MOVING A HEALTH DATA SYSTEM FROM A RESEARCH TO A BUSINESS ENVIRONMENT
Michael C. Rein and Lisa A. Rein - Hein Associates

ABSTRACT
Health research groups and businesses in the health care industry have an understandably different set of goals and methods. This is especially apparent when comparing the systems development styles and overall computing needs of the two groups. When converting a data system that was born in a research project into a usable tool for a corporation, moving the data from the university computer to the company machine is only one step among many. The priorities of the researcher are not those of the business person. This difference creates tasks such as paring away hundreds of intermediate research files, enhancing the code and data structures to provide new output or interfaces (reports, export files, and so on), and, of course, documentation.

One client of ours, a medical systems firm, needed just this conversion when they acquired a data system of intensive care unit records from a university research group. This paper describes the tasks that were required to accomplish the conversion - a conversion that crossed hardware platforms, SAS software releases, and most significantly, bridged the gap between the academic and the corporate worlds.

SPECIFICATIONS
When the work was first presented to us, it seemed like a straightforward task of converting the programs and datasets. The university computer was a VAX minicomputer running SAS version 5.18. The company was running SAS version 6.03 on a Sun SparcStation 1+ under Unix. A target date of sixty to ninety days was established to move all the necessary datasets and put the system in some kind of order. There were about 1000 SAS datasets and 4000-5000 SAS programs, logs and list files to be moved, some as much as seven years old. The storage requirements for all the VAX files totalled about five hundred megabytes. The available space on the SUN workstation was around two hundred megabytes. Even considering the amount of data, two to three months seemed like a reasonable amount of time.

PLANNING THE WORK
The first step in accomplishing the work was planning the discrete tasks that would be required. We divided the process into the following steps:

1. Identify which datasets were needed and divide the files into those which should be kept on-line and which could be kept on tape.
2. Convert the desired SAS datasets into SAS transport format and store them on a transportable medium. Store the programs, logs, and lists on a transportable medium.
3. Transfer the SAS transport files and text files to a medium which was readable by the SUN workstation.
4. Update the programs and organize the database to make it operational on the SUN under SAS 6.03.
5. Validate the results of the data conversion and all programming and data structure changes.

IDENTIFY THE DATASETS
Determining which datasets were required was a formidable task. The 1000 SAS datasets were stored in four VAX directories, and the naming conventions used were not immediately apparent to the uninitiated. This fact pointed out the first difference between university researchers and business system personnel in terms of a computing environment. The university staff who performed the research programming consisted of two people who worked closely together for five years. They knew which data was stored under what name because they had a five year learning curve; the important fact is that they did not need data management conventions. We had allotted one to two weeks for the identification of the necessary files and it was a milestone that was important because the later steps of conversion and streamlining would be time consuming and had more potential pitfalls.
Working with the company staff and the researchers, we identified four datasets which were central to our database, comprising the edited versions of the collected patient ICU stay information.

1. Admission records consisting of the diagnostic information, chronic health evaluation, and demographic data.

2. Daily physiology records storing vital signs and blood gas measurements for each day of an ICU stay.

3. Treatment records detailing the specific treatments provided to the patient on each day of an ICU stay.

4. Outcome information including ICU and hospital survival and admission/discharge dates.

From these datasets, it was possible to identify a set of programs which acted on the core data to calculate patient scores and predictions. These programs, along with their associated log and list files, and the SAS datasets which were created by them, completed the required files for conversion.

The surprising fact is that the number of files to convert was now about 100 SAS datasets and 100 program files. With the log and list files and some lookup files included, the total number of files had been pared down to around three hundred, and many of these were only required for historical reference and could be stored on an archive tape. This reduction was accomplished without losing any information and the surgery performed actually began to clarify the database for us.

PREPARING THE DATA TO BE MOVED

SAS software provides a transport facility for the purpose of sending datasets from one platform to another. The steps are simply to create a transport file on a medium readable by both platforms and then convert the transport file, using SAS on the target system, into a SAS dataset in the desired location.

Using version 5.16 of SAS on the university VAX, PROC XCOPY was used to write the transport files:

```
filename tran '/dev/rst0' ;
libname target 'temporary directory' ;
proc cimport file=tran libname=larget ;
run ;
```

The tape containing the text files was taken to a company which specializes in converting data formats. For a reasonable price, they converted the files to the 'tar' format and stored them on a quarter inch archive tape from which files were selectively loaded onto our machine. Now all the necessary files were available at our site.

