Software Development for Microcomputer Data Processing Systems

P. A. Dearnley
School of Computing Studies and Accountancy, University of East Anglia, Norwich NR4 7TJ, UK

The paper examines some of the problems of developing software for microcomputers used in commercial data processing. The availability and use of software tools is discussed. An example is given showing the fabrication of a system from existing software subsystems in preference to either a packaged program or a suite of tailor-made programs.

INTRODUCTION

The aspect of software development, for microcomputer data processing systems, considered here is that of the usage of software tools to generate commercial programs. Issues relating to obtaining the correct design and managing the production process are not discussed.

One view of the microcomputer field is of rapid development of hardware and a constant stream of new peripheral devices accompanied by very limited software. The software is provided using machine code or a simple version of traditional interpretive BASIC. This view is supported by scanning the popular computing press and trade magazines. Many of the texts on microcomputer software concentrate on unstructured BASIC and on machine code. This situation is thought to compare poorly with the software tools available on mainframes. Given the right level of entry into the microcomputer field the reality is somewhat different; the software provision is also developing rapidly with new tools being introduced and existing tools being transported from mainframes or minis.

PROBLEMS

The development of microcomputer software has all the problems identified for mainframe software (timeliness, correctness, goodness of fit to user needs, etc., but two problems are exacerbated by the low cost of hardware. First, the comparative cost of the software component of a system relative to the hardware and other costs, puts additional pressure on the already present need to reduce software costs. Second, the reduced total system costs, as compared with mainframes and traditional small business computers, have increased the breadth of applications which are now considered beneficial.

The growth of the share of total cost taken by software is not a new phenomenon, but current analysis suggests that the problems identified for mainframes by, for example, Boehm in the 1970s will continue in the 1980s and 1990s for micros.[1,2] Figure 1 is derived from the forecasts of Wise *et al.* It shows both the optimistic and pessimistic predictions for software costs, along with a likely cost. The author contends that the approach of programming each application in traditional BASIC or machine code is likely to lead to the pessimistic cost

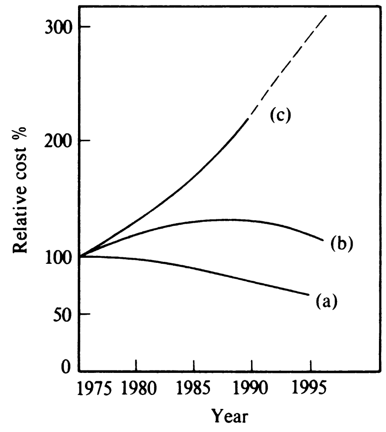


Figure 1. Relative cost of data processing software (a) optimistic forecast; (b) likely forecast if new programming techniques used; (c) pessimistic forecast.

situation. Introducing better tools and methods might lead to the likely cost, and some revolution in software production is needed to bring about the optimistic cost situation.

The increased breadth of applications militates against the packaged program solution. Often the packages are not available or not well supported.[3] Where packages in the correct functional area are available they may be rejected because they have the wrong external interfaces to the user. Many of the packages offered for micros are priced too low for any support to be provided. The areas which are covered tend to be restricted to routine production of accounting ledgers and to clerical functions such as word processing. Thus much of the software will have to be generated from scratch.

LEVEL OF SYSTEM

The bottom end of the hardware price range for microcomputers is dominated by systems with a simple operating system and a simple interpretive BASIC held in read-only memory. The program size is limited by the main store available to hold the source statements and files of small size can be held on cassette tape. The usual BASIC offered is unstructured and follows the traditional line oriented style of programming. Since source texts are usually held in main store, neat layout and comments are

CCC–0010–4620/82/0025–0253 $02.00

discouraged. Whilst discs and printers can be added, the program interface is often awkward. Further up-market are the systems based on the Z80 and CP/M.[4,5] Such systems have at least floppy disc storage and a reasonable set of tools. The former type of system has the price advantage when comparing raw hardware, but prices of CP/M based systems are continuing to drop. Comparing hardware costs alone creates a false impression. If the true software development and maintenance costs are considered, then the cost advantage of the small systems may be illusory. The author considers that the latter type of system represents the minimal level of microcomputer for developing data processing software.

