Behavior Research Methods, Instruments, & Computers
1987, 19 (2), 210-214

Incorporating good programs into the larger laboratory context

HOWARD L. KAPLAN
Addiction Research Foundation, Toronto, Ontario, Canada

The Scheduled Measurement System (SMS) is a collection of related programs for administering performance and rating tasks to human subjects. It was designed to provide a common task scheduling and data management environment for a diversity of measures. SMS has greatly reduced the effort required to analyze and archive data after experiments, but it has been less successful in reducing some of the effort required to implement experiments. While programming is not difficult, documentation and user training remain the most primitive parts of the system.

About 18 months ago I wrote a paper entitled "When do professional psychologists need professional programmers' tools?" (Kaplan, 1985). In that paper, I discussed the Scheduled Measurement System (SMS), a collection of related programs for human performance testing, describing the various modules of the collection and the program development environment in which they were created. I still believe that the development tools of SMS work well, but recently I have begun to recognize what does not work so well in SMS. In this paper, I want to discuss SMS, show how its design facilitates program development and maintenance, and share my understanding of why that kind of facilitation is not sufficient. I will conclude by suggesting that "programming" is too limited a concept to describe what ought to happen when we implement computer-controlled experiments.

To understand the successes and failures of SMS, it is helpful to review the stages of computer-controlled experiments large enough to require collaboration among five or six individuals: The *implementation stage* includes selection of independent manipulations and dependent measures, design of the assignment of treatments to subjects, regulatory approval, creation or modification of software, documentation of procedures, training of staff, and running of pilot tests for staff practice. The *execution stage* includes subject recruitment and qualification, subject training, administration of treatments, collection of data, editing of data as necessary, backing up of files, and production of such daily reports as are consistent with blinding of staff to experimental conditions. The *analysis stage* includes reprinting of reports with experimental conditions identified, production of summary descriptive and inferential statistics, archiving of data and of the programs used to produce them, and writing of reports.

The author's mailing address is: Computer Services Department, Addiction Research Foundation, 33 Russell St., Toronto, Ontario M5S 2S1, Canada.

AN OVERVIEW OF SMS

SMS is a computerized data collection and management system for studies in behavioral toxicology, that is, the effects of drugs on human attitudes, physiological signs, and performance. For many of the drugs investigated at the Addiction Research Foundation, the changes are measurable over the course of hours (as drugs are absorbed into and cleared from the bloodstream), although the experiments themselves may last for days or weeks. At scheduled times during the day, a battery of nonsimultaneous measures is performed in a specified sequence, and the results are recorded on the computer. Measures of interest to us include heart rate, blood pressure, estimated blood alcohol concentration (as measured in breath), actual concentration of alcohol and other drugs in the blood, manual tracking, recall memory, hand tremor, postural stability, spectral analysis of spontaneous electroencephalogram (EEG), endogenous evoked potentials, and various self-rating tasks. The data can come either directly from sensors or from the subjects' or experimenters' responses to computer-presented questions. The computer can simultaneously manage the data for up to four subjects' overlapping test schedules, although only one subject's data is actually being collected at any instant.

DATA MANAGEMENT IN SMS

Several factors contributed to the design of SMS. One major factor was a history of unsuccessful management of the data of previous studies. There was a recurring problem of data's being collected, stored, and then retrieved only with great difficulty because of the lack of proper documentation or, when the documentation was complete, because of the idiosyncratic organization of the data. In the area of non-real-time studies (generally questionnaires about drug history and the current effectiveness of therapy), we were solving the data management

problem by developing an integrated system for editing data, updating files, and producing audit trails of the edits and updates. At the time SMS was being designed, this data management package was being enhanced to include automated generation of data dictionaries and of proofreading and updating programs derived from those dictionaries. SMS was designed to be compatible with the formats of these data management tools, not because the standards were especially efficient, but primarily because the standards were in existence. Among other virtues, this compatibility would allow for the eventual integration of SMS-collected real-time data with long-term questionnaire data, should any studies require such integration.

