may require extremely long algorithms. Problems with high complexity tend to be perceived as difficult by the humans engaged in trying to solve them. Examples of problems with high algorithmic complexity include radar tracking and target acquisitions [Jones, 1991, p. 238].

Data complexity is defined as:

> This form of complexity, similar in concept to informational complexity, deals with the number of attributes that a single entity might have. For example, some of the attributes that might be used to describe a human being include sex, weight, height, date of birth, occupation, and marital status [Jones, 1991, p. 238].

Table 13 SPR Function Point Method [Jones, 1991]

SPR Function Point Method

Significant parameter	Empirical Weight		
Number of inputs?	_____	* 4 =	_____
Number of outputs?	_____	* 5 =	_____
Number of inquiries?	_____	* 4 =	_____
Number of data files?	_____	* 10 =	_____
Number of interfaces?	_____	* 7 =	_____
Unadjusted Total			_____
Complexity Adjustment			_____
Adjusted function point total			_____

Calculation of SPR Complexity Adjustment Factor

Problem complexity? _____

1. Simple algorithms and simple calculations
2. Majority of simple algorithms and calculations.
3. Algorithms and calculations of average complexity.
4. Some difficult or complex calculations.
5. Many difficult algorithms and complex calculations.

Data complexity? _____

1. Simple data with few variables and low complexity
2. Numerous variables, but simple data relationship.
3. Multiple files, fields, and data interactions.
4. Complex file structures and data interactions.
5. Very complex file structures and data interactions.

Sum of Problem complexity and Data complexity _____

The SPR Complexity Adjustment Factors

Complexity sum	Adjustment multiplier
1	0.5
2	0.6
3	0.7
4	0.8
5	0.9
6	1.0
7	1.1
8	1.2
9	1.3
10	1.4
11	1.5

Although a proponent of the Function Point method, Jones admits that functional metrics treat complexity inadequately.

3.3 HCM as a Complementary Tool

Function Point Analysis is a popular method for estimating the size of a system but the major flaw, it's "Achilles heel", is the arbitrary and subjective manner complexity is handled by the model. In contrast, HCM is an automated, mathematical calculation based on Hellerman's theory of computational work. Not only does HCM facilitate the decomposition of the system into elementary subsystems, it quantifies the input recognition complexity of the decomposed elementary subsystems.

In practical terms, the system developer or manager would be able to take into consideration both functional metric results as well as the HCM of the selected decomposition. For example, imagine that a fictitious firm, XYZ, maintains data on function counts, function points, predictive LOC, actual LOC, and development effort (time) for it's information systems written in one language. From this data, XYZ currently calibrates the FP:LOC and LOC:Effort linear relationships. XYZ decides to maintain data on the HCM of the developed decompositions. With sufficient historic data on each system's HCM, completed LOC, and work hours, XYZ could calibrate the HCM:LOC and HCM:Effort relationships. With this complementary data, an organization may predict more accurately the development effort and/or system size.

4. HCM as an Independent Metric

During the past decade, researchers have addressed software size estimation {for listing see [Cote et al, 1988, p. 122]}. Although many authors propose one or another model, few authors provide an underlying theoretical foundation for their choice. By contrast, Wrigley and Dexter [Wrigley and Dexter, 1991] relate their size estimation models' applications to a "theoretical research model for measuring information size" (Research Model). This next chapter describes the Research Model and then compares HCM to Wrigley and Dexter's Research Model.

4.1 Research Model for Measuring Information System Size

In Wrigley and Dexter's Research Model (illustrated in Figure 2), Wrigley and Dexter first show a conceptual relationship between a real system and it's software size, and then show an empirical relationship between a real system and it's eventual Source Lines of Code (the LOC excluding comments). Wrigley and Dexter write the following about the Research Model:

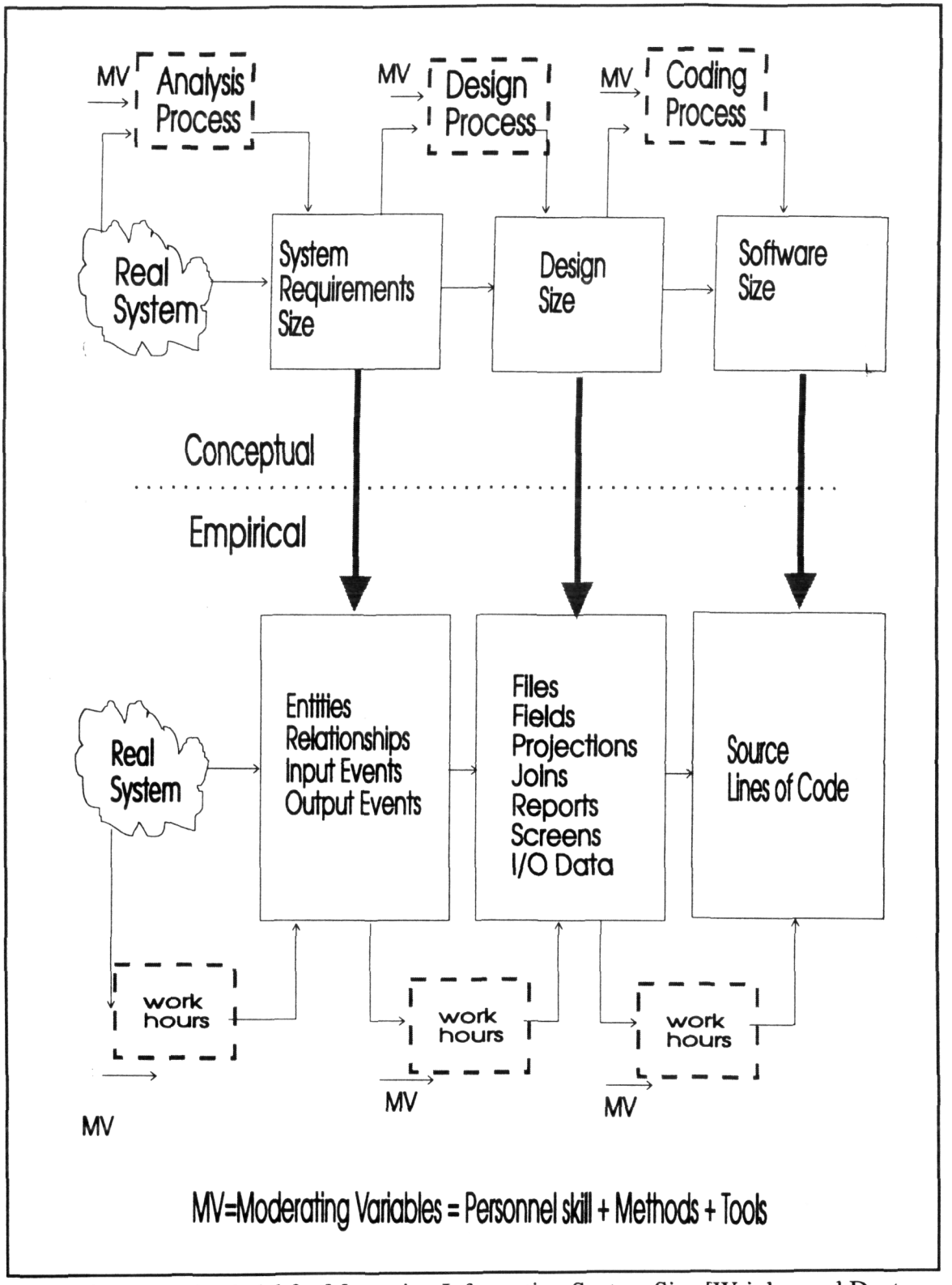

Figure 2 Research Model for Measuring Information System Size [Wrigley and Dexter, 1991, p. 247]

> Each stage of development is achieved through the various processes: analysis, design, and coding. System specifications at each stage are transformed into the next stage through these processes, which are moderated by two factors: kinds of methods and tools used and the skill level of system builders [Wrigley and Dexter, 1991, p.248].

Wrigley and Dexter''s empirical study of their Research model focuses on the relationship of the Design Size variables (ie. Files, fields, projections, joins, reports, screen & I/O data) to a system's completed LOC. The study uses data "reverse engineered" from twenty-six small business systems coded in a fourth generation language (4GL) [Wrigley and Dexter, 1991, p. 55]. The code analyzer "reverse engineers" the code back to the design metrics: "(the code analyzer) takes as input the source code of completed working systems and generates a number of software metrics." [Wrigley and Dexter, 1991, p. 251] Wrigley and Dexter's best result used regression analysis to show that the number of preliminary design variables (screens, reports, and files) explained 94 percent of the variance in code size [Wrigley and Dexter, 1991, p. 254].

4.2 HCM and the Research Model

The inputs into HCM correspond well to Wrigley and Dexter's Research Model (See Figure 2). The Research Model represents system requirement size with empirical elements consisting of Entities & Relationships, Input events, and Output events. SELMA represents both Input and Output events and the relations between them with state variables, external events, and sublaws. HCM calculated from a SELMA specification is a metric which quantifies the system dynamics.

Wrigley and Dexter believe that one limitation of their study is "that only the design to code transformation has been fully operationalized" [Wrigley and Dexter, 1991, p. 255]. Wrigley and Dexter write: "Future studies should focus on properties of information systems that are measurable during analysis and can be empirically correlated to the amount of effort and code required for development" [Wrigley and Dexter, 1991, p. 253].

In contrast, HCM can be calculated from a system analysis specification instead of a system design specification. Because it is calculated prior to system design, an estimation of information system size and required effort may be possible early in the system development life cycle (SDLC). (See Figure 3)

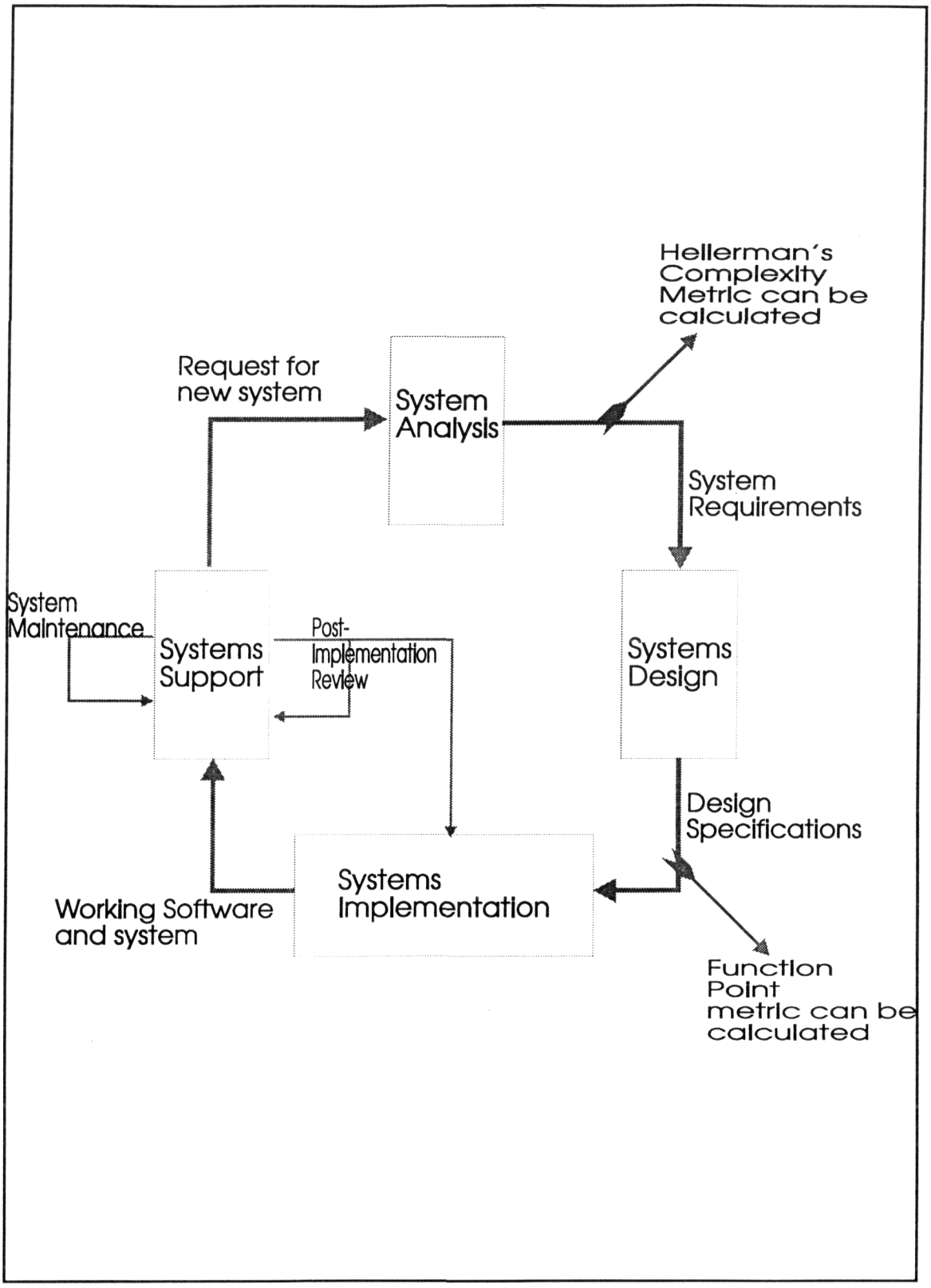

Figure 3 The Traditional Systems Development Life Cycle [Whitten et al., 1989, p. 82]

Although, HCM is theoretically a useful metric to measure the system's size, there are several weaknesses to be considered. First, there is no empirical evidence that the system requirements size is related to the design size or more radically, whether the system requirement's size is related to the software size. HCM inputs (state variables, external events, and sublaws) and the Research Models' variables (Entities, relationships, input events, output events) present two rough sketches of the proposed system. The HCM rough sketch is a representation of the "essential" system. For example, in the car insurance specification in Chapter 2, the drivers' sex was included but not the driver's last name, first name, address, and so on. The relationship between the essential system and the detailed system may be too unstable and the gap may be too large to allow use of a metric calculated from the system requirements size to predict work effort or project size. On the other hand, empirical study may show that a relationship exists between HCM, effort and software size.

As shown in Figure 3, both Function Points or Wrigley and Dexter's regression equation(s) are calculated after the System Design stage is completed. However, HCM is calculated after the System Analysis stage. Function Points and Wrigley and Dexter's regression equations(s) would likely be much more accurate than HCM in calculating the LOC because the information about the system is more complete at their stage of calculation. I suggest the analyst calculate a size estimate both before and after system design. There are advantages to estimating project size early. One advantage is it makes possible an early cost-benefit analysis about whether the project is worth pursuing to completion. If the project is not economically viable, it can be halted. It is easier psychologically to halt a project after system analysis than after system design. After system design, the time and money sunk in the project could be considerable and decision makers may be over committed to the project. Another advantage is that if the project is found to be feasible, the project budget planning and personnel scheduling can be determined earlier in the SDLC.

5. HCM Research

Although a reasonable theoretical argument supports using HCM as an estimator of complexity and/or size, no empirical evidence exists that HCM identifies the completed system's complexity and/or size. In this chapter, I report results from an investigation of three small information systems, each with a different HCM decomposition, to examine if HCM forecasts the corresponding size and complexity of the completed code.

5.1 Hypotheses

In this preliminary investigation, I study the relation between HCM computed at the systems analysis stage and the corresponding LOC of the completed code. The following null hypotheses was tested:

Ho: No significant relationship exists between HCM and actual LOC.

Also in the preliminary investigation, I look at the relationship between HCM and the McCabe complexity of the completed code. The following null hypotheses was tested:

Ho: No significant relationship exists between HCM and McCabe Complexity metric v(G).

5.2 Research Subjects

University students in a fourth year system design class and majoring in Management Information Systems, were supplied three cases and asked to program an information system for each case. I used the students' output as data for testing the two hypotheses stated in Section 5.1.