SOFTWARE ENVIRONMENT

If the 'Z80 and CP/M' type of microcomputer is to be used along with its choice of languages and range of tools what is the difference in software environment between it and the traditional mainframe? First, the pure batch processing type of working is virtually unknown on microcomputers; the usual types of system are, following Blackman's classification, enquiry, data capture and file update.[6] Second, the scale of operation relative to the available computer power is such that methods which would be unacceptable on a large multi-access mainframe may be acceptable on the microcomputer. Finally, the typical small business microcomputer runs as a single user machine located in the user department, this in turn removes many of the problems inherent in larger data processing systems; there is no need for teleprocessing software, if a database system is used then there is no need to cater for concurrent usage.

CP/M is now the *de facto* standard for simple application development and execution on single user microcomputers.[7] However, there are other operating systems providing a more sophisticated environment such as OASIS[8] or, on larger machines, a multiuser development environment compatible with the UNIX operating system of mini-computers[9] and through it access to various software tools.[10] All these operating systems provide support for a range of tools with which a professional approach to software development can be taken.

SOFTWARE TOOLS

The most obvious tools are language compilers and interpreters. These cover a wide range comparable with a mainframe. Fortran, Cobol, PL/I, Pascal and Algol are all available. There are many dialects of BASIC including some compiled versions supporting structured programming and many language extensions. The range of facilities in a 'modern' BASIC, for example OASIS Basic,[11] are such that the language bears little resemblance to the original Dartmouth BASIC.

For the programmer working alone, or, for the librarian in a team structure[12] there are excellent text processors which will both edit source programs and 'word-process' documentation. These tools assume that the system development will be through the normal process of writing and debugging high-level programs. Some of this work can be saved by the use of subsystems such as database managers callable from the application program, for example MDBS.[13] Other parts of a system can be produced with program generators. These generators are used to produce code for insertion in a conventional program, to produce a 'custom-built' complete program or to configure an existing complete program to the user requirements.

Source code generator

The assumption is that the developer is a normal programmer with the usual skill base. The source code generator allows him/her to produce more accurate code faster than the equivalent hand coding. The code produced is then included in the source text of the application program, which is tested in the normal way. An example of such a generator is the Micro Focus FORMS program.[14] This allows the programmer to layout the data to be displayed on and accepted from a VDU terminal on the terminal itself in a dialogue with FORMS. The generator then produces and files Cobol code required to implement the VDU display. The code generated is also used to accept data from the terminal and check the type of each item entered. This code is included in the application programs using the Cobol COPY verb. This tool relieves the programmer of what is otherwise a tedious and error prone part of the coding. Examination of the source text of a suite of programs using this generator produced the following statistics. There were seven programs containing approximately 5800 statements. The programs used 23 screens to collect data or menu options. The screen handling code was generated and tested in 15 hours; the volume of this code was 1,700 statements. Thus, part of the application was produced at over 100 lines per hour. This accounts for approximately 30% of the total coding. Since the suite was examined after it had been written, the statistics are not part of a properly controlled experiment; it is possible that the hand-written equivalent screen handling code would be more compact and thus the figures given above are over-estimates of the rate of production and the proportion of the job generated. However, the figures give some indication of the value of such a generator.

Parametrized generators

This style of program generator assumes that the programs to be generated fall into one of a number of general classes (e.g. data entry, file update, reporting). The generator produces a set of parameters which direct the generalized programs to perform the specific processing required by the user. The parameters are held with the users files and are accessed by each generalized application module at execution time. Additional 'support' files may be used to define the order of execution of modules, the text of menus, etc. A good example of this style of generator is the Configurable Business System.[15] This generator provides modules for file creation and update, for reporting and for sorting. A number of utilities for file definition, file tidy-up and menu definition are used to create and maintain the system. The parameters used to steer the application modules are held as header records in the users files.