All measures running under SMS incorporate data dictionary information as part of their executable code. In order to store data, an SMS measure first defines the name, range, number of alphanumeric characters or decimal digits, and precision of each item to be stored, and then makes use of system-level routines to format, store, and retrieve data records. All of SMS is written in Turbo Pascal (Borland International, 1985). Let us consider some of the code that would be used to present a 5-response multiple-choice rating scale. We can then see how the data definitions propagate through SMS:

```
{Specify the question and the answers}
const nreplies=5;
const qtext: string[40]=
   'How tired are you at the moment?';
const replies: array[1..nreplies] of
        array[1..12] of char=
   ('1Not at all  ','2A bit       ',
   '3Somewhat  ','4Quite a bit',
   '5Very much  ');
{...other intervening code...}
{Add the question to the data dictionary}
defineoutput(12 {definition serial number},
        'TIRED',qtext,integertype,
        1 {replicate},
        1,0,{digits and decimal places}
        1,nreplies {valid response range});
definereport(alwaysprint,
        {graph with} fixedscale);
definelabels(nreplies,
        11 {characters per label},
        replies);
```

The first block of code defines text strings for the question and the possible answers; the second block uses those text strings to create some data dictionary entries. Later in the program, the same text strings are used to actually present the question and possible replies on the video monitor. If the subject reports making an error in answering a question, the experimenter conducting the session can edit the response array at the end of the day; the question text is redisplayed as part of an editing screen, and the range of valid responses is used to control the editing process. The short question name "TIRED" is repeated in the row names of data tables and in the legend of graphs on the end-of-day printout, where the ordinate is the range of valid replies. At the end of the experiment, SMS writes SPSS-X code in which "TIRED" becomes a variable name, the complete question becomes a variable label, and the possible replies become value labels. Both the data records and the SPSS-X program are then transmitted to a VAX for statistical analysis.

Pascal provides a mechanism for constants (such as the number of answers per question) to be expressed by name rather than by value as they recur in the program source. This reduces the amount of work necessary to revise the program if a change is made, such as using a 7-point rather than 5-point rating scale. In SMS, such a change not only propagates easily through the program that presents questions and collects replies, it also propagates through the tools dependent upon the data dictionary: the printed copy of the data dictionary, the editing procedure, the daily printouts, and the SPSS-X program. In summary, SMS provides a set of tools in which decisions about data item names, ranges, and storage need be specified by the programmer only once; after that, the same decisions reappear in different contexts. In the experiment, the efficiencies so provided are divided between the execution (editing and daily individual-subject reports) and the analysis (final individual-subject reports and statistical analysis).

MEASURE DEVELOPMENT IN SMS

SMS is structured as a resident set of commonly used shared utilities and a set of overlay areas. At any one time, no more than 56K of SMS and its overlays are in the computer's memory, although a typical experiment's total code might occupy around 300K of disk storage. The remainder of the computer's memory is used for the operating system (PC-DOS), for control and character-shape tables, for the Turbo Pascal run-time library, for data areas (including large waveform buffers for EEG work), and occasionally for separately compiled FFT routines.

An overlay is simply program code that gets read into memory when it is needed and overwritten by other code when it is no longer needed. Overlays were originally developed for computers with very limited memory. It was more efficient to reload part of memory with a new program than to write intermediate results to disk or tape, reload all of memory with a new program, and resume processing the intermediate results. The idea of using such overlays in the psychology laboratory goes back at least as far as Creelman's first PSYCLE system (Creelman, 1971), running in 4K of 12-bit memory on a paper-tape based, 1967-vintage PDP-8/S. With memory prices rapidly declining, why should overlays be used today?

One reason to use overlays is to reduce program development time. According to the Turbo Pascal reference manual, all of the alternate overlays that might occupy the same memory area must be compiled as consecutive procedures or functions of the same base program. Were this in fact the case, then the entire body of SMS code would need to be recompiled whenever any part of it were

changed. In fact, because Turbo Pascal is a one-pass compiler, all of the references from an overlay to code or data outside of itself refer to addresses that are already known before the overlay is encountered in the source code. Therefore, it is possible to divide SMS into a set of separate programs, each of which substitutes dummy code for the preceding overlays and for the following unresolved forward references. Once that is done, a fairly simple routine to modify the overlay loading process allows separately compiled overlays to be loaded as if they were compiled as part of the base program.

In SMS, the base program consists of shared subroutines to handle common tasks, such as formatting data, validating keyboard entry, reading the light pen position, accessing disk files, and presenting menus. All of the other SMS routines fall into one of three classes: those that handle analog devices, those that take specific measurements, and those that manage the flow of procedures and data.