According to Wrigley and Dexter's Research Model for Measuring Information System Size, the size of a system is influenced by "moderating factors" [Wrigley and Dexter, 1991, p. 248]. Their three moderating variables are personnel skill, methods, and tools. It is easy to equalize methods and tools in the study but it is impossible for a large group of people to program at the same skill level. Wrigley identifies the following personnel skill factors: programming experience, experience with the programming language, experience with the hardware to be used, application experience, and the existence of similar systems

[Wrigley, 1988]. Benbasat and Vessey describe programmer characteristics as a combination of innate ability, programming experience, and source language experience [Benbasat and Vessey, 1980, p.32]. In this research study, I attempt to mitigate the personnel skill factors by having several students program the same systems. Therefore, I studied average variable data from all the students rather than data from one individual's system. There were twenty-two students in the System Design class. The students were assigned to design and program three systems: a car insurance system, a payroll system and a hotel registration systems (Appendix J)[9].

5.3 Research Design

Each student was provided with a basic system analysis documentation for each system including the following:

1. a short description,
2. a context diagram and level 0 data flow diagram, and
3. a data dictionary. (Documentation of data stores, data structures, processes, and data elements. Decompositions from SELMA were included in the process descriptions.)

In addition, I provided students with databases, test data, and library program files for downloading. The library files (See Appendix J for descriptions.) helped to structure the assignment and to better replicate the approach used in a typical software development office or department. A software development office would likely require a programmer to use a consistent style and promote the use of reusable files. The students wrote the programs in dBASE IV for DOS, a relational database. I wrote the SELMA specification and calculated the HCM before the students began to program. The SELMA specification, decomposition

---

[9] I adapted the car insurance system, the payroll system and the hotel system from cases in system analysis texts. The car insurance system databases and data flow diagram are largely based on the "Open Road Insurance System" in System Analysis and Design Methods [Whitten et al., 1989]. The process decision rules are based on an example in Auditing EDP Systems [Watne et al., 1990]. The payroll process decision rules are based on an example in Paulson's doctoral dissertation [Paulson, 1989]. The payroll databases are revised from the book, Accounting Information Systems [Nash and Heagy, 1993]. The hotel system is a simplified version of a "Luxury Inns" system in System Analysis and Design Methods [Whitten et al., 1989].

and corresponding HCM are reproduced in Appendix G, H and I. The students were not told the HCM of the systems.

The students designed and programmed the assigned systems. From the completed code, I calculated three classes of variables for each system as follows: line of code measures, McCabe complexity metrics, and function count. Specifically, eight variables were calculated for each system:

I. LOC

1. Total LOC,
2. Generated LOC - Lines of code generated by a report and/or screen generator.
3. Programmed LOC - Lines of code not generated but programmed.

II. McCabe Complexity Metric

4. System Complexity - Modified McCabe Complexity metric of the entire Programmed LOC,
5. Process Complexity - Modified McCabe Complexity metric of the "essential process(es)" code. The "essential process(es)" are the process(es) documented in the SELMA specification on which the HCM is calculated for each system.

III. Function Count

5. Function Count
6. Number of Screens, and the
7. Number of Reports

SNAP [Kennamer, 1991] is a software shareware program to assist in the documentation and development of X-Based Systems. I used SNAP to compile the above data for each system. Data for each system about the total lines of code, the number of reports/label forms, and the number of format files was taken from two SNAP output files, the tree diagram and the system summary report (See Table 14.) As well, SNAP regenerated all files in the system without blank lines and comments. These newly regenerated files were used for LOC counting. Finally, SNAP generated action diagrams. (See Table 16.) The action diagrams allowed me to objectively calculate a modified McCabe complexity metric.

Table 14 SNAP system summary and tree diagram for Subject Four

System: Car Program
Author: Subject Four - Lalonde
System Summary

This system has:
710 lines of code
7 program files
3 procedure files
10 procedures and functions
2 databases
2 multiple index files
0 index files
2 report forms
2 format files
0 label forms
0 binary files
0 memory variable files
0 menu files
0 screen files
0 other files
0 cross-referenced tokens

See the tree diagram for programs, procedures, functions and format files

Databases	Index Files	Report Forms	Label Forms	Memory Files
INSUREE.DBF		COVERAGE.FRG		
POLICY.DBF		ACCIDENT.FRG		

System: Car Program
Author: Subject Four - Lalonde
Tree Diagram

```
CONTROL.PRG
└──CENTRAL.PRG
   ├──COLOUR    (procedure in CENTRAL.PRG)
   └──DEFPOP.PRG
      ├──MODULE1.PRG
      │   INSUREE.DBF (database)
      │   POLICY.DBF (database)
      │  ├──INSUREE.FMT
      │  └──POLICY.FMT
      ├──MODULE2.PRG
      │   INSUREE.DBF (database)
      │   POLICY.DBF (database)
      └──MODULE5.PRG
```

5.3.1 LOC calculation.

Table 15 Line-Counting Methods [Jones, 1986, p.15]

Method 1.	Count only executable lines
Method 2.	Count executable lines plus data definitions.
Method 3.	Count executable lines, data definitions, and comments.
Method 4.	Count executable lines, data definitions, comments, and Job Control Language.
Method 5.	Count lines as physical lines on an input screen.
Method 6.	Count lines as terminated by logical delimiters.

Generated LOC and programmed LOC (non generated code) are measured separately since the required effort in design, programming, debugging and maintenance of programmed LOC is considerably more than generated code. Generated code is automatically generated from "WYSIWYG (What you see is what you get)" screen and report forms. LOC calculation can vary considerably. Jones lists several line counting methods. See Table 15. According to Jones, Method Two was used in productivity studies by IBM and is common among IBM customers [Jones, 1986, p. 16]. The detailed rules for counting LOC by Method 2 are the following:

1. Lines are terminated by delimiters.
2. Verbs or operational statements are included.
3. Data definitions are included.
4. The code delivered to the user is the basis for the count.
5. Job control language is excluded.
6. Comments are excluded.
7. Temporary code developed to aid testing is excluded.[Jones,1986, p. 90]

SNAP reports the total LOC (excluding blank lines & comments) in the system summary. As well, SNAP generated new files which excluded blank lines and comments. From these new files, I was able to document the LOC in each generated and programmed file.

5.3.2 McCabe Complexity Metric Calculation.

SNAP was also integral to calculating a modified McCabe Complexity metric for the programmed files and the "essential process(es)" code. Cyclomatic complexity "is found by determining the number of decision statements in a program and is calculated as: v(G)=number of decision statements + 1 " [McCabe and Butler, 1989, p. 1416]. A simple explanation of cyclomatic complexity is quoted from a paper by McCabe and Butler in Table 16. In this study, I calculated cyclomatic complexity by adding the decision statements identified by SNAP action diagrams which depict the paths in the program. One exception is that I did not count a DO CASE as a decision statement but I counted each following case routine. In general, I followed the policy of counting a statement if it was equivalent to one or more IF statements. This corresponds to McCabe's example where a compound statement (ex. *IF A=B and C+D then*) was counted as two decision statements [McCabe and Butler, 1989, p. 1416]. I also counted indirect decisions statements statements which incorporate the "*for* ..." phrase in the dBASE language. For example the following dBASE line:

Replace m->madjustment with m->adjustment+200 for maccident=.t.

is the same as the following three lines:

IF maccident=.t.

m->madjustment=m->adjustment + 200

ENDIF

An example of the procedure I followed to sum the decision statements is in Table 17.

Table 16 Cyclomatic Complexity [McCabe and Butler, 1989, p. 1416]

The cyclomatic complexity approach is to measure of the number of paths through a program. Cyclomatic complexity, v(G), is derived from a flowgraph and is mathematically computed using graph theory. More simply stated, it is found by determining the number of decision statements in a program and is calculated as:
v(G)=number of decision statements + 1
By counting the decision statements, called predicates, the complexity of a program can be calculated. However many decision statements contain compound conditions. An example is a compound IF statement:
IF A=B and C=D then
If the predicates are counted in this example, v(G) is equal to 2 (1 IF statement + 1). If compound conditions are counted, the statement could be interpreted as:
IF A=B and IF C=D then
Therefore, v(G) would be 3. Cyclomatic complexity recognizes that compound predicates increase program complexity and integrates individual conditions in order to calculate v(G).

Table 17 Decision Statement Counting Procedure

decision
statements Module2.prg code - Subject Four - excludes blank lines & comments

```
#          1   mod2exit="N"
1          2   DO WHILE mod2exit="N"
           3     ACTIVATE POPUP mod2
           4     DO CASE
2          5     CASE BAR() = 1
           6       SELECT A
           7       USE insuree ORDER insuree
           8       SELECT B
           9       USE policy ORDER insuree
           10      SET RELATION TO insuree INTO insuree->insuree
           11      SET FIELDS TO insuree, insuree->sex, insuree->birth, accident, adjustment, premium
           12      m->minsuree=0
           13      m->maccident=.T.
           14      m->madjustment=0
           15      m->mpremium=0
           16      m->msex=SPACE(LEN(sex))
           17      m->mbirth={  /  /  }
           18      DEFINE WINDOW quest FROM 10,10 TO 18,70
           19      ACTIVATE WINDOW quest
           20      @1,5 SAY "Please enter the Insuree # :  " GET m->minsuree PICTURE "9999"
           21      READ
           22      SEEK m->minsuree
           23      m->maccident=accident
3          24      IF maccident=.T.
           25        m->madjustment=m->madjustment + 200
           26      ENDIF
           27      m->msex=sex
           28      m->mbirth=birth
45         29      IF (m->msex="F") .OR. (m->mbirth < {01/01/70})
           30        m->mpremium=500
6          31      ELSE
           32        m->mpremium=1000
           33      ENDIF
           34      REPLACE adjustment WITH m->madjustment
           35      REPLACE premium WITH m->mpremium
           36      DEACTIVATE WINDOW quest
           37      CLOSE ALL
7          38     CASE BAR() = 2
and so on.....
```

Procedure to add decision statements: Counted all decision statements identified by the action diagram. Did not count a DO CASE as a decision but each following case routine. In general, I followed the policy of counting a statement if it was equivalent to one or more IF statements. This corresponds to McCabe's example [McCabe and Butler, 1989, p. 1416] where a compound statement was counted as two decision statements (ie. IF A=B and C=D then), an IF statement + 1. I extended summing decision statements dBASE **for ....** phrases where dBASE language incorporates indirectly the IF statement.

5.3.3 Function Count Calculation.

The Function Count for each system was calculated employing the SPR Function Point Method (Table 13). The FC calculation is the same as employing Albrecht and Gaffney's Function-point Worksheet (Table 8) with average weights. There are five inputs into the FC equation including the following:

1. the number of inputs (screens or forms),
2. the number of outputs (reports),
3. the number of inquiries,
4. the number of data files (tables within a relational database), and
5. the number of interfaces.

Definitions of these five components are in Appendix D. Only three components were applicable to the student systems:

1. the number of inputs (format files),
2. the number of outputs (report forms), and
3. the number of data files (*.dbf files).

SNAP reported these three numbers for each student system in the Summary report. The Function Count for each system was calculated from the following equation:

FC = (# of screens * 4) + (# of reports * 5) + (# of database files * 10)

5.4 Results

Thirteen students completed the programming for two systems, the car insurance system with a HCM of 17.25 and the payroll system with a HCM of 23.09. Only three students completed programming the hotel system with a HCM of 43.9.[10]

Five variables are reported in Table 18 for each of the thirteen students and for both the car insurance (low HCM) and payroll (high HCM) systems. The significance of the

[10] According to the students' written feedback, they were unable to complete all three cases because of time constraints.

HCM measure as a prior indicator of system size or complexity is tested by applying a t-test to the difference of means of the pairs of dependent variables matched for each student. Table 18 reports the mean and standard deviation of each dependent variable. The t-test is applied to different significance levels on a one-tailed test[11] in Table 19 and 20.

The mean difference of the two systems is significant for the Total LOC variable and Generated LOC variable only at a 25% significance level but for the Programmed LOC variable is significant at a 5% level. It is not surprising that the mean difference of the programmed LOC is more significant since it includes the "essential" processes code that SELMA documents and HCM captures. On the other hand, the size of Generated Code is highly influenced by the number of reports or screens the designer/programmer decides to include in the system. The Total LOC in both the payroll and car insurance systems comprises more of generated code (approximately 60%) than programmed code.

By contrast, at a 5% level of significance, the sample result is statistically significant for the difference of means for the system complexity and at a 1% level of significance, the result is statistically significant for the difference of means for the process complexity variable. This is a strong result from a single test of two systems with relatively similar HCM values. It was anticipated that the students would program three systems with a broader range of HCM values but only three students were able to complete the programming of the more complex system.

---

[11] A one-tailed test was applied since the alternative hypothesis for all variables is that the payroll mean is greater than the car insurance mean.

Ho: $\mu_A - \mu_B \leq 0$

Ha: $\mu_A - \mu_B > 0$

Based on the mean difference testing between the car insurance system (HCM-17.25) and the payroll system (HCM - 23.09), I conclude that HCM is a modestly significant determinant of the lines of code variables and a highly significant determinant of system complexity and "essential process(es)" complexity.

In order to make use of the information from the three students who programmed the hotel system, I developed forecasting equations from the car and payroll systems. The equation is found by running a line through the point of means of the two systems for each of the dependent variables. The equations are shown in Table 21 with the forecast and actual values for the hotel system. The forecasted values are surprisingly accurate. It would be dangerous to draw any conclusion from an analysis based on the results of three student programs. Though the accuracy of the forecast equation based on the first two student systems is extraordinary, it should only be used as an indication that the HCM method has promise.

HCM is calculated from a system analysis specification before the design and coding of the system. On the other hand, FC is calculated from a detailed system design specification when a lot more is known about the system. Because the system design was the responsibility of each student, the calculated FC was different for each system. The results of the regression analysis with FC as an independent are shown in Table 22. FC explains 88% of the variation in Total LOC. . The R-squared is 83% for the Generated LOC, 44% for the programmed LOC, 10% for the system complexity, and 3% for the "essential process(es)" complexity. When the linear equations were employed to forecast LOC and complexity for the hotel system, the accuracy is better for the LOC variables than the system complexity variable (See Table 21). In short, the FC variable is a reasonable measure of the

LOC but a poor measure of system complexity. In contrast, HCM appears to be an excellent forecast measure both of system complexity and lines of code.

Table **18** Student System Data: LOC and Complexity Variables

Subject ID#	#1 TOTAL LOC			#2 PROGRAMMED LOC		
	Car Total LOC	Payroll Total LOC	Difference Total LOC	Car Programmed LOC	Payroll Programmed LOC	Difference Programmed LOC
1	750	790	40	269	317	48
2	541	575	34	240	244	4
3	710	568	-142	242	253	11
4	747	751	4	223	282	59
5	622	965	343	332	332	0
6	720	546	-174	244	240	-4
7	807	1302	495	369	478	109
8	708	910	202	225	276	51
9	548	608	60	223	236	13
10	674	865	191	228	243	15
11	811	704	-107	304	250	-54
12	759	551	-208	214	225	11
13	711	795	84	315	330	15
MEAN	700.6	763.8	63.2	263.7	285.1	21.4
S.D.	82.2	207.2	197.4	48.0	65.8	37.3

Subject ID#	#3 GENERATED LOC		
	Car Generated LOC	Payroll Generated LOC	Difference Generated LOC
1	465	461	-4
2	295	326	31
3	460	308	-152
4	467	426	-41
5	283	624	341
6	468	299	-169
7	433	796	363
8	473	625	152
9	319	377	58
10	435	613	178
11	499	447	-52
12	537	320	-217
13	390	458	68
MEAN	424.9	467.7	42.8
S.D.	76.7	147.9	173.5

Subject ID#	#4 SYSTEM COMPLEXITY			#5 PROCESS COMPLEXITY		
	Car System Complexity	Payroll System Complexity	Difference System Complexity	Car Process Complexity	Payroll Process Complexity	Difference Process Complexity
1	25	37	12	5	7	2
2	28	28	0	5	5	0
3	16	30	14	4	8	4
4	27	33	6	4	10	6
5	32	29	-3	4	4	0
6	29	30	1	4	6	2
7	37	31	-6	5	7	2
8	28	37	9	3	9	6
9	28	32	4	5	9	4
10	28	32	4	3	5	2
11	28	39	11	5	9	4
12	28	18	-10	4	4	0
13	31	41	10	4	10	6
MEAN	28.1	32.1	4.0	4.2	7.2	2.9
S.D.	4.5	5.6	7.1	0.7	2.1	2.2

Table **19** Difference of Means between the Payroll and Car Systems for LOC variables

Total LOC-Payroll is at most the same mean as Total LOC-Car

Ho: ua - ub <=0	Ha: ua - ub >0	Accept Ho if t <= critical value
Average Difference	63.23	
Standard Deviation	197.40	

Significance Level	Confidence Level	Critical Value One-tailed Test 12 Degrees of Freedom	Test Statistic
0.01	0.99	2.681	1.2
0.05	0.95	1.782	1.2
0.10	0.90	1.356	1.2
0.25	0.75	0.695	1.2

Matched Pair Sample 13-1 for DF — Formula

Reject Ho at 25% level of significance.