Complete program generators

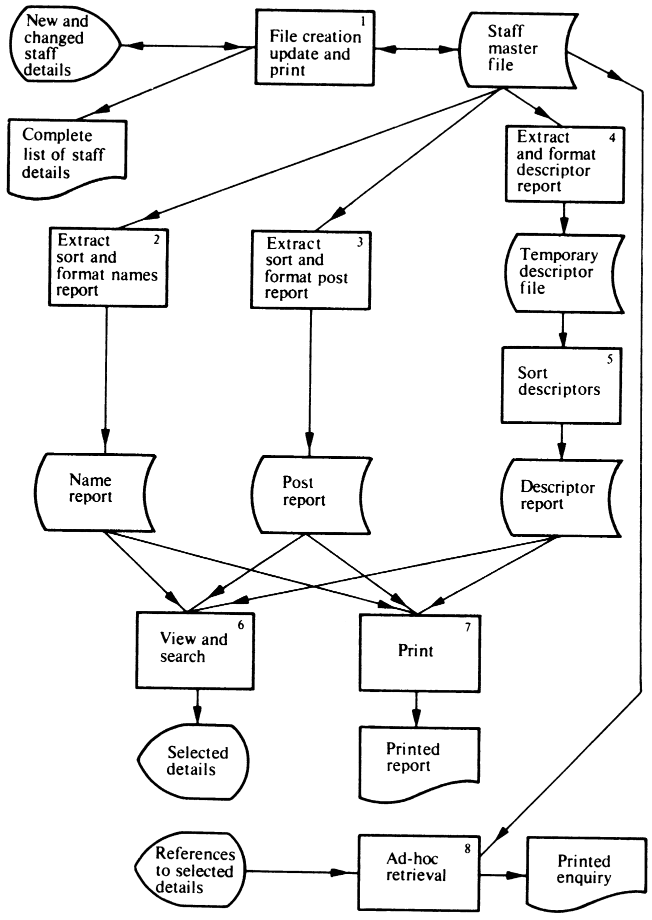
A number of complete program generators are available for the 'normal' data processing tasks. These usually work in the same fashion as source code generators except that they produce complete programs which are then compiled in the normal way, or, they produce an internal representation of the program which is then interpreted at execution time. Typical tasks are data entry and report generation. Examples of data entry program generators are FORMs 2[16] and Datastar;[17] whilst the Selector IV[18] generator provides data entry, file processing and reporting. The example given below makes use of such a generator. The vendors of these generators suggest that the end-user can produce the programs required directly and, whilst it is true that they do not require programming skills in the conventional high-level languages, such users would require a good knowledge of computer data processing. The author views these tools as aids for the analyst-programmer.

EXAMPLE

The following example serves to show how various tools can be combined to make almost all of an application. The example is based on a real system built for a CP/M based microcomputer; the number of attributes has been reduced to simplify the description.

The user required an information system to keep track of a large number of professional staff holding various posts in various locations. The data to be recorded concerning a person included a reference code, name, current post held, geographic location, personal details (data of birth, nationality, etc.) and a free format career resumé. The resumé contained various interesting descriptors which were highlighted in the original manual system. The staff data had to be stored, updated, amended and printed. The preferred user interface for this operation was a form displayed on the VDU which could be completed in a similar manner to the manual system, then corrected or printed as required. From the stored data, three reports were required in addition to the printing of complete entries. The first report was to be a sorted list of names along with current post and reference number. The second was similar, but giving post followed by name and sorted on post. Finally, a list was required of all the 'interesting' descriptors from the career resumés along with the names and reference numbers in which they occurred. The three lists were to be printed for subsequent photo-reduction and distribution. They were also to be available for perusal on the VDU both in a serial fashion, and, for the descriptors, subject to a context search.

The package solutions suggested involved either personnel subsystems which would run as a by-product of a payroll system, or, a contract staff package which would involve the unwanted aspects of customers and charge-out rates. Neither system would allow the use of a form close to that familiar to the user. The alternative of programming the entire system from scratch was roughly estimated at several months for a contract programmer. The system as built is shown in Fig. 2. Program 1 was implemented using the DATASTAR



New and changed staff details
1 File creation update and print
Staff master file
Complete list of staff details
Extract 4 and format descriptor report
Extract 2 sort and format names report
Extract 3 sort and format post report
Temporary descriptor file
5 Sort descriptors
Name report
Post report
Descriptor report
6 View and search
7 Print
Selected details
Printed report
References to selected details
8 Ad-hoc retrieval
Printed enquiry

Figure 2. Example system: data flow.

data entry and retrieval program generator. This package allows a form to be designed as the basis of the user dialogue. Data is entered, corrected, deleted and viewed by reference to the form. New data and corrections are validated. The type, range and format of fields can be checked. Copies of individual entries and of all entries can be printed in the same layout as the VDU form. The same software is used for program 8 where individual entries are called up by reference number following perusal of the lists produced by programs 6 and 7.

