G E N E R A L I Z E D E D I T I N G

(A DESIGN STUDY OF A COBOL ORIENTATED EDIT PROGRAM GENERATOR)

by Lance W Pearson B Sc

July 1978

A Thesis presented in partial fulfilment
of the requirements for the degree of
Master of Science in Computer Science
at Massey University

ABSTRACT

All commercial data processing installations include programs to detect errors in input data. There is a high degree of commonality in the editing (i e validating) of such input data throughout the data processing industry.

This thesis defines a generalized editing package which will allow a user to specify the editing requirements for any set of input data. From the specifications a COBOL program will be created to carry out the required operations on the input file.

Included as an introduction to this thesis, is a survey of editing needs, and a discussion on the merits of generalized software.

The thesis emphasizes the methodology of the generation of a specific "tailor-made" editor program.

Key Words: Editing
EPG (Edit Program Generator)
Generalized Software
Program Generator

Computing Review Category: 2.0, 3.50, 4.12, 4.41

ACKNOWLEDGEMENTS

I would like to express my deep appreciation to those people who have helped me throughout the writing of this thesis. It has been important to me to have their informative ideas and encouraging enthusiasm for this project.

Sincere thanks are due to my supervisors, Professor G Tate and Mr P J Melhuish, for all the able assistance and guidance they have given me. I particularly wish to thank Peter Melhuish for his many comments and suggestions offered after the careful reading of this thesis, even when it has been at his inconvenience. His personal friendship and advice I am grateful for.

I would like to take this opportunity to thank my parents for their consistent generosity and the optimism they have expressed to me throughout my years of study at Massey. The motivation this has been is appreciated and will not be forgotten.

Finally, I would like to convey thanks to my typist for her kind and competent assistance given during the preparation of this dissertation. She has made many sacrifices to share with me the headaches and nightmares that form part of the trauma associated with this type of excercise. ... A very special 'thank you' Karen.

Lance W Pearson
Massey University
Palmerston North
NEW ZEALAND

July 1978

TABLE OF CONTENTS

LIST OF FIGURES AND TABLES

1. INTRODUCTION - THE DP ENVIRONMENT

The processing of data, within a computer-assisted business typically follows a standardized set of procedures.

This is true, irrespective of the size of the organization or the number of systems requiring computer processing or the complexity of each procedural system.

Four areas common to data processing systems would be:

(1) Manual and clerical procedures

(2) Data capture, preparation, and input

(3) Computer processing and computer programming

(4) Report production and distribution

This is shown diagramatically in figure 1.1.

The diversity of manual and clerical procedures is great because the many industries have different services and objectives. This is not true for the other three common areas. They follow standard procedures which can be easily described.

In general the data processed is converted to some machine readable form. It is entered via an input device to be edited for errors and processed, then organized as a file structure held on some high capacity storage device. From it, readable documents conveying the input information in an organized and usable form, can be produced. The data to be processed can be very different in appearance. For example, the detail on items sold by a retailer barely resembles the detail on graded meat and its shipping destinations. Both are collections of transaction records. They can be processed in essentially the same way. For this reason the numerous models of computers used possess functional resemblances.