Programmed LOC-Payroll is at most the same mean as Programmed LOC-Car

Ho: ua - ub <=0	Ha: ua - ub >0	Accept Ho if t <= critical value
Average Difference	21.38	
Standard Deviation	37.35	

Significance Level	Confidence Level	Critical Value One-tailed Test 12 Degrees of Freedom	Test Statistic
0.01	0.99	2.681	2.1
0.05	0.95	1.782	2.1
0.10	0.90	1.356	2.1
0.25	0.75	0.695	2.1

Matched Pair Sample 13-1 for DF — Formula

Reject Ho at 5% level of significance.

Generated LOC-Payroll is at most the same mean as Generated LOC-Car

Ho: ua - ub <=0	Ha: ua - ub >0	Accept Ho if t <= critical value
Average Difference	467.69	
Standard Deviation	147.92	

Significance Level	Confidence Level	Critical Value One-tailed Test 12 Degrees of Freedom	Test Statistic
0.01	0.99	2.681	0.9
0.05	0.95	1.782	0.9
0.10	0.90	1.356	0.9
0.25	0.75	0.695	0.9

Matched Pair Sample 13-1 for DF — Formula

Reject Ho at a 25% level of significance.

$$\text{Test statistic} = t = \frac{\bar{D}}{S_{\bar{D}}/\sqrt{n}}$$

Table **20** Difference of Means between the Payroll and Car systems for Complexity Variables

Process Complexity-Payroll is at most the same mean as Process Complexity LOC-Car

Ho: ua - ub <=0	Ha: ua - ub >0	Accept Ho if t <= critical value
Average Difference	2.92	
Standard Deviation	2.16	

Significance Level	Confidence Level	Critical Value One-tailed Test 12 Degrees of Freedom	Test Statistic
0.01	0.99	2.681	4.9
0.05	0.95	1.782	4.9
0.10	0.90	1.356	4.9
0.25	0.75	0.695	4.9

Matched Pair Sample
13-1 for DF

Reject Ho at a 1% level of significance.

System Complexity - Payroll is at most the same mean as System Complexity -Car

Ho: ua - ub <=0	Ha: ua - ub >0	Accept Ho if t <= critical value
Average Difference	4.00	
Standard Deviation	7.06	

Significance Level	Confidence Level	Critical Value One-tailed Test 12 Degrees of Freedom	Test Statistic
0.01	0.99	2.681	2.0
0.05	0.95	1.782	2.0
0.10	0.90	1.356	2.0
0.25	0.75	0.695	2.0

Matched Pair Sample
13-1 for DF

Reject Ho at a 5% level of significance.
(notebk2.wb1(A):Differ)

$$\text{Test statistic} = t = \frac{\bar{D}}{S_{\bar{D}}/\sqrt{n}}$$

Table **21** Forecast Results for the Hotel System

Forecast Equations - HCM Independent Variable
Dependent Variable = A + B * HCM

Constants (1)	Dependent Variables Total LOC	Generated LOC	Programmed LOC	System Complexity	Process Complexity
A	513.8	298.6	200.5	16.3	-4.4
B	10.8	7.3	3.7	0.7	0.5
Forecast with HCM=43.9	**987.92**	**619.07**	**362.93**	**47.03**	**17.55**
Actual					
Student 1	1128.0	265.0	863.0	35.0	
Student 2	1172.0	408.0	764.0	54.0	**14**
Student 3	696.0	342.0	354.0	48.0	**17**
MEAN	**998.7**	**338.3**	**660.3**	**45.7**	**15.5**
Error	10.7	-280.7	297.4	-1.4	**-2.0**

Forecast Equations - FC Independent Variable
Dependent Variable = A + B * FC

Constants (2)	Dependent Variables Total LOC	Generated LOC	Programmed LOC	System Complexity	Process Complexity
A	-701.28	-589.54	-96.11	13.82	2.48
B	37.53	27.12	9.7	0.43	0.08
FC Forecast					
Student 1 - 47	1062.6	685.1	359.8	34.0	6.2
Student 2 - 46	1025.1	658.0	350.1	33.6	6.2
Student 3 - 33	537.2	305.4	224.0	28.0	5.1
Actual					
Student 1	1128.0	265.0	863.0	35.0	
Student 2	1172.0	408.0	764.0	54.0	
Student 3	696.0	342.0	354.0	48.0	
Error					
Student 1	65.37	-420.1	503.21	0.97	
Student 2	146.9	-249.98	413.91	20.4	
Student 3	158.79	36.58	130.01	19.99	
Calculation of Function Count					
Student	Screens	Reports	dBASE tables		FC
Student 1	4.0	3.0	4.0		47.0
Student 2	5.0	2.0	4.0		46.0
Student 3	3.0	1.0	4.0		33.0

(test3.wb1:forecast)

Footnote (1) :
Constants A & B are derived from running a line through the two point of means of the Dependent variable. For example, for Total LOC there are two lines:
of LOC = a + b * HCM.
Car Insurance System: 700.6 = a + b * 17.2
Payroll System: 763.8 = a + b * 23.1
Solve for A and B: A= 513.8 and B=10.7.

Footnote (2):
Constants A & B are derived from regression analysis. See Table 22.

Table **22** Regression Analysis with Independent Variable, Function Count (FC)

Dependent Variable = a + b * FC Analysis based on data from Appendix C.

Total LOC = a + b * FC Regression Output:	
Constant	-701.28
Std Err of Y Est	56.76
R Squared	**0.88**
No. of Observations	26.00
Degrees of Freedom	24.00
X Coefficient(s)	37.53
Std Err of Coef.	2.76

Process(es) Complexity = a + b * FC Regression Output:	
Constant	2.48
Std Err of Y Est	2.20
R Squared	**0.03**
No. of Observations	26.00
Degrees of Freedom	24.00
X Coefficient(s)	0.08
Std Err of Coef.	0.11

Programmed LOC = a + b * FC Regression Output:	
Constant	-96.11
Std Err of Y Est	45.45
R Squared	**0.44**
No. of Observations	26.00
Degrees of Freedom	24.00
X Coefficient(s)	9.70
Std Err of Coef.	2.21

Generated LOC = a + b * FC Regression Output:	
Constant	-589.54
Std Err of Y Est	50.97
R Squared	**0.83**
No. of Observations	26.00
Degrees of Freedom	24.00
X Coefficient(s)	27.12
Std Err of Coef.	2.48

(testdat2.sb1(c):anareg)

6. Conclusion

HCM quantifies the input recognition complexity of a system from a formal system analysis specification. Paulson and Wand employed Hellerman's Computational Work metric to rank the input complexities of different system decompositions in an information system context. My research reported in this thesis suggests that HCM might have broader applications. First, HCM could be a complementary metric to Function Points, a software size estimation model. Second, HCM could be an independent metric for an early raw size/effort estimation. In this thesis, I show that there is a theoretical foundation for these wider applications. As well, the empirical investigation of the relationship between HCM and completed code size and complexity, supports the theoretical argument.

Function Points has been criticized for it's treatment of complexity. Kemerer, a researcher who tested the FP method, found a better correlation of LOC to function counts which excludes the processing adjustment formula (PCA) rather than LOC to function points [Kemerer, 1987, p. 419]. In order to improve FP's weak treatment of complexity, Jones SPR variation on FPA, excludes the PCA and employs only input average weights. Alternatively, HCM objectively quantifies input recognition complexity and therefore, HCM could compensate for FPA's ineffectual treatment of complexity. The regression analysis on the student system's completed code indicates a high correlation of FC (R^2=88%) to total system LOC, and a low correlation of FC to system complexity (R^2=10%).

FPA is an often used model for size/effort estimation. On the other hand, Function points or function counts can only be estimated after a detailed system design specification. In the system development lifecycle, a system design specification typically may not be completed until after half way through the total effort (man hours) toward a working system. Although budgets and planning can be modified after a FP:LOC or FP:Effort estimation, the original budget, scheduling and cost-benefit decisions are made typically at the beginning of the system life cycle. This thesis suggests that HCM could be used effectively as an early and independent estimate of a raw size/effort. HCM can be calculated after a specification of the essential processes of the proposed system are documented. Wrigley and Dexter developed a model which theoretically indicates that properties (Entities & relationships, Input events, and Output events) at the system analysis stage have a link to the final completed code size and amount of effort variables. The results from an investigation of student systems indicates modest empirical support for Wrigley and Dexter's theory. The difference of means of the programmed LOC (α=5%), the difference of means of the system complexity (α=5%), and difference of means of the process complexity (α=1%) between the payroll system (high HCM) and the car system (low HCM) are significant. When the results of the car and payroll system were used to forecast the three hotel systems, the prediction numbers of both LOC and complexity levels were surprisingly accurate.

This thesis suggests the use of HCM as a new software effort and size estimation tool. HCM has several outstanding qualities. HCM is an objective, mathematical calculation that can be computer generated from a Prolog SELMA specification early in the SDLC. The preliminary investigation with student data suggests that there is a relationship of HCM to

final LOC and complexity. Future studies should focus on gathering empirical data on "real" systems. Because the benefits of a reliable model for software effort and size prediction for budget and scheduling resources are numerous, HCM could be a valuable tool.

References

Albrecht, A. J. (1984, May). AD/M Productivity Measurement and Estimate Validation. Purchase, N. Y.: IBM Corp.

Albrecht, A. J., & Herron, D. A. (1990). A Functional Metric Course. Burlington, Mass.: Software Productivity Research, Inc.

Albrecht, A. J., & Gaffney, J. (1983). Software function, source lines of code, and development effort prediction. IEEE Transactions on software Engineering, 9(6), 639-648.

Benbasat, I., & Vessey, I. (1980, June). Programmer and Analyst Time/Cost Estimation. MIS Quarterly, 4(2), 31-42.

Boehm, B. W. (1987, September). Improving Software Productivity. IEEE Computer, 20(1), 43-57.

Bunge, M. (1977). Treatise on Basic Philosophy: Volume 3: Ontology I: The Furniture of the World. Boston, Mass.: Reidel Publishing Co..

Bunge, M. (1979). Treatise on Basic Philosophy: Volume 4: Ontology II: The World of Systems. Boston, Mass.: Reidel Publishing Co..

Cote, V., Bourque, P., Oligny, S., & Rivard, N. (1988, March). Software metrics: An overview of recent results. The Journal of Systems and Software, 8(2), 121-131.

Dreger, J. B. (1989). Function Point Analysis. Englewood Cliffs, N. J.: Prentice-Hall.

Garmus, D. ed. (1990, April). IFPUG Counting Practices Manual, Release 3.0. Westerville, Ohio: International Function Point User's Group.

Halstead, M. H. (1977). Elements of Software Science. New York: Elsevier.

Hellerman, L. (1972, May). A measure of computational work. IEEE Transactions on Computers, C-21(5), 439-446.

Jeffery, D. R., & Low, G. (1990, July). Calibrating estimation tools for software development. Software Engineering Journal, 215-221.

Jones, C. (1986). Programming Productivity. New York, NY: McGrawHill.

Jones, C. (1991). Applied Software Measurement. McGraw-Hill, Inc.: New York.

Kemerer, C. F. (1987, May). An empirical validation of software cost estimation models. Communications of the ACM, 30(5), 416-429.

Kemerer, C. F. (1993, February). Reliability of Function Points Measurement. Communications of the ACM, 36(2), 85-97.

Kennamer, Walter J. (1991). SNAP! [Computer program]. Version 5.0. Documentation and Development System for FoxPro, dBASE, Clipper, and Other X-Base Systems.

McCabe, T. J., & Butler, C. W. (1989, December). Design complexity measurement and testing. Communications of the ACM, 32(12), 1415-1425.

Nash, J. F., & Heagy, C. D. (1993). Accounting Information Systems (3rd),Cincinnati, Ohio: South-Western Publishing.

Paulson, D. (1989, February). Reasoning Tools to support System Analysis and Design. Unpublished doctoral dissertation, University of British Columbia.

Paulson, D., & Wand, Y. (June 1992b). Analyzing the Completeness and Consistency of Information System Specifications. Working Paper. Faculty of Commerce and Business Administration, U.B.C.

Paulson, D., & Wand, Y. (1992, March). An automated approach to information systems decomposition. IEEE Transactions on software engineering, 18(3), 174-189.

Simon, H. A. (1969). The Sciences of the Artificial, 2nd ed. Cambridge, Massachusetts: The MIT Press.

Symons, C. R. (1988, January). Function point analysis: Difficulties and improvements. IEEE Transactions on Software Engineering, 14(1), 2-11.

Wand, Y., & Weber, R. (1987, May). Formalization of Information Systems Design.

Watne, D. A., & Turney, P. B. (1990). Auditing EDP Systems, 2nd Ed. Englewood Cliffs, New Jersey: Prentice Hall.

Whitten, J. L., Bentley, L. D., & Barlow, V. M. (1989). Systems Analysis and Design Methods, 2nd ed. Homewood, IL: Irwin.

Wing, J. M. (1990, September). A Specifier's Introduction to Formal Methods. IEEE Computer, 23(9), 8-24.

Wrigley, C. D., & Dexter, A. S. (1991, June). A model for measuring information system size. MIS Quarterly, 15(2), 245-257.

Wrigley, C. D. (1988, December). A model and method for measuring information system size. Unpublished doctoral dissertation, University of British Columbia.

Appendices

Appendix A
Three Car Insurance System Varieties: Description & Stable State Space

I. Functional Form of Batch and On-Line systems

SYSTEM A) Insurance Car System: Append or Edit Client "Batch"
This Car system is a "batch entry" system in which all data on age, driver and accident of a new insurance client(s) or current client(s) are entered before the data is processed to determine output(s). The processing could occur after one record is entered or many records are entered. The flag to indicate entry is complete is 'end' flag=1.

Stable State Space for system: CAR Complexity: 21

	accident	**adjustment**	**age**	**driver**	**end**	**premium**
1.	yes	blank	over	male	0	blank
2.	no	blank	over	male	0	blank
3.	yes	blank	under	male	0	blank
4.	no	blank	under	male	0	blank
5.	yes	blank	over	female	0	blank
6.	no	blank	over	female	0	blank
7.	yes	blank	under	female	0	blank
8.	no	blank	under	female	0	blank
9.	yes	no	over	male	1	low
10.	no	yes	over	male	1	low
11.	yes	no	under	male	1	high
12.	no	yes	under	male	1	high
13.	yes	no	over	female	1	low
14.	no	yes	over	female	1	low
15.	yes	no	under	female	1	low
16.	no	yes	under	female	1	low

SYSTEM B) Insurance Car System: Edit a Client "On-Line"
This system differs from the above because it is only for editing a current client and not for appending a new client. The data on the customer (age,driver and accident) already exist and the user is changing one or more fields of data. This system is titled "on-line" because processing occurs after each change in a field ie. the age is changed from under to over. It has a smaller complexity than System A because the output fields, adjustment and premium can not be blank (a state before appending new client).