SMS can use almost any reasonable analog-to-digital converter system, but some are limited to a sampling rate of 60 Hz per channel. In order to synchronize all other events with the screen refresh rate (because the screen is sometimes used to present stimuli), the time base for scheduling events is the screen vertical retrace interval. Some analog hardware, such as the Tecmar Lab Master board, has no separate memory for digitized data and no DMA access to the main system memory; such hardware can be sampled only once per clock tick, although many channels may be scanned in sequence. Other analog hardware, such as the Data Translation 2801 board, has DMA access, and therefore it can provide faster analog samples, provided that the sampling clock is derived from the vertical retrace signal to maintain synchronization. The Modular Instruments M-100 series of analog instrumentation has internal memory for samples, and, with a similar synchronized clock, it too can store data from sampling faster than 60 Hz providing that the backlog of accumulated data is read into the laboratory computer 60 times per second. Each analog interface board has its own interface routine written as an overlay occupying a maximum of 2K of memory. When first loaded, one part of the routine sets the sampling parameters (data rate, number of channels, number of bits per sample, etc.) and passes the address of the interrupt-handling routine back to the calling program. Subsequently, the interrupt handler is called 60 times per second to pass digitized analog data back to a memory array. SMS saves a 2-sec backlog of this data in memory, and any measurement module can request part or all of the backlog, isolated from most of the details of which kind of hardware provided the samples.

In SMS, a measure is an overlay procedure that generates a vector of related data items. In most cases, these items are dimensions of a single task, such as manual tracking (e.g., in keeping the image of an airplane over the image of a moving road, the dimensions are RMS distance from the road center, time over the road, and peak distance from the road center) or recall memory (the dimensions are the number of words recalled without and with cuing from each of four presentation categories, plus control information, such as which categories were presented and in what order the words were presented). In a few cases, the dimensions are technically unrelated items that are simply convenient to collect together. Each measure has a number of different tasks it can perform on request from the calling program: it can simply define its data dictionary, print a record of which of its options have been selected (such as the difficulty of the tracking task), collect raw data, or rescore data (revise a scale total after some of its items have been corrected in the editing process).

The primary control over the sequence of events in SMS is provided by a set of main menu overlays, so called because they correspond to selections that the operator can make from the main menu presented when SMS is loaded. Some of these selections include Schedule, to describe the times and orders in which measures are to be conducted; Practice, to run measures outside of a schedule for test or demonstration purposes without saving the data on disk; Run, to conduct measures according to the schedule and to build disk data files; Edit, to correct data or to enter values not known at run time (such as the results of drug concentration assays); Display, to print and graph summary data; and Merge, to combine data from different subjects' files into a larger file suitable for uploading to a VAX for statistical analysis.

Because the various functions are programmed as separate overlays, some aspects of program development are fairly efficient. Each compilation of a module involves definitions shared among modules plus code peculiar to that module, and there is no time-consuming linking step to resolve references to other modules. If we accept the usual understanding of the goal of programming, to provide instructions to the computer, then this is a reasonable accomplishment. However, the front-end stage of implementing an experiment is much wider than simply instructing the computer. It also includes instructing the staff who must conduct the experiment, and it is in this area that SMS is not advanced over previous tools.

DOCUMENTATION AND TRAINING IN SMS

Earlier we looked at some sample code to ask a subject to make one of five choices. The text of the question and the number and contents of replies propagated through several SMS procedures. However, when SMS asks the experimenter a question, the nature of the question and the consequences of the answers propagate only through the program module in which the question is asked; they do not propagate to any kind of documentation other than the hard copy of the program source listing. After the programmer writes 100 lines of code to calibrate amplifiers prior to recording EEG, then 100 lines of documentation are needed to tell the operator (who is not expected to understand computer languages) how to understand the calibration procedure and what new questions are likely

to follow from certain previous answers. What is most annoying about this process is that both the computer and the programmer can understand the sequence and the consequences perfectly well from the 100 lines of code: why should the programmer need to translate it back into English?

What clearly is needed is some way to capture documentation information as part of the process of writing a measurement module. I am currently aware of two possible approaches to the outlined documentation problem. Neither has been investigated in great depth, and so far I can only speculate on how well they might work.

The first alternative is to adopt an object-oriented approach, possibly using a different programming language. There is an extensive series of articles about object-oriented languages in the August 1986 issue of *BYTE* (e.g., Pascoe, 1986; Tessler, 1986); I describe them only briefly here. Programmers are accustomed to languages in which the verbs operate upon external objects; conceptually, we tell the computer's adder to combine 2 with 2. In an object-oriented language, we instead pass a command to an object; conceptually, we tell 2 to add 2 to itself. The benefit is that "add" can take on a meaning that is peculiar to the object being addressed. It seems useful to be able to tell a measurement module to document itself, where the exact nature of the documentation would depend on inner details of the module. Something along these lines now happens, where SMS can instruct any module to document its data. This concept might be extended, so that SMS could instruct a module to document its operator interface and its flow of control. Although this is possible in Turbo Pascal, it might work much better in a language where such commands are natural rather than simply possible.