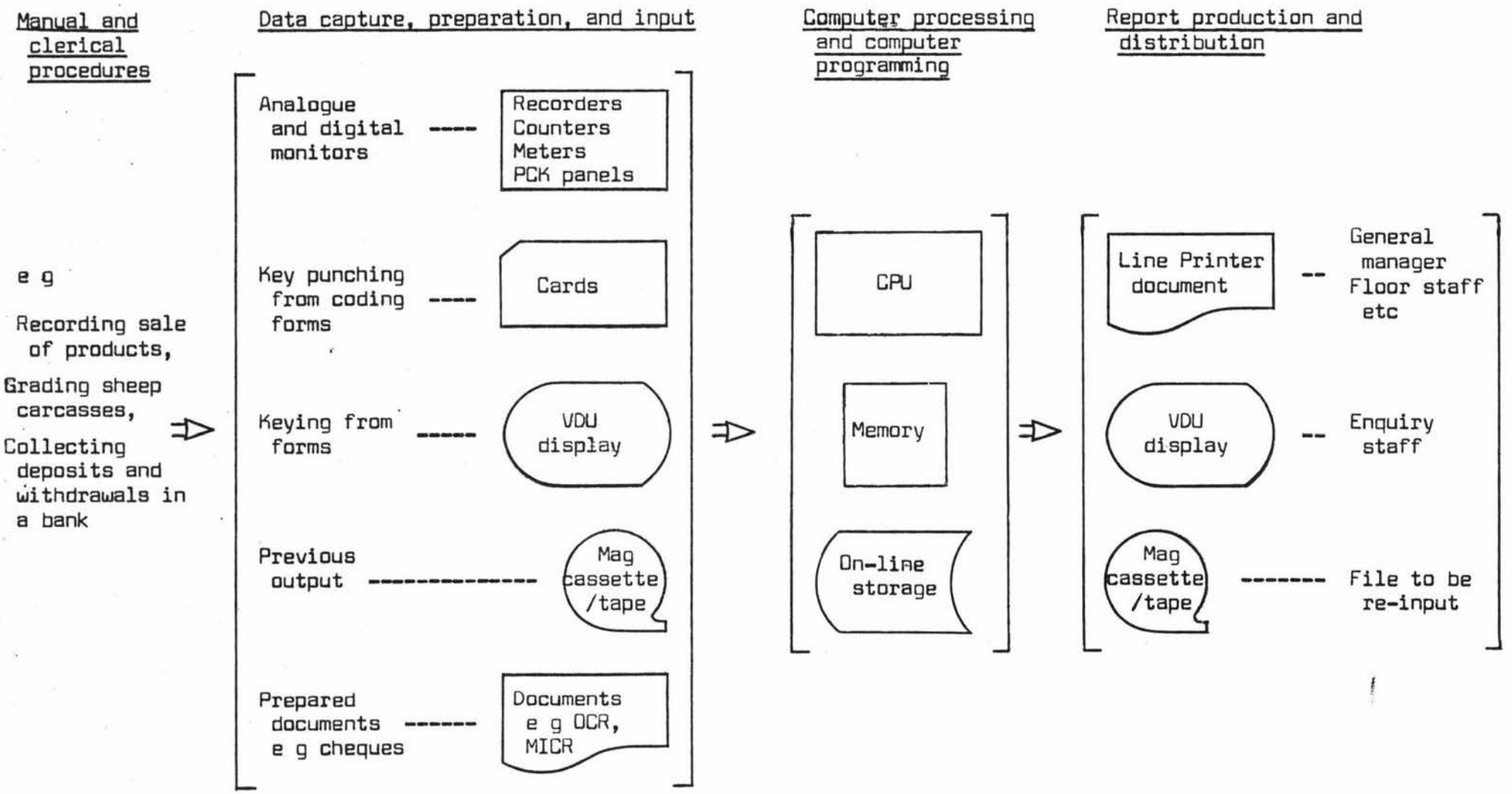

Fig 1.1 Common organization of data processing systems.

In data processing, user software aids exist to solve the many common problems needing computational processing. Program generators are such an example of generalized software aids. Error editing of input data is also an example of a common computational problem in this case shared by all computer users. This thesis is the specification for an edit program generator (EPG). Background discussion, on generalized software, the editing problem, and the alternatives to the solution of editing, precedes this specification.

2. GENERALIZED SOFTWARE

2.1 Use of Generalized Software

Within the computer industry much attention has been given to the development of generalized software. These are programs written to perform functions common to DP applications. They are generalized to allow modifications to suit the unique problems and needs of a wide range of possible users. In contrast, custom-built software is designed to solve the particular problems of one user. The latter is almost always less expensive to write and develop. However, generalized software is becoming more and more competitive in performance, efficiency and economy with tailor-made products because it is shared by more than one user.