Stable State Space for system: NEWCAR

	accident	**adjustment**	**age**	**driver**	**premium**
1.	yes	no	over	male	low
2.	no	yes	over	male	low
3.	yes	no	under	male	high
4.	no	yes	under	male	high
5.	yes	no	over	female	low
6.	no	yes	over	female	low
7.	yes	no	under	female	low
8.	no	yes	under	female	low

SYSTEM C) Insurance Car System: Edit/Append a Client "On-Line"

This system is an 'on-line'example of editing or appending a new client. After each change in field, processing occurs. For example, if it is a new client, the user changes the 'driver' field from blank to female or if it is a current client, the user changes the 'accident' fields from no to yes. It has a higher complexity than System A because it has extra stable states. Extra stable states such as when only one field is entered (ie. age) and the other field is still blank (ie. driver's sex), the output field is still blank (ie, premium). Unlike System A, the input fields (driver, age, and accident) all can have blank values.

Stable State Space for system: NEWCAR3

	accident	adjustment	age	driver	premium
1.	yes	no	over	male	low
2.	no	yes	over	male	low
3.	yes	no	under	male	high
4.	no	yes	under	male	high
5.	yes	no	blank	male	blank
6.	no	yes	blank	male	blank
7.	yes	no	over	female	low
8.	no	yes	over	female	low
9.	yes	no	under	female	low
10.	no	yes	under	female	low
11.	yes	no	blank	female	blank
12.	no	yes	blank	female	blank
13.	yes	no	over	blank	blank
14.	no	yes	over	blank	blank
15.	yes	no	under	blank	blank
16.	no	yes	under	blank	blank
17.	yes	no	blank	blank	blank
18.	no	yes	blank	blank	blank

Summary of Insurance Systems

System C where the input fields can have blank values can get very complicated. In "real practise" when **appending** a new record, a programmer would not have processing until all required fields are filled and the user has verified the data correct (such as System A). For editing a record, System B, processing after each data field change, or System A, processing after all field changes, are realistic systems. It is a judgement call whether the programmer chooses a system like A or B. In conclusion, as shown from the above examples, the timing of processing can affect complexity.

Appendix B
Three Car Insurance System Varieties : Selma Specifications

II. SELMA specification of Batch and On-Line systems

SYSTEM A) Insurance Car System: Append or Edit Client "Batch"

```
/* car insurance program */
/* append or edit "batch" data entry of new or current client data*/

/* state variables */
state_variable(driver).
state_variable(age).
state_variable(accident).
state_variable(adjustment).
state_variable(premium).
state_variable(end).

/* variable values */
values(driver,[male,female]).
values(age,[over,under]).  /* over or under age 25 */
values(accident,[yes,no]).  /* accident in last five years */
values(premium,[high,low,blank]).
values(adjustment,[yes,no,blank]).
values(end,["0","1"]).  /* flag to initiate end of data entry */

/* events */
event("Begin data entry",[v(end,"0")]).
event("End of data entry",[v(end,"1")]).

/* stable states */
static("end or beginning",[v(end,"0")]).
static("end or beginning",[v(end,"1")]).


static("male or female",[v(driver,male)]).
static("male or female",[v(driver,female)]).

static("age",[v(age,over)]).
static("age",[v(age,under)]).

static("accident",[v(accident,yes)]).
static("accident",[v(accident,no)]).

static("premium calc",[v(end,"1"),v(driver,male),v(age,under),v(premium,high)]).
static("premium calc",[v(end,"1"),v(driver,female),v(age,under),v(premium,low)]).
static("premium calc",[v(end,"1"),v(age,over),v(premium,low)]).
static("premium calc",[v(end,"0"),v(premium,blank)]).

static("good-driver adjustment",[v(end,"1"),v(accident,yes),v(adjustment,no)]).
static("good-driver adjustment",[v(end,"1"),v(accident,no),v(adjustment,yes)]).
```

```
static("good-driver adjustment",[v(end,"0"),v(adjustment,blank)]).


/* corrective actions */

/* beginning of data entry */
dynamic("begin",[v(end,"0")],[v(adjustment,blank),v(premium,blank)]).

/* end of data entry */
/* calculate premium */
dynamic("premium",[v(end,"1"),v(driver,male),v(age,under)],
                  [v(premium,high)]).
dynamic("premium",[v(end,"1"),v(driver,female),v(age,under)],
                  [v(premium,low)]).
dynamic("premium",[v(end,"1"),v(age,over)],
                  [v(premium,low)]).
dynamic("premium",[v(end,"0")],
                  [v(premium,blank)]).

/* calculate good driver adjustment */
dynamic("good driver adjustment",[v(end,"1"),v(accident,no)],
                                 [v(adjustment,yes)]).
dynamic("good driver adjustment",[v(end,"1"),v(accident,yes)],
                                 [v(adjustment,no)]).

dynamic("good driver adjustment",[v(end,"0")],[v(adjustment,blank)]).
```

SYSTEM B) Insurance Car System: Edit a Client "On-Line"

```
/* car insurance program */
/* edit (no append) of current client data */


/* state variables */
state_variable(driver).
state_variable(age).
state_variable(accident).
state_variable(adjustment).
state_variable(premium).


/* variable values */
values(driver,[male,female]).
values(age,[over,under]).  /* over or under age 25 */
values(accident,[yes,no]).  /* accident in last five years */
values(premium,[high,low]).
values(adjustment,[yes,no]).


/* events */
event("Sex entry",[v(driver,male)]).
event("Sex entry",[v(driver,female)]).

event("Age entry",[v(age,over)]).
event("Age entry",[v(age,under)]).

event("Accident entry",[v(accident,yes)]).
event("Accident entry",[v(accident,no)]).


/* stable states */

static("male or female",[v(driver,male)]).
static("male or female",[v(driver,female)]).


static("age",[v(age,over)]).
static("age",[v(age,under)]).


static("accident",[v(accident,yes)]).
static("accident",[v(accident,no)]).


static("premium calc",[v(driver,male),v(age,under),v(premium,high)]).
static("premium calc",[v(driver,female),v(age,under),v(premium,low)]).
static("premium calc",[v(age,over),v(premium,low)]).


static("good-driver adjustment",[v(accident,yes),v(adjustment,no)]).
static("good-driver adjustment",[v(accident,no),v(adjustment,yes)]).
```

```
/* corrective actions */

/* beginning of data entry */

/* calculate premium */
dynamic("premium",[v(driver,male),v(age,under)],
                  [v(premium,high)]).
dynamic("premium",[v(driver,female),v(age,under)],
                  [v(premium,low)]).
dynamic("premium",[v(age,over)],
                  [v(premium,low)]).

/* calculate good driver adjustment */
dynamic("good driver adjustment",[v(accident,no)],
                                 [v(adjustment,yes)]).
dynamic("good driver adjustment",[v(accident,yes)],
                                 [v(adjustment,no)]).
```

SYSTEM C) Insurance Car System: Edit/Append a Client "On-Line"

```
/* car insurance program */
/* on-line edit or append client data */


/* state variables */
state_variable(driver).
state_variable(age).
state_variable(accident).
state_variable(adjustment).
state_variable(premium).


/* variable values */
values(driver,[male,female,blank]).
values(age,[over,under,blank]).  /* over or under age 25 */
values(accident,[yes,no,blank]).  /* accident in last five years */
values(premium,[high,low,blank]).
values(adjustment,[yes,no,blank]).


/* events */
event("Sex entry male",[v(driver,male)]).
event("Sex entry female",[v(driver,female)]).

event("Age entry over",[v(age,over)]).
event("Age entry under",[v(age,under)]).

event("Accident entry yes",[v(accident,yes)]).
event("Accident entry no",[v(accident,no)]).

/* stable states */
static("male or female",[v(driver,male)]).
static("male or female",[v(driver,female)]).
static("male or female",[v(driver,blank)]).

static("age",[v(age,over)]).
static("age",[v(age,under)]).
static("age",[v(age,blank)]).

static("accident",[v(accident,yes)]).
static("accident",[v(accident,no)]).
static("accident",[v(accident,blank)]).

static("premium calc",[v(driver,male),v(age,under),v(premium,high)]).
static("premium calc",[v(driver,male),v(age,blank),v(premium,blank)]).
static("premium calc",[v(driver,male),v(age,over),v(premium,low)]).

static("premium calc",[v(driver,female),v(age,under),v(premium,low)]).
static("premium calc",[v(driver,female),v(age,blank),v(premium,blank)]).
static("premium calc",[v(driver,female),v(age,over),v(premium,low)]).
static("premium calc",[v(driver,blank),v(premium,blank)]).
```

```
static("good-driver adjustment",[v(accident,yes),v(adjustment,no)]).
static("good-driver adjustment",[v(accident,no),v(adjustment,yes)]).


/* corrective actions */
/* beginning of data entry */
/* calculate premium */

dynamic("premium",[v(driver,male),v(age,under)],
                  [v(premium,high)]).
dynamic("premium",[v(driver,male),v(age,blank)],
                  [v(premium,blank)]).
dynamic("premium",[v(driver,male),v(age,over)],
                  [v(premium,low)]).

dynamic("premium",[v(driver,female),v(age,under)],
                  [v(premium,low)]).
dynamic("premium",[v(driver,female),v(age,blank)],
                  [v(premium,blank)]).
dynamic("premium",[v(driver,female),v(age,over)],
                  [v(premium,low)]).


/* calculate good driver adjustment */
dynamic("good driver adjustment",[v(accident,no)],
                                 [v(adjustment,yes)]).
dynamic("good driver adjustment",[v(accident,yes)],
                                 [v(adjustment,no)]).
```

Appendix C
Data for Regression Analysis (FC)

DATA - BASIS FOR REGRESSION ANALYSIS with Independent Variable : Function Count (FC)

System and Number	Function Count	Actual Total LOC	Programmed LOC	Generated LOC	System Complexity	Process Complexity	Screens	Reports
	FC	LOC - actual	LOC-prg	LOC-gen	DEC - prg	DEC - proc	SCR	REP
Car -1	38	750	269	465	25	5	2	2
Car -2	33	541	240	295	28	5	2	1
Car -3	38	710	242	460	16	4	2	2
Car -4	38	747	223	467	27	4	2	2
Car -5	33	622	332	283	32	4	2	1
Car -6	38	720	244	468	29	4	2	2
Car -7	41	807	369	433	37	5	4	1
Car -8	38	708	225	473	28	3	2	2
Car -9	33	548	223	319	28	5	2	1
Car -10	38	674	228	435	28	3	2	2
Car -11	38	811	304	499	28	5	2	2
Car -12	38	759	214	537	28	4	2	2
Car -13	41	711	315	390	31	4	4	1
Payroll-1	38	790	317	461	37	7	2	2
Payroll-2	37	575	244	326	28	5	3	1
Payroll-3	33	568	253	308	30	8	2	1
Payroll-4	38	751	282	426	33	10	2	2
Payroll-5	43	965	332	624	29	4	2	3
Payroll-6	33	546	240	299	30	6	2	1
Payroll-7	51	1302	478	796	31	7	4	3
Payroll-8	43	910	276	625	37	9	2	3
Payroll-9	37	608	236	377	32	9	3	1
Payroll-10	43	865	243	613	32	5	2	3
Payroll-11	38	704	250	447	39	9	2	2
Payroll-12	33	551	225	320	18	4	2	1
Payroll-13	41	795	330	458	41	10	4	1
Mean	38	732	274	446	30	6	2	2

(testdat2.wb1(c)):regdata)

Appendix D
"Backfiring" FP Worksheet

Calculation of SPR Complexity Adjustment Factor for Retrofitting of Function points to existing software -- "Backfiring"

Problem complexity? _____
1. Simple algorithms and simple calculations
2. Majority of simple algorithms and calculations.
3. Algorithms and calculations of average complexity.
4. Some difficult or complex calculations.
5. Many difficult algorithms and complex calculations.

Data complexity? _____
1. Simple data with few variables and low complexity
2. Numerous variables, but simple data relationship.
3. Multiple files, fields, and data interactions.
4. Complex file structures and data interactions.
5. Very complex file structures and data interactions.

Code complexity? _____
1. Nonprocedural.
2. Well structured with reusable modules.
3. Well structures (small modules and simple paths).
4. Fair structure, but some complex modules and paths.
5. Poor structure, with large modules and complex paths.

Sum of Problem, Code and Data complexity _____

The SPR Complexity Adjustment Factors

Complexity sum	Adjustment multiplier
3	0.70
4	0.75
5	0.80
6	0.85
7	0.90
8	0.95
9	1.00
10	1.05
11	1.10
12	1.15
13	1.20
14	1.25
15	1.30

Appendix E
FP component definitions
[Jones, 1991, p. 68-69]

Inputs Inputs are screens or forms through which human users of an application or other programs add new data or update existing data. If an input screen is too large for a single normal display (usually 80 columns by 25 lines) and flows over onto a second screen, the set counts as 1 input. Inputs that require unique processing are what should be considered.

Outputs: Outputs are screens or reports which the application produces for human use or for other programs. Note that outputs requiring separate processing are the units to count: In a payroll application, an output function that created, say, 100 checks would still count as one output.

Inquiries: Inquiries are screens which allow users to interrogate an application and ask for assistance or information, such as HELP screens.

Data files: Data files are logical collections of records which the application modifies or updates. A file can be a flat file such as a tape file, one leg of a hierarchical database such as IMS, on table within a relational database, or one path through a CODASYL network database.

Interface: Interfaces are files shared with other applications, such as incoming or outgoing tape files, shared databases, and parameter lists.

Appendix F
FP to LOC ratios
[Jones, 1991, p. 76]

Language	Level	Source statements per function point
1. Low-level default	1.0	320
2. Machine language	1.0	320
3. First generation default	1.0	320
4. Basic assembly default	1.0	320
5. Macro assembly default	1.5	213
6. C default	2.5	128
7. Interpreted Basic default	2.5	128
8. Fortran II	2.5	128
9. Fortran 66	2.5	128
10. Second generation default	3.0	105
11. Procedural language default	3.0	105
12. Fortran 77	3.0	105
13. Algol 68	3.0	105
14. Algol W	3.0	105
15. Chill	3.0	105
16. ANSI Cobol 74	3.0	105
17. Coral 66	3.0	105
18. Jovial	3.0	105
19. Strongly typed default	3.0	105
20. ANSI Cobol 85	3.5	91
21. Pascal default	3.5	91
22. Compiled Basic default	3.5	91
23. PL/S	3.5	91
24. High-level default	3.5	91
25. Third generation default	3.5	91
26. Report generator default	4.0	80
27. PL/I	4.0	80
28. Modula 2	4.0	80
29. Problem-oriented default	4.5	71
30. Ada	4.5	71
31. Weakly typed default	5.0	64
32. Prolog	5.0	64
33. Lisp	5.0	64
34. Forth	5.0	64
35. ANSI/Quick/Turbo Basic	5.0	64
36. English-like default	6.0	51
37. AI shell default	6.5	49
38. Simulation default	7.0	46
39. Decision table default	7.0	46
40. Database default	8.0	40
41. Nonprocedural default	9.0	35
42. Decision support default	9.0	35
43. Statistical language default	10.0	32
44. APL	10.0	32
45. Object-oriented default	11.0	29
46. Fourth generation default	16.0	20
47. Program generator default	20.0	16
48. Query language default	25.0	13
49. Spreadsheet default	50.0	6
50. Fifth generation default (graphic icons)	75.0	4

Appendix G
Selma specification & Decomposition of Car Insurance Project

DECOMPOSITION

Decomposition #1 Complexity = 17.25
1: {accident,end,adjustment} {age,driver,end,premium}