The second alternative is to retain the use of Turbo Pascal as only one of several compilers that would operate on the same program source. Right now, the material appearing in comments is simply for the information of the programmer; it has no other formal use unless the programmer copies some of the text into an instruction manual. It should be possible, however, to write a program that would inspect the source text to look for specific information inside what Pascal considers comments and to use that information to build tables, screens, or other items used in the interaction between the computer and the experimenter. These items would then be available not only to the run-time control system but also to a run-time help facility and an off-line documentation-production system. A related feature would capture results screens, graphics displays, and other examples of actual program execution to disk, also for incorporation into printed documents.

Incorporation of these features might mean that SMS would require more storage, either main memory or disk, than it now does. I worked in the laboratory where Creelman's 4K computer ran many sophisticated, useful experiments, and the experience conditioned me to be conservative in my use of computer resources. Of course, there was no on-line help system and little user friendliness: the computer simply performed a laboratory task in the only way possible. That tradition influenced the development of SMS. Currently, SMS needs less than 256K for most procedures, and usually the run-time modules for any one experiment will fit onto one 360K diskette. When SMS was first designed, larger memory complements and hard disks were more expensive, and there were good economic reasons for limiting the requirements to a minimal configuration. When prices began to decline, I continued to reject suggestions to upgrade the hardware. People with access to 640K of memory or to a hard disk are accustomed to running programs like Lotus 1-2-3, developed by a team of full-time programmers over a year or more. How could one programmer, myself, possibly have the time to fill that much memory with anything useful? Larger computers simply would create unfulfillable expectations about the sophistication and user friendliness of the software. It is only in the last few months that I have realized that having the additional resources would save me time, not because I waste time shoehorning modules into small, fixed-size overlay areas (that is only a minor problem) but because a good, on-line help system would actually reduce the work required to document operating procedures.

"PROFESSIONAL" TOOLS: A RECONSIDERATION

Advertisements in the popular computer magazines often suggest that some compilers are "beginner's" or "toy" products, but that the one being advertised is a "professional" tool. The qualities attributed to the "professional" tools are support for large memory models, separate compilation of procedures (implying a subsequent linking stage), production of optimized code, and integrated debugging environments. To some extent, these attributes are relevant to experiments in psychology. As my ambitions for SMS start to include integrated documentation and guidance, I can see the need to use more memory to hold the programs. I am already using separate compilation and can see its benefits. Optimized code would reduce both the time and the space occupied by programs, and better debugging tools are always welcome. However, most of these features would contribute to the program development stage of SMS that already works reasonably well; support of larger programs would only indirectly aid in the documentation process. Even the most "professional" Pascal compiler is still a third-generation tool, a procedural language. What it produces is compiled code plus auxiliary products (such as symbol tables and memory maps) to help users understand the compiled code. In the commercial world, some tools produce not only code but also data dictionaries, menus, maps of the paths between menus, and similar aids to using the system. It is useful to think of these as systems analysts' tools rather than programmers' tools, because they

directly address the wider environment within which the programs must function. We psychologists need analogous tools.

In summary, I currently perceive SMS as a partial success. It is most successful in collecting and managing data. It is successful in helping me write programs, but programs are only one of the products that ought to be produced from what I write. SMS is successful in helping the operators conduct experiments, but better and more timely documentation would improve their efficiency. Since it is unlikely that a separate technical writer will be added to our staff, it is important that the computer relieve me of some of the work involved in writing text. I see acquiring or building tools to accomplish that as the next stage in the evolution of SMS.

REFERENCES

BORLAND INTERNATIONAL. (1985). *Turbo Pascal Version 3.0 Reference Manual*. Scotts Valley, CA: Author.

CREELMAN, C. D. (1971). Rapid response and flexible experimental control with a small on-line computer: PSYCLE. *Behavior Research Methods & Instrumentation*, **3**, 265-267.

KAPLAN, H. L. (1985). When do professional psychologists need professional programmers' tools? *Behavior Research Methods, Instruments, & Computers*, **17**, 546-550.

PASCOE, G. A. (1986). Elements of object-oriented programming. *BYTE*, **11**(8), 139-144.

TESSLER, L. (1986). Programming experiences. *BYTE*, **11**(8), 196-206.