In New Zealand most computer data processing is done on small scale machines for relatively small companies. There are a number of reasons why generalized software is attractive to them.

- Software development is time consuming (high level languages just aren't high level enough).
- Newly written in-house software is likely to be unreliable for some time.
- Generalized software to suit requirements is often available, from manufacturers, software houses, bureaux or other users.
- Software development principally requires labour. Labour is:-
  - expensive (more than the hardware costs)
  - unpredictable - length of time expected for programming
    - length of employment so that the software is adequately supported
  - often unavailable.
- Good documentation is often provided.

2.2 Examples of Generalized Software

Problem solution by the use of generalized software has evolved to increasing levels of generality as listed below:

(1) System software

(These appear at the lowest level of problem solution although they perform complex functions).

e g - Operating systems,
- Assemblers,
- Compilers,
- Message Control Systems,
- Management Information Systems,
- Data Base Management Systems.

(2) High level language extensions

e g - COBOL Sort and Report Writer,
- Indexed sequential access routines.

(3) Languages for specialized DP functions

e g - Reporters (such as RPG),
- On-line inquiry,
- Graphics packages,
- Text editing,
- Maths/Statistics functions.

(4) Applications programs

(These appear at the highest level of problem solution. They are usually independent enough so that data does not need pre or post processing).

e g - Payroll and labour cost packages,
- Inventory control.

2.3 Forms of Generalized Software

There are several forms of generalized software:

- code producing software,
- modified compilers allowing language extensions,
- macro- and pre-processors,
- parameter driven packages,
- program generator routines that will produce a program based on the specifications for a particular type of problem.

Each form has its advantages and disadvantages which will influence selection. Factors include:

- availability of software or programming resources,
- suitability for running on available hardware,
- degree of expertise required to use them,
- software support provided,
- type of application.

2.3.1 Code producing software

Like a compiler, code producing software generates unique machine-code from the user's specifications. This code is efficient but is machine dependent and therefore the generalized software can only be used on a subset of machines. Computer manufacturers are the primary source for such systems e g Burroughs DMS-II [1].

2.3.2 Extended compiler software, macro- and pre-processors

As with code producing generalized software, efficient code is generated but portability is a problem. Extended compilers do not promote standardization within the industry although some features gain enough acceptance to eventually become language features e g SORT verb in COBOL and more recently Indexed Sequential file

access routines, also in COBOL. These extensions never achieve total acceptance since some machines do not have adequate hardware to support the new features. In other cases the software enhancements required are prohibitively uneconomic.

2.3.3 Parameter driven packages

This software is completely generalized. As a result some degree of efficiency is lost. Each time a job is performed, all parameters have to be interpreted. As the job runs many data moves are necessary to communicate with the generalized procedures used by the package. The code for functions not being used occupies valuable space in the machine's memory. As a result the package is often too large for small to medium systems. The portability of parameter driven packages depends on the portability of the language used to write them.

2.3.4 Program generators

Program generators are the most flexible form of generalized software. They are as portable as the host language they are written in. The symbol-code produced by them can be just as portable, depending on the purpose this generalized software has. The programs they generate can be as compact and efficient as their custom-built equivalents. Their reliability and ease of use is immediate. Another popular advantage is that the symbol-code they generate is readily modified to suit user exceptions.

They have some disadvantages also. To produce a new program or modify an existing one the program generator must be run to generate symbol-code which then must be compiled into machine-code. This two-task operation is undesirable, particularly when a separate, larger machine must be used if the program generator is too large. This

also necessitates a third task of converting and transferring machine-code back to the application computer. Note, that this same problem may also be present with code producing software and extended compiler software.

Many RPG (Report Program Generator) implementations are program generators. Others are interpreters (i e Parameter driven packages). In New Zealand SPL, Systems and Programs (N Z) Ltd, have produced a program generator called "PROGENI" [2]. It is a collection of MACROS to assist COBOL programming.