SELMA SPECIFICATION

```
/* car insurance program */
/* append or edit "batch" data entry of new or current client data*/

/* state variables */
state_variable(driver).
state_variable(age).
state_variable(accident).
state_variable(adjustment).
state_variable(premium).
state_variable(end).

/* variable values */
values(driver,[male,female]).
values(age,[over,under]).  /* over or under age 25 */
values(accident,[yes,no]).  /* accident in last five years */
values(premium,[high,low,blank]).
values(adjustment,[yes,no,blank]).
values(end,["0","1"]).  /* flag to initiate end of data entry */

/* events */
event("Begin data entry",[v(end,"0")]).
event("End of data entry",[v(end,"1")]).

/* stable states */
static("end or beginning",[v(end,"0")]).
static("end or beginning",[v(end,"1")]).


static("male or female",[v(driver,male)]).
static("male or female",[v(driver,female)]).

static("age",[v(age,over)]).
static("age",[v(age,under)]).

static("accident",[v(accident,yes)]).
static("accident",[v(accident,no)]).


static("premium calc",[v(end,"1"),v(driver,male),v(age,under),v(premium,high)]).
static("premium calc",[v(end,"1"),v(driver,female),v(age,under),v(premium,low)]).
static("premium calc",[v(end,"1"),v(age,over),v(premium,low)]).
static("premium calc",[v(end,"0"),v(premium,blank)]).
```

```
static("good-driver adjustment",[v(end,"1"),v(accident,yes),v(adjustment,no)]).
static("good-driver adjustment",[v(end,"1"),v(accident,no),v(adjustment,yes)]).
static("good-driver adjustment",[v(end,"0"),v(adjustment,blank)]).


/* corrective actions */

/* beginning of data entry */
dynamic("begin",[v(end,"0")],[v(adjustment,blank),v(premium,blank)]).

/* end of data entry */
/* calculate premium */
dynamic("premium",[v(end,"1"),v(driver,male),v(age,under)],
                  [v(premium,high)]).
dynamic("premium",[v(end,"1"),v(driver,female),v(age,under)],
                  [v(premium,low)]).
dynamic("premium",[v(end,"1"),v(age,over)],
                  [v(premium,low)]).
dynamic("premium",[v(end,"0")],
                  [v(premium,blank)]).

/* calculate good driver adjustment */
dynamic("good driver adjustment",[v(end,"1"),v(accident,no)],
                                 [v(adjustment,yes)]).
dynamic("good driver adjustment",[v(end,"1"),v(accident,yes)],
                                 [v(adjustment,no)]).

dynamic("good driver adjustment",[v(end,"0")],[v(adjustment,blank)]).
```

Appendix H
Selma Specification & Decomposition of Payroll Project

DECOMPOSITION

Decomposition #1 Complexity = 23.09
2: {base,comm,overtime,total_pay}
1: {end,hours,payrate,base} {emp_pos,emp_type,end,sales,comm}
{emp_pos,emp_type,end,hours,overtime}

SELMA SPECIFICATION

```
/*     Event Definitions      */
event("End of Period",[v(end,"0")]).
event("End of Period",[v(end,"1")]).

/*     State Variable Definitions     */
state_variable(end).          /* end of period flag */
state_variable(emp_type).      /* employee type  */
state_variable(emp_pos).       /* employee position  */
state_variable(payrate).       /* pay rate   */
state_variable(sales).         /* sales */
state_variable(hours).         /* hours worked */
state_variable(base).          /* base pay  */
state_variable(overtime).      /* overtime pay  */
state_variable(total_pay).     /* total pay */
state_variable(comm).          /* commission */

/*     State Variable    */
values(end,["0","1"]).
values(emp_type,[off,sal]).
values(emp_pos,[reg,mgt]).
values(payrate,[zero,nz]).
values(sales,[zero,nz]).
values(hours,[zero,reg,ot]).

values(base,[zero,nz]).
values(overtime,[zero,nz]).
values(total_pay,[zero,nz]).
values(comm,[zero,nz]).

/*     Stability Conditions   */

/* Base salary, overtime, commissions and benefits are not calculated
   except at EOP.  */

static("EOP requirements",[v(end,"0")]).
static("EOP requirements",[v(end,"1")]).

/* An employee may be in a regular or management position */

static("regular or management",[v(emp_pos,reg)]).
static("regular or management",[v(emp_pos,mgt)]).

/* An employee may have either an office or sales job  */
```

```
static("office or sales",[v(emp_type,off)]).
static("office or sales",[v(emp_type,sal)]).

/* Pay rate can be zero or nonzero  */
static("pay rate",[v(payrate,zero)]).
static("pay rate",[v(payrate,nz)]).

/* hours may be zero, regular or overtime  */
static("hours",[v(hours,ot)]).   /*  overtime > 40 hours */
static("hours",[v(hours,reg)]).  /*  regular >0 .and. <=40  hours a week */
static("hours",[v(hours,zero)]).


/* Sales may be zero or nonzero  */
static("sales",[v(sales,zero)]).
static("sales",[v(sales,nz)]).
```

```
/* Nonmanagement office staff are entitled to overtime if hours is overtime  */
static("regular staff gets overtime",
        [v(end,"1"),v(emp_pos,reg),v(emp_type,off),v(hours,ot),v(overtime,nz)]).
static("regular staff gets overtime",
        [v(end,"1"),v(hours,reg),v(overtime,zero)]).
static("regular staff gets overtime",
        [v(end,"1"),v(hours,zero),v(overtime,zero)]).
static("regular staff gets overtime",
        [v(end,"1"),v(emp_pos,mgt),v(overtime,zero)]).
static("regular staff gets overtime",
        [v(end,"1"),v(emp_type,sal),v(overtime,zero)]).
static("regular staff gets overtime",
        [v(end,"0"),v(overtime,zero)]).

/* Nonmanagement SALES staff are entitled to commission is sales are nonzero */
static("nonmgt sales staff gets commissions",
        [v(end,"1"),v(emp_type,sal),v(emp_pos,reg),v(sales,nz),v(comm,nz)]).
static("nonmgt sales staff gets commissions",
        [v(end,"1"),v(emp_pos,mgt),v(comm,zero)]).
static("nonmgt sales staff gets commissions",
        [v(end,"1"),v(emp_type,off),v(comm,zero)]).
static("nonmgt sales staff gets commissions",
        [v(end,"1"),v(sales,zero),v(comm,zero)]).
static("nonmgt sales staff gets commissions",
        [v(end,"0"),v(comm,zero)]).

/* All employees are entitled to base pay if hours and pay rate are not zero */
static("everyone gets base pay",
        [v(end,"1"),v(hours,ot),v(payrate,nz),v(base,nz)]).
static("everyone gets base pay",
        [v(end,"1"),v(hours,reg),v(payrate,nz),v(base,nz)]).
static("everyone gets base pay",
        [v(end,"1"),v(hours,zero),v(base,zero)]).
static("everyone gets base pay",
        [v(end,"1"),v(payrate,zero),v(base,zero)]).
static("everyone gets base pay",
        [v(end,"0"),v(base,zero)]).


/* Total pay must be calculated at EOP  */
static("total pay",[v(end,"1"),v(base,nz),v(total_pay,nz)]).
static("total pay",[v(end,"1"),v(overtime,nz),v(total_pay,nz)]).
static("total pay",[v(end,"1"),v(comm,nz),v(total_pay,nz)]).
static("total pay",[v(end,"1"),v(base,zero),v(overtime,zero),v(comm,zero),
        v(total_pay,zero)]).
static("total pay",[v(end,"0"),v(total_pay,zero)]).

/* CORRECTIVE ACTIONS  */

/* At start of period all calculated values must be reset to zero. */
dynamic("SOP",[v(end,"0")],
        [v(overtime,zero),v(base,zero),v(comm,zero),v(total_pay,zero)]).

/* At end of period calculate base pay  */
dynamic("calculate base pay",
```

```
        [v(end,"1"),v(hours,ot),v(payrate,nz)],
        [v(base,nz)]).
dynamic("calculate base pay",
        [v(end,"1"),v(hours,reg),v(payrate,nz)],
        [v(base,nz)]).
dynamic("calculate base pay",
        [v(end,"1"),v(hours,zero)],
        [v(base,zero)]).
dynamic("calculate base pay",
        [v(end,"1"),v(payrate,zero)],
        [v(base,zero)]).

/* At end of period calculate overtime  */
dynamic("calculate overtime",
        [v(end,"1"),v(emp_pos,reg),v(emp_type,off),v(hours,ot)],
        [v(overtime,nz)]).
dynamic("calculate overtime",
        [v(end,"1"),v(hours,reg)],
        [v(overtime,zero)]).
dynamic("calculate overtime",
        [v(end,"1"),v(hours,zero)],
        [v(overtime,zero)]).
dynamic("calculate overtime",
        [v(end,"1"),v(emp_pos,mgt)],
        [v(overtime,zero)]).
dynamic("calculate overtime",
        [v(end,"1"),v(emp_type,sal)],
        [v(overtime,zero)]).

/* At end of period calculate commissions */
dynamic("calculate commissions",
        [v(end,"1"),v(emp_type,sal),v(emp_pos,reg),v(sales,nz)],
        [v(comm,nz)]).
dynamic("calculate commissions",
        [v(end,"1"),v(emp_pos,mgt)],
        [v(comm,zero)]).
dynamic("calculate commissions",
        [v(end,"1"),v(emp_type,off)],
        [v(comm,zero)]).
dynamic("calculate commissions",
        [v(end,"1"),v(sales,zero)],
        [v(comm,zero)]).


/* At end of period, calculate total pay */
dynamic("calculate total pay",[v(end,"1"),v(base,nz)],
        [v(total_pay,nz)]).
dynamic("calculate total pay",[v(end,"1"),v(overtime,nz)],
        [v(total_pay,nz)]).
dynamic("calculate total pay",[v(end,"1"),v(comm,nz)],
        [v(total_pay,nz)]).
dynamic("calculate total pay",[v(end,"1"),v(base,zero),v(overtime,zero),v(comm,zero)],
        [v(total_pay,zero)]).
```

Appendix I
Selma Specification & Decomposition of Hotel Project

DECOMPOSITION

Decomposition #1 Complexity = 43.90
3: {discount,roomstatus,totalbill}
2: {guest,roomstatus,discount} {roomstatus,roomstyle,season,roombill}
1: {check,roomstatus}

SELMA SPECIFICATION

```
/* Events */

event("guest check in",[v(check,in)]).
event("guest check out",[v(check,out)]).


/* State Variable Definitions */
state_variable(check).
state_variable(roomstyle).
state_variable(roomstatus).
state_variable(season).
state_variable(guest).
state_variable(roombill).
state_variable(discount).
state_variable(totalbill).

/* State Variable Value Definitions */
values(check,[in,out]).

values(roomstyle,[regular,suite,honeymoon]).
values(season,[high,low]).
values(guest,[normal,business,employee,patient]).

values(roomstatus,[open,occupied]).
values(discount,[yes,no,all]).
values(roombill,[A,B,C,D,E,F,zero]).
values(totalbill,[nz,zero]).

/* Stability Conditions */

/* (events) guest can either be checked-in or checked-out */
static("guest check",[v(check,in)]).
static("guest check",[v(check,out)]).

/* roomstyle is in three categories: regular, suite and special */
static("regular, suite & special",[v(roomstyle,regular)]).
static("regular, suite & special",[v(roomstyle,suite)]).
static("regular, suite & special",[v(roomstyle,honeymoon)]).

/* the hotel season is either high or low */
static("high or low season",[v(season,high)]).
static("high or low season",[v(season,low)]).
```

```
/* the guest can be a regular, business, employee or a "free-bee" patient */
static("guest type",[v(guest,normal)]).
static("guest type",[v(guest,business)]).
static("guest type",[v(guest,employee)]).
static("guest type",[v(guest,patient)]).

/* the guest's checked-in room gets an occupied status */
static("room status",[v(check,in),v(roomstatus,occupied)]).
static("room status",[v(check,out),v(roomstatus,open)]).

/* guest discount */
static("discount",[v(check,out),v(guest,normal),v(discount,no)]).
static("discount",[v(check,out),v(guest,patient),v(discount,all)]).
static("discount",[v(check,out),v(guest,business),v(discount,yes)]).
static("discount",[v(check,out),v(guest,employee),v(discount,yes)]).
static("discount",[v(check,in),v(discount,no)]).



/* guest room bill */
static("guest room bill",[v(check,out),v(roomstyle,regular),v(season,high),
                v(roombill,C)]).
static("guest room bill",[v(check,out),v(roomstyle,suite),v(season,high),
                v(roombill,B)]).
static("guest room bill",[v(check,out),v(roomstyle,honeymoon),v(season,high),
                v(roombill,A)]).
static("guest room bill",[v(check,out),v(roomstyle,regular),v(season,low),
                v(roombill,F)]).
static("guest room bill",[v(check,out),v(roomstyle,suite),v(season,low),
                v(roombill,E)]).
static("guest room bill",[v(check,out),v(roomstyle,honeymoon),v(season,low),
                v(roombill,D)]).
static("guest room bill",[v(check,in),v(roombill,zero)]).


/* guest total bill */
static("total bill",[v(check,out),v(discount,yes),v(totalbill,nz)]).
static("total bill",[v(check,out),v(discount,no),v(totalbill,nz)]).
static("total bill",[v(check,out),v(discount,all),v(totalbill,zero)]).
static("total bill",[v(check,in),v(totalbill,zero)]).


/* corrective actions */

/* When the guest checks in all calculated values are set to zero
  and the guest's room becomes occupied */
dynamic("checks in",[v(check,in)],
      [v(roombill,zero),v(discount,no),v(totalbill,zero),v(roomstatus,occupied)]).


/* When a guest checks-out room becomes open */
dynamic("room becomes open",[v(check,out)],[v(roomstatus,open)]).


/* When a guest checks-out -- Calculate roombillcharge */
dynamic("calc room bill",[v(check,out),v(roomstyle,regular),v(season,high)],
```

```
                    [v(roombill,C)]).
dynamic("calc room bill",[v(check,out),v(roomstyle,suite),v(season,high)],
                    [v(roombill,B)]).
dynamic("calc room bill",[v(check,out),v(roomstyle,honeymoon),v(season,high)],
                    [v(roombill,A)]).
dynamic("calc room bill",[v(check,out),v(roomstyle,regular),v(season,low)],
                    [v(roombill,F)]).
dynamic("calc room bill",[v(check,out),v(roomstyle,suite),v(season,low)],
                    [v(roombill,E)]).
dynamic("calc room bill",[v(check,out),v(roomstyle,honeymoon),v(season,low)],
                    [v(roombill,D)]).

/* When a guest checks-out -- Calculate appropriate discount */
dynamic("calc discount",[v(check,out),v(guest,normal)],
      [v(discount,no)]).
dynamic("calc discount",[v(check,out),v(guest,patient)],
      [v(discount,all)]).
dynamic("calc discount",[v(check,out),v(guest,business)],
      [v(discount,yes)]).
dynamic("calc discount",[v(check,out),v(guest,employee)],
      [v(discount,yes)]).

/* Calculate guest total bill */
dynamic("total bill",[v(check,out),v(discount,no)],[v(totalbill,nz)]).
dynamic("total bill",[v(check,out),v(discount,yes)],[v(totalbill,nz)]).
dynamic("total bill",[v(check,out),v(discount,all)],[v(totalbill,zero)]).
```

Appendix J
System Analysis for Student Systems

ASSIGNMENT #1
PROTOTYPING

Class	:	Management 4841 System Design
Date	:	Thursday January 12, 1995
Due Date	:	Thursday January 26, 1995
Subject	:	Prototype three small information systems

1. Car Insurance System
2. Payroll System
3. Hotel Registration System

You are working for XYZ System Development Company. The system analysis has been completed on three current projects. The XYZ Company wishes to prototype all three projects for their clients. You must follow COMPANY POLICY for each prototype.

"A prototype is an actual working model of the system, including sufficient functionality to allow the model to be used in a "live" setting. It is also built and revised fairly rapidly - in a matter of days or weeks, not months or years. Models are certainly not built using conventional programming languages and file access techniques. Prototype requires a set of interactive software development tools that allow the designer or even the user to rather quickly define screens, create files and data entry/update routines, generate program modules that handle processing logic, create basic query and reporting functions, and so on. These tools should be integrated around a common data dictionary that maintains the definitions of such things as data elements, edit rules, records, processing modules, screens, reports - in short, all the components of the prototype. ...[Powers, Cheney & Crow, Structured Systems Development, p.54]"

XYZ SYSTEM DEVELOPMENT COMPANY POLICY

Language
dBASE IV in DOS
Employ the report generator and the screen generator. DO NOT use the program generator.