Programs 2 and 3 use a sort package which allows both sorting and data selection/exclusion. The selection feature is used to choose just the desired fields for the output list and to format them. The exclusion feature is used to skip data entries marked as deleted. Thus the sort package selects active records, selects desired fields, formats the data and sorts onto an output file. The lists are printed by program 7 which is the CP/M list utility program. The lists can be viewed using a word processor program (program 6). The program allows the lists to be viewed a screen-full at a time, and allows the viewer to move backwards or forwards through the file. The context searches are provided by the scan feature of the word processor. No actual word processing in the sense of editing or justification is done! The word processor is used as a convenient tool for viewing formatted data files. Program 4 has to scan the free format career resumé and output one record per descriptor found. The descriptors are delimited by special characters but can be in any position in the lines making up the resumé. To extract this data a special program was written in BASIC. The program needed 150 statements using the file handling and string processing features of CBASIC. The output from this program is then sorted (program 5) using the sort package.

The design, development and testing of the entire system took 9.5 days. This included familiarization with the data entry program generator which had not been used previously. The system is not ideal since the different software tools have different conventions for their user dialogues. However, it has been easy to modify; the original form layout has been changed once. This, with essential changes to the sort parameters and the BASIC program, took one half day of editing and testing.

With the exception of one small BASIC program the entire system is made using a program generator, a sort package, a word processor and an operating system utility program. These programs are used as sub-assemblies and are combined to engineer the system. The resulting system is usable, easy to change and cheap. The components from which it is built are well tested adding to the reliability of the whole system. A 'better' hand coded system could have maintained secondary indices in place of extracting and sorting. It could have produced a single uniform user interface instead of the data entry dialogue and the word processor command conventions. It is unlikely that the cost, timescale and reliability of this 'pure' approach would have been justifiable compared with the 'engineering' approach.

CONCLUSION

The basic conclusion of the paper is that, whilst some systems will need substantial 'original programming', many could be built using generators and existing subsystems if the correct engineering approach is taken to development projects. Building applications in this fashion may result in systems which are not ideal compared with hand-coding but the price of construction and maintenance will be lower. Where extensive programming is needed there is no excuse for assuming that microcomputer projects should or must be undertaken in the 'traditional line oriented BASIC plus machine code' style given the wide range of tools currently available.

Note. To make this paper reasonably specific and practical various commercial software products are mentioned. It should not be assumed that the lists of products are comprehensive nor should it be assumed that the products are in some sense 'best buys'; they have been used to explain and illustrate.

REFERENCES

1. B. W. Boehm, Software Engineering, *IEEE Trans. Comput.*, **C-25** (No. 12) (1976).
2. K. D. Wise, K. Chen and R. E. Yokely, *Microcomputers: A Technology Forecast and Assessment to the Year 2000.* Wiley, New York (1981).
3. P. Hammersley, The Impact of Microcomputer Systems on Commercial Data Processing, *Comput. J.*, **24** (No. 1), pp. 14–16 (1981).
4. Digital research, *CP/M User's Guide*, Digital Research, Pacific Grove, California (1979).
5. D. Powys-Lybbe, CP/M operating System—The Software Bus, *Microprocessing and Microsystems*, **5** (No. 3) (1981).
6. M. Blackman, The Design of Real Time Applications, Wiley, New York (1975).
7. M. Healy, CP/M—the De Facto Standard. *Computer Age* (April 1980).
8. Phase One Systems, *OASIS System Reference Manual*, Phase One Systems, Oakland, California (1980).
9. K. L. Thompson and D. M. Ritchie, The UNIX Time-sharing System, *Communications of the ACM*, **19** (No. 7) (1974).
10. B. W. Kernighan and P. J. Plauger, *Software Tools*, Addison-Wesley, Reading, Massachusetts (1976).
11. Phase One Systems, BASIC Language Reference Manual. Phase One Systems, Oakland, California (1980).
12. IBM, *Improved Programming Technologies—An Overview.* CG20-1850 (1975).
13. Micro Data Base Systems, *MDBS User's Manual*, Lafayette, Indiana (1979).
14. Micro Focus Ltd, *CIS COBOL Operating Guide*, Version 3, Chap. 8, Micro Focus, London (1979).
15. Dynamic Microprocessor Associates (1979).
16. Micro Focus Ltd, *FORMS 2 Utility Manual.* Micro Focus, London (1979).
17. Micro-Pro, *DATASTAR Reference Manual* (1980).
18. Micro-Ap, *Selector IV Manual.* Micro-Ap, Dublin, California (1980).

Received August 1981