Modular Construction
Modular Construction is employed throughout the prototype system. In order to save prototype development time, the company reuses library code. See Page 3 for a sample of the modular menu system. See Page 4 for a description of the library program files.

- Download the files onto one HD disk. Access the network and choose the COPY FILES option. Choose the PROTOTYPE option. Three directories will be created on your disk and the appropriate files will be downloaded into each subdirectory.
  a:\car a:\payroll a:\hotel

- Prototype the systems in the following order:
  1. Car Insurance System
  2. Payroll System
  3. Hotel Registration System

  Create necessary screens, reports and processing code. See Page 4 for more detailed instructions.

- Do NOT COPY other student's screen formats, report formats, or code. This is an individual project. You will receive a mark of ZERO if any copying is identified.

- Carefully follow the system analysis documentation for each case.

- Run the test data provided. Make sure your prototype works!

- Hand-in one HD disk (2 if necessary). Hand-in Project questionnaire.

- No written documentation is required for this assignment. Please note the DESIGN PROJECT will require written documentation. Insert brief internal documentation where appropriate.

MODULAR STANDARD MENU CONSTRUCTION

Modular Design : The XYZ company employs the same standard menu for prototype designing. Label and the Utilities menus are excluded for this assignment.

Updates	Action/Reports	Labels	Utilities	Exit
Database1		Report1		
Database2		Report2		
Database3		Exit		
Exit				

LIBRARY CODE

The reusable library code for each project consists of the below files. The necessary modifications for each case are highlighted in italics.

CONTROL.PRG Initiates each prototype. Calls the environment setting program (central.prg). Activates main menu.

CENTRAL.PRG Sets the environment settings and defines public variables. Calls the DEFPOP.PRG program which defines the main bar menu and all related popup menus.

DEFPOP.PRG Includes the code for the main bar menu and corresponding popup menus. *You are responsible for the design and code of the popup menu for ACTIONS/REPORTS since this is unique for each prototype. Modify the MOD2 popup menu and add appropriate submenus if necessary.*

MODULE1.PRG Includes the standard update menus and corresponding actions for each database. Modify the code to include your screen formats for each edit, and append option. The screen formats should include editing/append requirements such as all capital letters, only Y or N, and so on.

MODULE2.PRG This program contains the outline code for the menu (Mod2) you shall define in DEFPOP.prg. *All ACTIONS/REPORTS should be called from the module2.prg. Call your generated reports and your processing programs within module2.prg. ALL WRITTEN PROCESSING CODE should be written in procedure files in module2.prg.*

MODULE5.PRG This program runs the library exit program. Modification is not necessary.

ASSIGNMENT #1 QUESTIONNAIRE Name:_____

Student I.D. Number: _____ Major:_____

Computer and M.I.S. courses:	Check if Completed?	Semester?
1.Management 3820, Database Management	☐	_____
2.Management 4840, System Analysis	☐	_____
3.Management 3820, Business Data Processing	☐	_____
4.Management 2061, Microcomputers in Business	☐	_____
5.Comp Sci 1620(1600),Intro. to a Programming Lang.	☐	_____

Please list other completed Computer Science or M.I.S. courses: _____

_____

_____

6. Have you completed a COOP experience semester related to MIS or Computer Science? ☐Yes ☐No

7. Have you had any programming experience other than in university or college courses? ☐Yes ☐No

If answered "Yes", please briefly describe your experience.

_____

_____

_____

8.DURATION OF Project TIME BREAK-DOWN. Specify to nearest 1/2 hour.

Project	Learning dBASE IV	Understanding System Requirements	Programming
Car Insurance System			
Payroll System			
Hotel System			

9. The sequence you completed the projects.
Car Insurance System (1,2,or 3) _____
Payroll System (1,2, or 3) _____
Hotel System (1,2, or 3) _____

10. Comments about your experience with this assignment?

CAR INSURANCE PROTOTYPE SYSTEM

Description of a simple car insurance system

The customer makes an application request and provides personal information (if a new customer or if the data has changed), when they want to start a policy, and the length of the policy. Once the insurance company employee has entered the necessary input data, a policy is processed. The customer receives a coverage card.

Data Dictionary

Data Stores (Databases) *.dbf

(The databases are already created in the project directory.)

Structure for database : INSUREE.DBF

Field	Field name	Type	Width	Dec
1	INSUREE	Numeric	4	
2	LASTNAME	Character	25	
3	FIRSTNAME	Character	25	
4	BIRTH	Date	8	
5	STREET	Character	50	
6	CITY	Character	50	
7	PROVINCE	Character	2	
8	POSTALCODE	Character	6	
9	COUNTRY	Character	2	
10	TELEPHONE	Numeric	10	
11	SEX	Character	1	

Structure for database : POLICY.DBF

Field	Field name	Type	Width	Dec
1	POLICYNO	Numeric	6	
2	INSUREE	Numeric	4	
3	WRITEUP	Date	8	
4	BEGINDATE	Date	8	
5	ENDDATE	Date	8	
6	ACCIDENT	Numeric	1	
7	PREMIUM	Numeric	7	2
8	ADJUSTMENT	Numeric	7	2

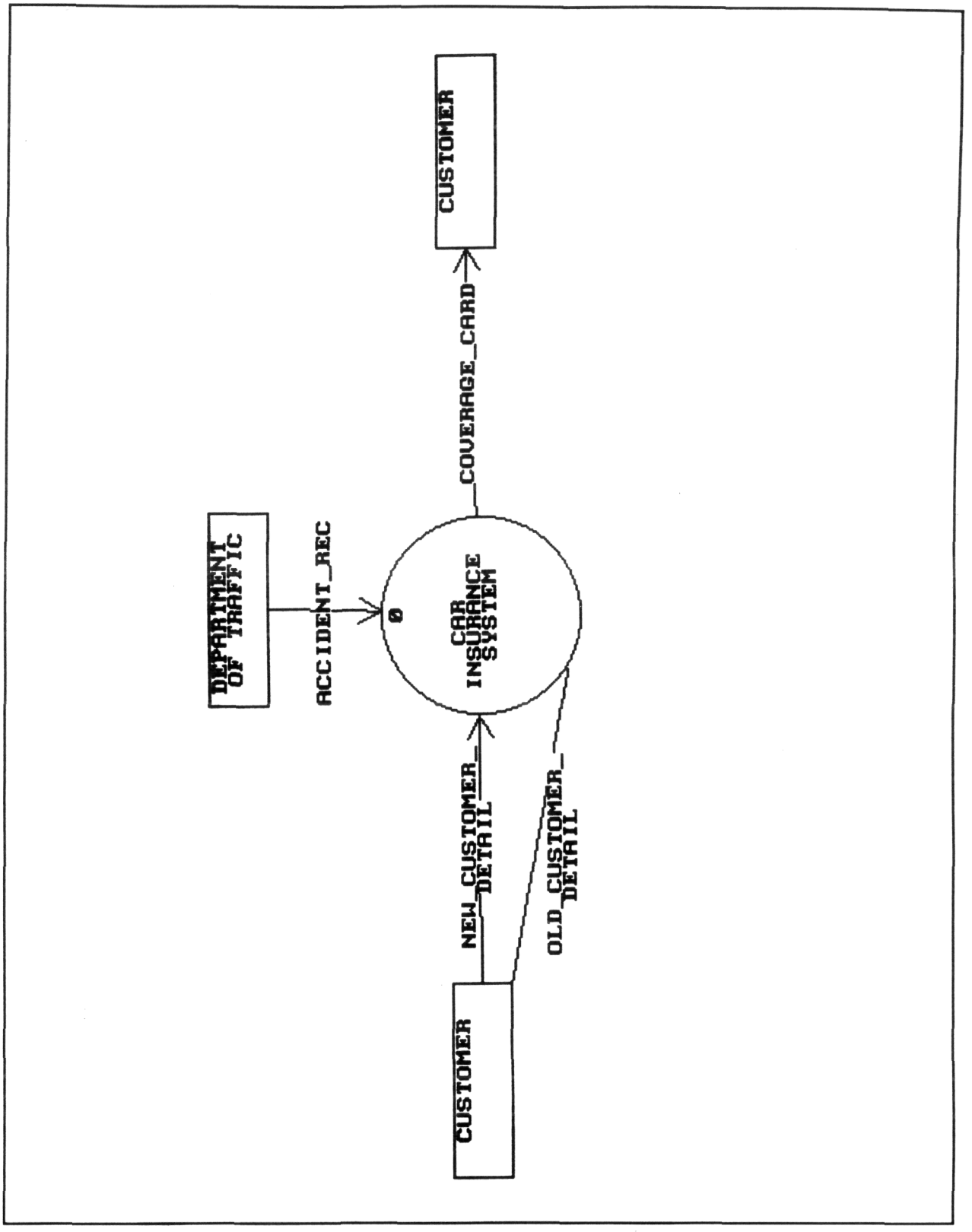

Figure 4 Context Diagram - Car insurance system

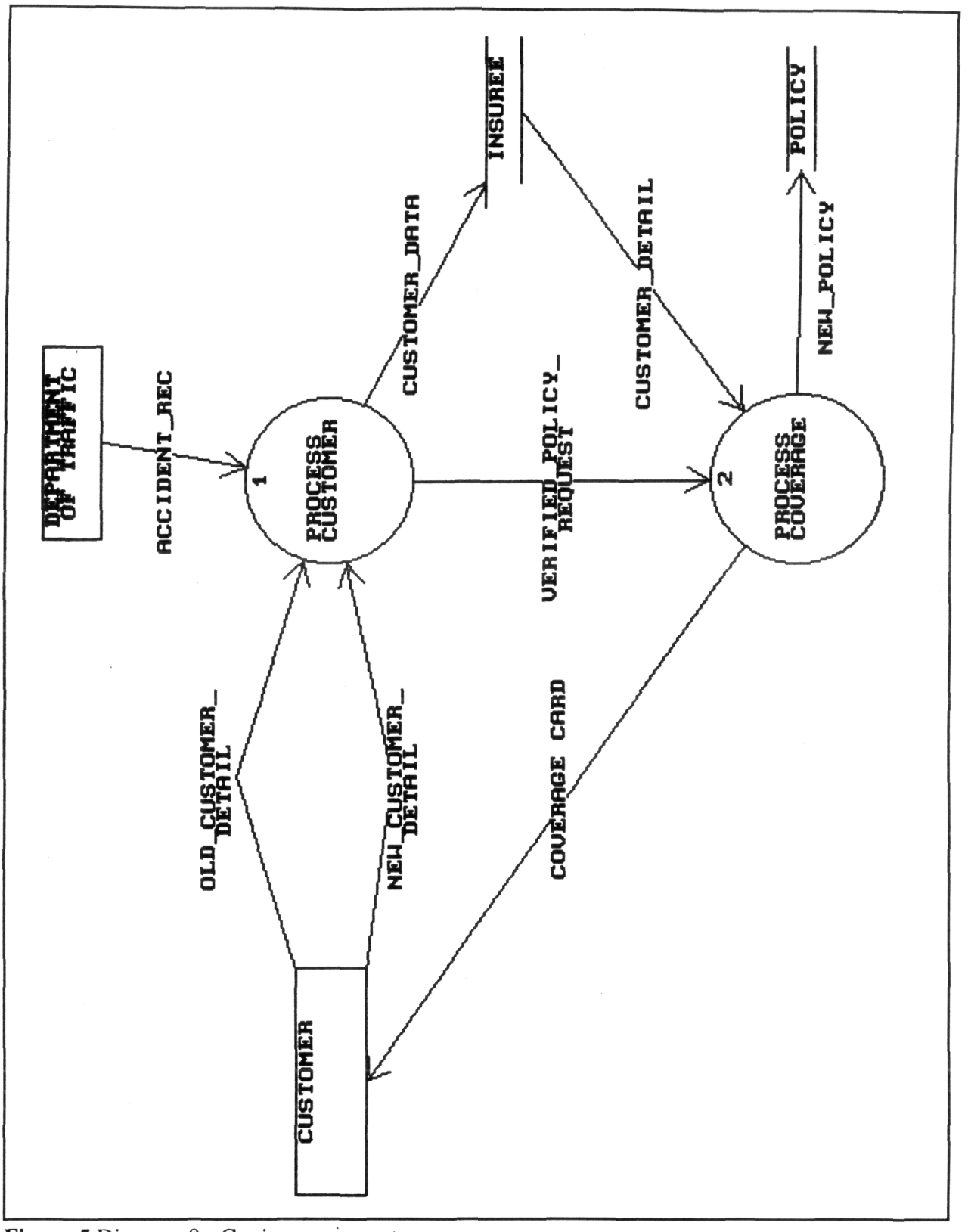

Figure 5 Diagram 0 - Car insurance system

Data Structures

ADDRESS = STREET + CITY + PROVINCE + POSTALCODE + COUNTRY.

PERSON = INSUREE + LASTNAME + FIRSTNAME + BIRTH + ADDRESS + TELEPHONE + SEX.

MOD_PERSON = INSUREE + (LASTNAME) + (FIRSTNAME) + (BIRTH) + (ADDRESS) + (TELEPHONE) + (SEX).

POLICY = POLICYNO + INSUREE + WRITEUP + BEGINDATE + ENDDATE + ACCIDENT + PREMIUM + ADJUSTMENT.

Data Flows

OLD_CUSTOMER_DETAIL = MOD_PERSON + ACCIDENT + BEGINDATE + ENDDATE.

NEW_CUSTOMER_DETAIL = PERSON + ACCIDENT + BEGINDATE + ENDDATE.

ACCIDENT_REC = LASTNAME + FIRSTNAME + VERIFIED_ACCIDENT_REC.

VERIFIED_POLICY_REQUEST = INSUREE + BEGINDATE + ENDDATE + VERIFIED_ACCIDENT_REC.

CUSTOMER_DETAIL = PERSON.

CUSTOMER_DATA = MOD_PERSON.

NEW_POLICY = POLICY.

COVERAGE_CARD = PERSON + POLICY.

Process Documentation

Process Two Process Coverage

Purpose: Calculate policy premium and adjustment.

Inputs: VERIFIED_POLICY_REQUEST Outputs: COVERAGE_CARD
NEW_POLICY

Process Definition:
After the employee has entered the data -->

Calculate:
ADJUSTMENT as a function of ACCIDENT RECORD.
If the client has an accident, their is a 200.00 adjustment.

PREMIUM as a function of SEX & AGE.
Premium is either high ($1000) a year or low ($500) a year.

- Sex is either (F)emale or (M)ale.
- Age is either (O)ver age of 25 and (U)nder age of 25 or age 25

Decision Table

Conditions	1	2	3	4
Sex	F	F	M	M
Age	O	U	O	U
Actions				
High Premium				X
Low Premium	X	X	X	

Process One Process Customer

Purpose: Process previous or new customer and the policy request.
Verify whether the customer has had an accident in the last 5 years.

Inputs: OLD_CUSTOMER_DETAIL Outputs: COVERAGE_CARD
NEW_CUSTOMER_DETAIL NEW_POLICY

Data Elements

Field Name	Type(1)	Len	Dec	Database	Brief Description
ACCIDENT	L	1	0	POLICY.DBF	Insuree had an accident inlast 5 years.
ADJUSTMENT	N	5	2	POLICY.DBF	Insurance adjustment amount
BEGINDATE	D	8	0	POLICY.DBF	Policy start date
BIRTH	D	8	0	INSUREE.DBF	Insuree birthdate
CITY	C	50	0	INSUREE.DBF	
COUNTRY	C	2	0	INSUREE.DBF	
ENDDATE	D	8	0	POLICY.DBF	Policy end date
FIRSTNAME	C	25	0	INSUREE.DBF	
INSUREE	N	4	0	INSUREE.DBF POLICY.DBF	Insuree Number
LASTNAME	C	25	0	INSUREE.DBF	
POSTALCODE	C	6	0	INSUREE.DBF	
POLICYNO	N	8	0	POLICY.DBF	Policy identification number
PREMIUM	N	5	2	POLICY.DBF	Policy premium
PROVINCE	C	2	0	INSUREE.DBF	
SEX	C	1	0	INSUREE.DBF	
STREET	C	50	0	INSUREE.DBF	
TELEPHONE	N	10	0	INSUREE.DBF	
VERIFIED_ACCIDENT_REC	L			--------	Whether the applicant had an accident in the last 5 years.
WRITEUP	D	8	0	POLICY.DBF	Policy writeup date

(1) C=CHARACTER, N=NUMERIC, L=LOGICAL, D=DATE

Test Data

The below customers have all had car insurance policies at this company in the past. The customers are already in the insuree database. The accident element (whether they had an accident in the last 5 years) has been verified by the traffic department. All these customers want car insurance for 1995.

INSUREE	LASTNAME	ACCIDENT	BEGINDATE	ENDDATE
1	DAVIDSON	Y	01/01/95	01/01/96
2	DAWSON	N	01/01/95	01/01/96
3	DAWSON	Y	01/01/95	01/01/96
4	DAY	Y	01/01/95	01/01/96
5	DANGERFIELD	N	01/01/95	01/01/96
6	DANIELSON	N	01/01/95	01/01/96
7	MICHEL	N	01/01/95	01/01/96
8	MILES	N	01/01/95	01/01/96
9	MESSENGER	N	01/01/95	01/01/96
10	SCHEBEL	Y	01/01/95	01/01/96

PAYROLL PROTOTYPE SYSTEM

1. Description of a payroll system for a company

(Quotation from Thesis: Reasoning Tools to Support Systems Analysis and Design, February 1989, Dan Paulson)

> "The company has two types of jobs: office and sales. An employee may be in either a regular or in a managerial position. Salaries are comprised of base pay, overtime pay and commissions. The way in which total salary is calculated depends on the job type and employee position. Company policy is as follows:
>
> - the office staff is entitled to overtime pay but not to commissions.
> - the sales staff is entitled to commissions but not to overtime pay.
> - managers are not entitled to overtime pay nor commissions.
> - hours and sales are recorded for all employees. (This might happen if managers are required to report hours and office workers may take a telephone order.)
>
> Also assume that all payroll processing takes place at the end of some period"

2. General Description

At the end of a period (ie. end of the month) all personnel hand-in a time sheet. The time sheet data is entered into the system by the accountant. Then the accountant initiates the pay check processing for the period. The processing **MUST** be programmed according to the below documentation.

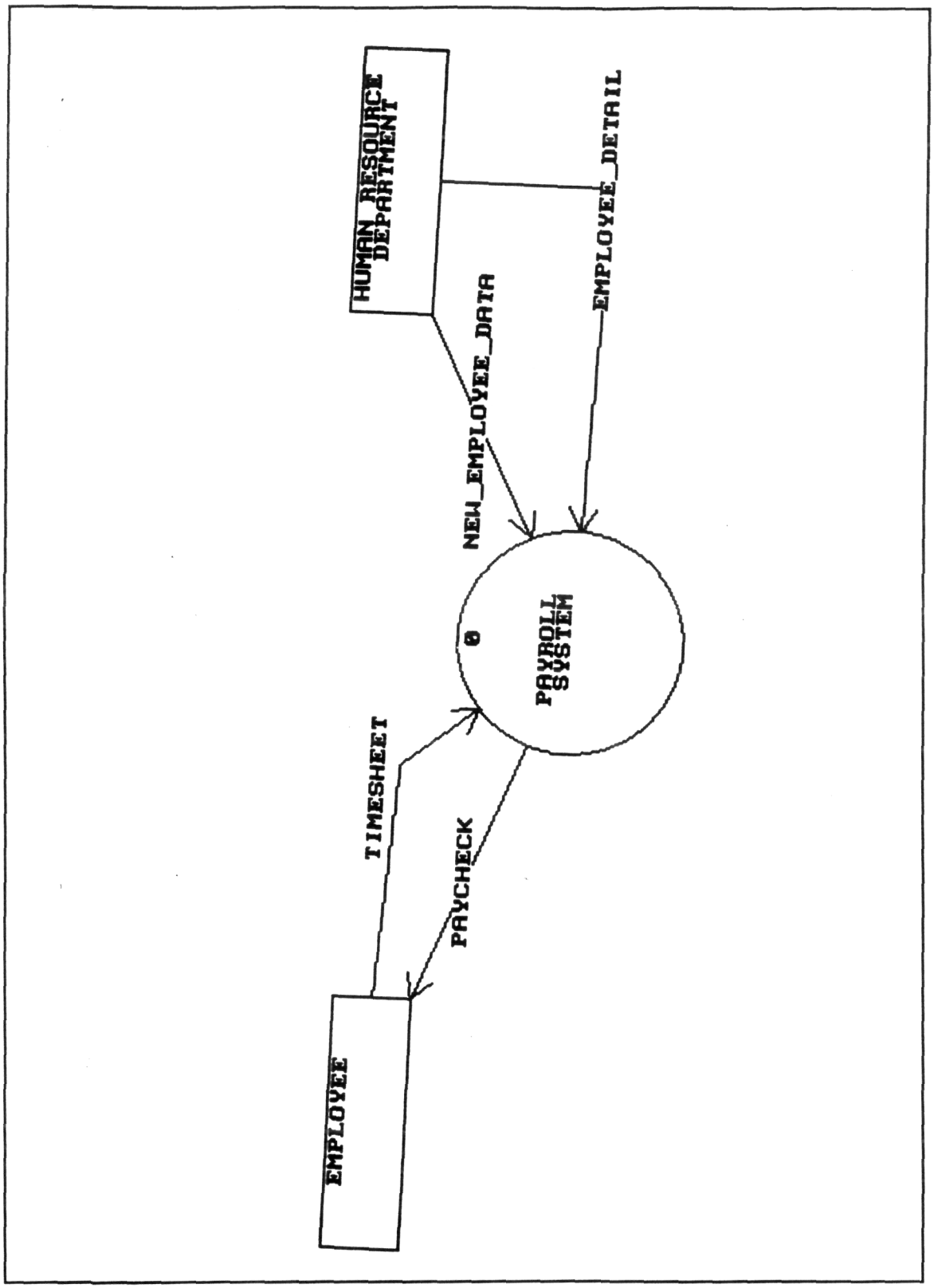

Figure 6 Context Diagram - Payroll system

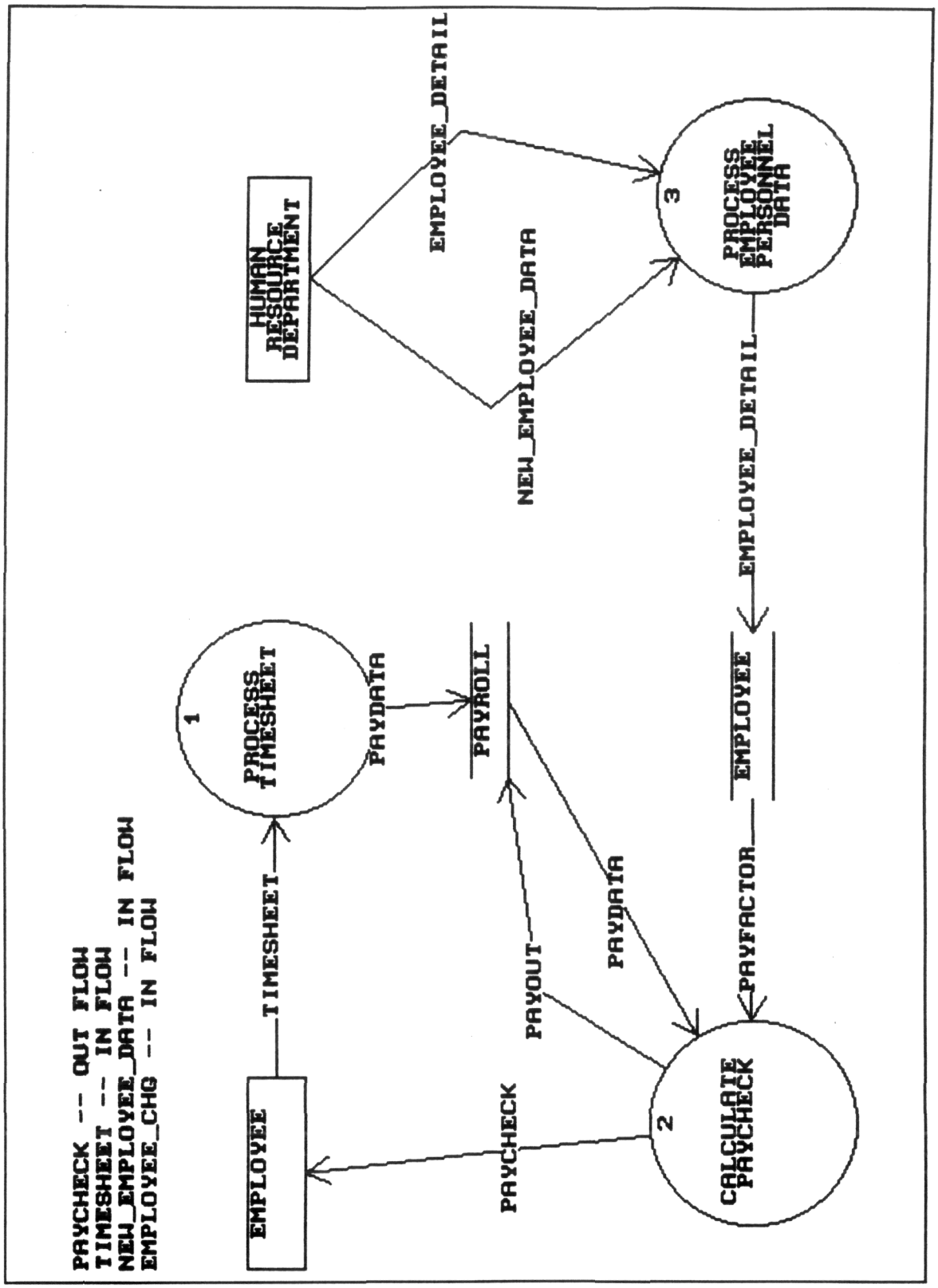

Figure 7 Diagram 0 - Payroll System

Data Dictionary

Data Stores (Databases) *.dbf

(The databases are already created in the project directory.)

Structure for database : EMPLOYEE.DBF

Field	Field name	Type	Width	Dec
1	EMPLOYEENO	Numeric	4	
2	LASTNAME	Character	25	
3	FIRSTNAME	Character	25	
4	HIRE	Date	8	
5	MARITAL	Character	1	
6	DEPENDENTS	Numeric	2	
7	STREET	Character	50	
8	CITY	Character	50	
9	PROVINCE	Character	2	
10	POSTALCODE	Character	6	
11	COUNTRY	Character	2	
12	TELEPHONE	Numeric	10	
13	PAYRATE	Numeric	6	2
14	TYPE	Character	1	
15	POSITION	Character	1	
** Total **			194	

Structure for database : PAYROLL.DBF

Field	Field name	Type	Width	Dec
1	PAYDATE	Date	8	
2	EMPLOYEENO	Numeric	4	
3	HOURS	Numeric	4	
4	SALES	Numeric	8	2
5	CUR_RATE	Numeric	6	2
6	TOTALPAY	Numeric	8	2
7	COMMISSION	Numeric	8	2
8	OVERTIME	Numeric	8	2
9	BASEPAY	Numeric	8	2
** Total **			36	

Data Structures

ADDRESS = STREET + CITY + PROVINCE + POSTALCODE + COUNTRY.

Data Flows

TIMESHEET = EMPLOYEENO + LASTNAME + FIRSTNAME + SALES + HOURS.

PAYDATA = EMPLOYEENO + SALES + HOURS.

PAYOUT = EMPLOYEENO + TOTALPAY + OVERTIME + COMMISSION + PAYRATE.

PAY CHECK = EMPLOYEENO + LASTNAME + FIRSTNAME + ADDRESS + PAYRATE + HOURS + SALES + BASEPAY + OVERTIME + COMMISSION + TOTALPAY + PAYDATE.

PAYFACTOR = EMPLOYEENO + TYPE + POSITION + PAYRATE + LASTNAME + FIRSTNAME + ADDRESS

NEW_EMPLOYEE_DATA = EMPLOYEENO + LASTNAME + FIRSTNAME + ADDRESS + (TELEPHONE) + PAYRATE + TYPE + POSITION + HIRE + (MARITAL) + (DEPENDENTS).

EMPLOYEE_DETAIL = EMPLOYEENO + (LASTNAME) + (FIRSTNAME) + (ADDRESS) + (TELEPHONE) + (PAYRATE) + (TYPE) + (POSITION) + (HIRE) + (MARITAL) + (DEPENDENTS) + (EMP_DEL).

Process Documentation

Process Two
Process Name: Calculate Pay check

Purpose: Process end of the period paychecks..

Inputs: PAYFACTOR, PAYDATA Outputs: PAY CHECK

Process Definition:

1. First Calculate:

   BASE PAY as a function of HOURS & PAY RATE
   (Max. of 40 hours a week)

   COMMISSION as a function of EMPLOYEE POSITION, EMPLOYEE TYPE, & SALES. (Note: -- Only non-management sales employees get a commission -- 5% of sales)

   OVERTIME PAY as a function of EMPLOYEE POSITION, EMPLOYEE TYPE, & HOURS. (Note: -- Only non-management office employees can get overtime at regular pay rate if their hours are above 40 hours a week.)

2. Second Calculate:

   TOTAL PAY as a function of BASE PAY, COMMISSION, & OVERTIME PAY.

Process One
Process Name: Process timesheet

Purpose: Input employee data from timesheets

Inputs: TIMESHEET
Outputs: PAYDATA

Process Three
Process Name: Process employee personnel data

Purpose: Create a record and input data for a new employee.
Modify a record of an existing employee.
Delete a record of an employee leaving the company.

Inputs: NEW_EMPLOYEE_DATA, EMPLOYEE_CHG
Outputs: EMPLOYEE_DATA

Data Elements

Element Name	Type(1)	Len	Dec	Database	Brief Description
CITY	C	50	0	EMPLOYEE.DBF	
COUNTRY	C	2	0	EMPLOYEE.DBF	
CUR_RATE	N	5	2	PAYROLL.DBF	Current pay rate
COMMISSION	N	8	2	PAYROLL.DBF	
DEPENDENTS	N	2	0	EMPLOYEE.DBF	No. of dependents
EMPLOYEENO	N	2	0	PAYROLL.DBF, EMPLOYEE.DBF	
EMP_DEL	1	0		----------	Deletion indicator
FIRSTNAME	C	25	0	EMPLOYEE.DBF	
HIRE	D	8	0	EMPLOYEE.DBF	Hire date
HOURS	N	4	0	PAYROLL.DBF	Period work hours
LASTNAME	C	25	0	EMPLOYEE.DBF	
MARITAL	*C*	*1*	*0*	*EMPLOYEE.DBF*	*Marital status*
OVERTIME	N	8	2	PAYROLL.DBF	Period overtime
PAYDATE	D	8	0	PAYROLL.DBF	Payroll date
PAYRATE	N	6	2	EMPLOYEE.DBF	Dollar/hour pay
POSITION	C	1	0	EMPLOYEE.DBF	Mgt/Regular employee
POSTALCODE	C	6	0	EMPLOYEE.DBF	
PROVINCE	C	2	0	EMPLOYEE.DBF	
SALES	N	8	2	PAYROLL.DBF	Period sales $
STREET	C	50	0	EMPLOYEE.DBF	
TELEPHONE	N	10	0	EMPLOYEE.DBF	
TOTALPAY	N	8	2	PAYROLL.DBF	Total period pay
TYPE	C	1	0	EMPLOYEE.DBF	Sales/Office employee

(1) C = CHARACTER, N=NUMERIC, L=LOGICAL, D=DATE

TEST DATA

The following is the data from employee time sheets. The Pay Date is January 30 1995.

EMPLOYEENO	LASTNAME	HOURS	SALES
1	DAVIDSON	150	0.00
2	DAWSON	170	200.00
3	DAWSON	150	0.00
4	DAY	150	0.00
5	DANGERFIELD	190	0.00
6	DANIELSON	187	1000.00
7	MICHEL	190	0.00
8	MILES	120	0.00
9	MESSENGER	180	0.00
10	SCHEBEL	150	1000.00
11	SCHELL	160	0.00
12	SCHENK	150	0.00
13	TRENHOLM	120	0.00
14	TRENT	180	0.00
15	TRENTINI	190	0.00
16	TRATCH	110	800.00
17	TROLLIP	175	1000.00
18	TROT	160	0.00
19	TROUSIL	100	1293.00
20	VAN DYK	100	900.00
21	VAN EDEN	100	765.00
22	VAILE	101	1000.00
23	UPTON	80	1500.00
24	URBAN	90	1000.00
25	UYEDA	99	999.00
26	VACZY	80	100.00
27	VALENTINE	110	1400.00
28	VALGARDSON	90	1450.00
29	VALGARDSON	100	50.00
30	VANDENBERG	120	2000.00
31	VANDENBERG	100	0.00

Hotel Registration Prototype System

Description of a simple hotel registration system

When the guests registers into this hotel, the system initially records the guest name and registration data such as check-in date and time. It also changes the status of the room to occupied. When the guest wants to check-out of the hotel, the system should calculate the total charge which is based on the number of nights, the season (high or low), the room style, and the appropriate discount (ie. business customers get a 20% discount). Finally, a bill is printed for the guest.

Data Dictionary

Data Stores (Dabases) *.dbf

(The databases are already created in the project directory.)

Structure for database : GUEST.DBF

Field	Field name	Type	Width	Dec	Start	End
1	REGISTERNO	Numeric	4		1	4
2	LASTNAME	Character	25		5	29
3	FIRSTNAME	Character	25		30	54
4	STREET	Character	50		55	104
5	CITY	Character	50		105	154
6	PROVINCE	Character	2		155	156
7	POSTALCODE	Character	6		157	162
8	COUNTRY	Character	2		163	164
9	TELEPHONE	Numeric	10		165	174

Structure for database : ROOM.DBF

Field	Field name	Type	Width	Dec	Start	End
1	ROOMNO	Numeric	3		1	3
2	ROOM_STYLE	Character	1		4	4
3	OCCUPIED	Logical	1		5	5
4	SINGLEBED	Numeric	1		6	6
5	DOUBLEBED	Numeric	1		7	7
6	QUEENBED	Numeric	1		8	8
7	KINGBED	Numeric	1		9	9

Structure for database : RATE.DBF

Field	Field name	Type	Width	Dec	Start	End
1	YEAR	Numeric	4		1	4
2	HIGH_HONEY	Numeric	6	2	5	10
3	HIGH_SUITE	Numeric	6	2	11	16
4	HIGH_REG	Numeric	6	2	17	22
5	LOW_HONEY	Numeric	6	2	23	28
6	LOW_SUITE	Numeric	6	2	29	34
7	LOW_REG	Numeric	6	2	35	40

Structure for database : REGISTER.DBF

Field	Field name	Type	Width	Dec	Start	End
1	REGISTERNO	Numeric	4		1	4
2	CHECK_DATE	Date	8		5	12
3	CHECK_TIME	Numeric	4		13	16
4	GUEST_TYPE	Character	1		17	17
5	ROOMNO	Numeric	3		18	20
6	OUT_DATE	Date	8		21	28
7	OUT_TIME	Numeric	4		29	32
8	TOTALDISC	Numeric	7	2	33	39
9	TOTALBILL	Numeric	7	2	40	46

Data Structures

ADDRESS = STREET + CITY + PROVINCE + POSTALCODE + COUNTRY.
NAME = LASTNAME + FIRSTNAME

Data Flows

CHECK_IN = NAME + ADDRESS + ROOM_STYLE_PREFERENCE + GUEST_TYPE.

GUEST_DETAIL = NAME + ADDRESS.

CHECK_IN INFO = NAME + ROOMNO + CHECK_DATE + CHECK_TIME + GUEST_TYPE.

YEAR_RATES = YEAR + HIGH_HONEY + HIGH_SUITE + HIGH_REG + LOW_HONEY + LOW_SUITE + LOW_REG .

ROOM_INFO = ROOMNO + ROOM_STYLE + OCCUPIED.

ROOM_STATUS = ROOMNO + OCCUPIED.

CHECK_OUT_INFO = REGISTERNO + OUT_DATE + OUT_TIME + TOTALDISC + TOTALBILL.

CHECK_OUT = NAME.

BILL = REGISTERNO + NAME + ADDRESS + ROOMNO + ROOM_STYLE + CHECK_DATE + CHECK_TIME + OUT_DATE + OUT_TIME + GUEST_TYPE + TOTALDISC + TOTALBILL.

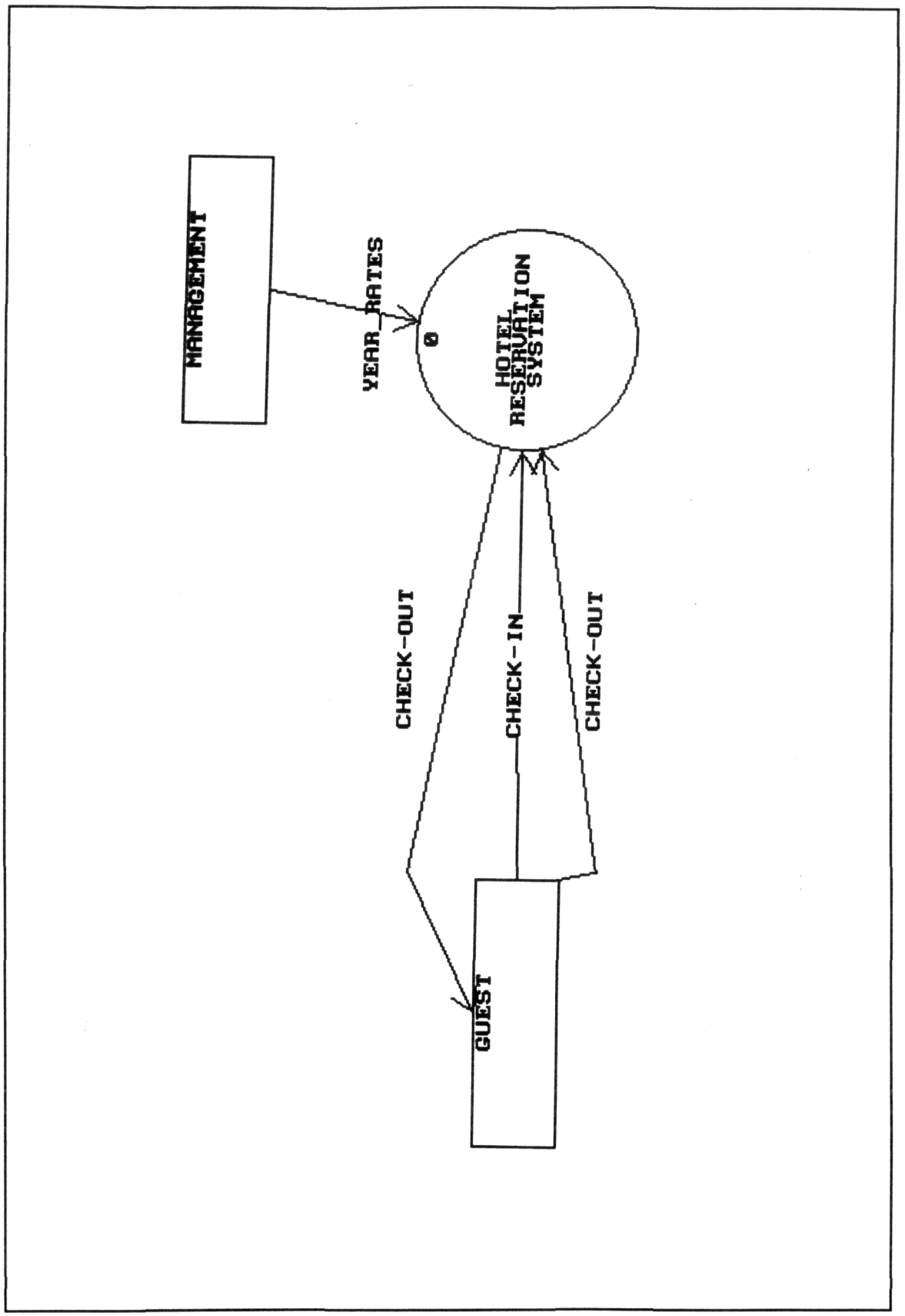

Figure 8 Context Diagram - Hotel reservation system

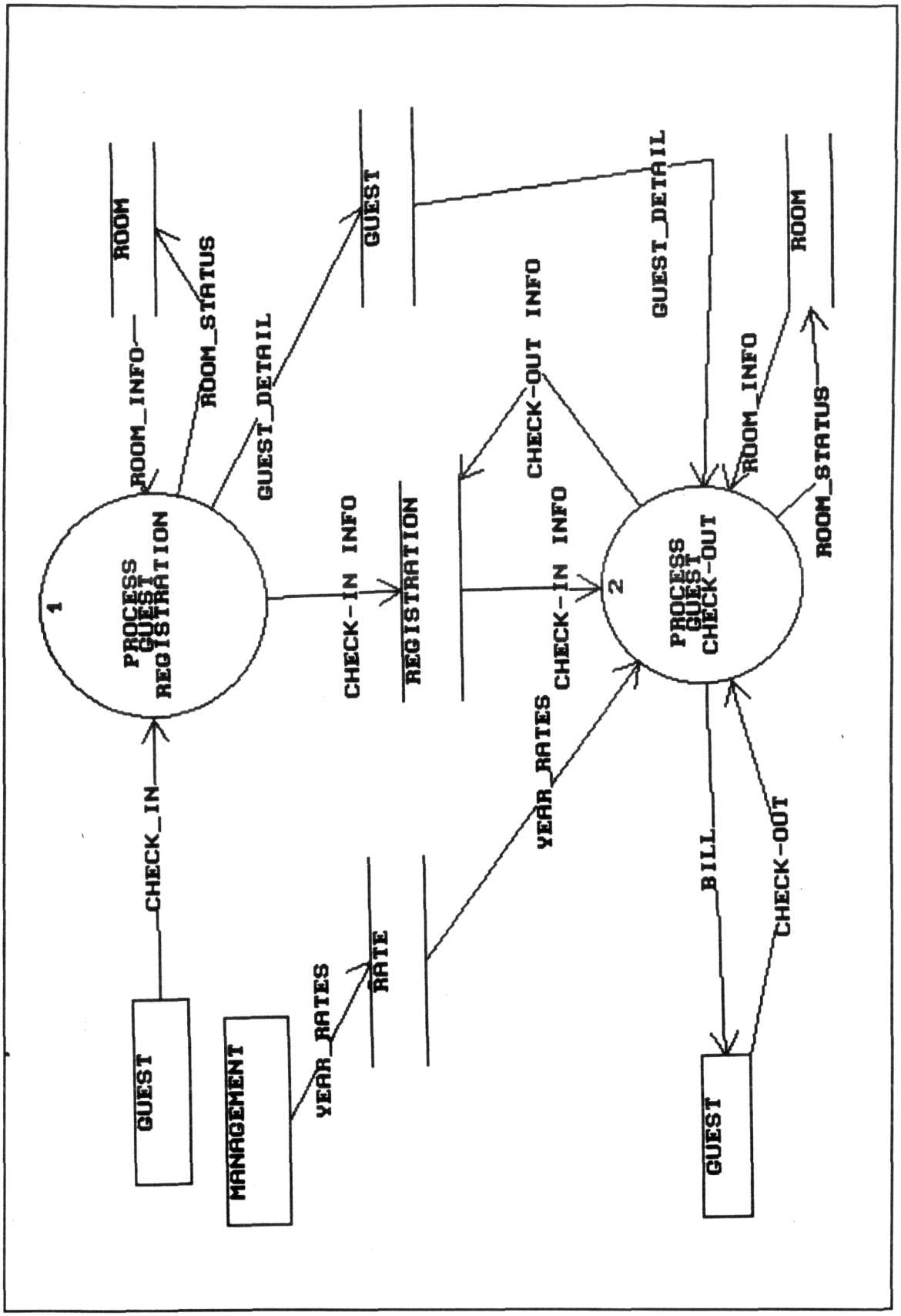

Figure 9 Diagram 0 - Hotel Reservation System

Process Documentation

Process One Process Guest Registration

Purpose
- Accept guest personal information and room preference
- Search for appropriate room
- Change room number status to occupied.
- Input guest information.
- Input registration information.

Inputs CHECK-IN, ROOM_INFO

Outputs GUEST_DETAIL,CHECK-IN INFO, ROOM STATUS

Process Two Process Guest Check-out

Purpose Calculate Guest Total Bill.

Process Definition **Important to do in the following order!!!!!
After the guest has requested to check-out -->

1. Change roomstatus to unoccupied.

2a). Calculate discount based on guest type.

Guest Type	Percent Discount
BUSINESS (B)	25%
EMPLOYEE (E)	25%
REGULAR (R)	0
PATIENT (P)	100%

Note: The company has a policy of allowing relatives of patients to temporary stay for free at the hotel. This is arranged through the hospital administration.

2b). Calculate roomrate.

The roomrate is dependent on the hotel season and the room style. The hotel season is high from the beginning of may to the end of august. The hotel season is low from the beginning of september to the end of april. The rates for each season depending of room style are stored each year in the rate file. For example:

YEAR	1994	
HIGH_HONEY	(High season, honeymoon suite)	$250
HIGH_SUITE	(High season, suite)	$200
HIGH_REG	(High season, regular room)	$100

3. Calculate Total Bill
It is a function of the discount.
Other factors in calculation: days & roomrate

Data Elements

Field Name	Type	Len	Dec	Database	
CHECK_DATE	D	8	0	REGISTER.DBF	Check-in date
CHECK_TIME	N	4	0	REGISTER.DBF	Check-in time
CITY	C	50	0	GUEST.DBF	
COUNTRY	C	2	0	GUEST.DBF	
DOUBLEBED	N	1	0	ROOM.DBF	Number of double beds
FIRSTNAME	C	25	0	GUEST.DBF	
GUEST_TYPE	C	1	0	REGISTER.DBF	Guest type (B)usiness,(E)mployee,or(R)egular
HIGH_HONEY	N	6	2	RATE.DBF	Rate highseason - honeymoon suite
HIGH_REG	N	6	2	RATE.DBF	Rate for regular room in high season.
HIGH_SUITE	N	6	2	RATE.DBF	Rate for suite in high season.
KINGBED	N	1	0	ROOM.DBF	Number of kingbeds.
LASTNAME	C	25	0	GUEST.DBF	
LOW_HONEY	N	6	2	RATE.DBF	Rate for honeymoon suite in low season.
LOW_REG	N	6	2	RATE.DBF	Rate for regular room in low season.
LOW_SUITE	N	6	2	RATE.DBF	Rate for suite in low season.
OCCUPIED	L	1	0	ROOM.DBF	Whether room is occupied.
OUT_DATE	D	8	0	REGISTER.DBF	Date guest checks out.
OUT_TIME	N	4	0	REGISTER.DBF	Time guest checks out.
POSTALCODE	C	6	0	GUEST.DBF	
PROVINCE	C	2	0	GUEST.DBF	
QUEENBED	N	1	0	ROOM.DBF	Number of queen beds.
REGISTERNO	N	4	0	GUEST.DBF REGISTER.DBF	Unique registration number.
ROOMNO	N	3	0	REGISTER.DBF ROOM.DBF	
ROOM_STYLE	C	1	0	ROOM.DBF	Room style (H)oneymoon,(S)uite,(R)egular
ROOM_STYLE_PREFERENCE		1		-----	The room style the guest requests.
SINGLEBED	N	1	0	ROOM.DBF	Number of single beds
STREET	C	50	0	GUEST.DBF	
TELEPHONE	N	10	0	GUEST.DBF	
TOTALBILL	N	7	2	REGISTER.DBF	Guest total bill
TOTALDISC	N	7	2	REGISTER.DBF	Guest total discount
YEAR	N	4	0	RATE.DBF	The year the roomrates apply.