ORGANIZING THE DATA SYSTEM

To this point, the project had been standard data processing. The work had held few surprises and was actually ahead of schedule. All that remained was to clean the system up, update the code with the new library references, run the programs, and validate the results.

As we began to investigate the programs and the datasets upon which they acted, however, certain problems
began to surface. First, the code was not up to 'usual'
standards concerning coding and data structure; by the
standards of a system professional, it was a little messy.
The gap between research and business was there again.

Here is where we have to explain these differences
mentioned earlier. Research programmers have a goal: to
discover things. The 'things' can be trends, correlations,
cause and effect relationships, etc. Researchers tend, as in
case, to work alone or with one assistant and their
purpose in using the computer is to facilitate testing the
hypotheses. In this case, the researchers had a collection
of data on patients in intensive care units across the
country. The goal was to find relationships between
physiology measurements and several patient outcome
variables, including mortality, length of stay, amount of
active treatment required, and the likelihood of discharge
alive the next day. Contrast these purposes with the
computing process was a common technique with the
research staff and was intentional, not the result of sloppy
practices. Much of the analysis work performed on the
medical data was centered on determining the predictive
value of physiology and diagnostic fields when analyzing
outcome. Thus, carrying source variables, such as
admission diagnosis, into the files containing calculated
physiology severity scores allowed quicker and easier
multi-variable validation such as two-way frequency
analyses. Compare this with the more formal business
database design that we implemented which attempted to
keep a given variable in only one file. Our emphasis was
on a consistent structure, reliability, and also, storage
efficiency. The more formal structure was more easily
documented and was simpler to reproduce and test when
creating project-based data systems for clients. The storage
efficiency was an added bonus which reduced the
company's expenditures for hardware storage devices. The
trade-off for these benefits is that a merge is required if a
source variable is to be analyzed alongside a generated
variable, such as a physiology score, which is computed in
a later step and stored in a different file.

Similarly, the methods used by the two sites for
representing an array of eighty disease categories are
different for reasons of application. The research
programmers store eighty three-byte numeric variables,
having values of 0 or 1. As they are mutually exclusive
flags, only one of the eighty is set to 1 for any ICU
admission. Fifty different regression and prediction
programming programs use this array of flags as independent
variables. The expanded storage used at the university
greatly simplifies the programs which use the disease
categories since they can be used directly in the
regressions. In contrast, we stored one numeric variable
which served as an index pointing to the array of disease
flags. The flag names were stored in an ordered list in a
utility macro which passed a macro variable back to the
calling environment. Since the macro did not change, the
reliability that comes with modular storage of reusable code
was guaranteed. It meant that for each regression program,
the index variable had to be 'expanded' into the eighty flag
array by setting to 1 the element which corresponded to the
index value. We were setting up stable routines which
would be maintained by a professional programming staff
and the added code was not a high price to pay to reduce
the amount of storage needed and to eliminate repetitive
lists of variables in several programs. The code variable
actually eased adhoc analysis programs since one variable
is much easier to work with than eighty.
Beyond the gap caused by the requirements of each site, differences also exist in the amount of exposure to various coding techniques that the programmers had received. One of the determinations performed in the data system is the calculation of the neurological severity score. It is determined by the response levels for eye opening, motor reflexes, and verbal response which are used in the assignment of the Glasgow Coma Score. At the university, if-then-else logic is used to assign the severity score. There are about sixty valid combinations of the three response variables which are cross-indexed and assigned into one of eight severity scores. This translates to about 50 lines of if-then statements including the handling of missing values. When implementing this calculation at the company site, we created a table, GLASGOW.SSD which stored one observation for each combination of the three scores and stored the associated severity score for each combination. Using this dataset, a format was created using the CNTLIN option of PROC FORMAT to map the response levels to a score (see figure 2) which was used as the method for recoding in the physiology program. This difference can be attributed to lack of familiarity with such techniques and serves to underscore the benefits of working with large technical staffs. With more eyes and brains examining a problem or needed function, it is inevitable that better solutions will be proposed. Most business environments afford the programmer with opportunities to interact with more diverse elements of the computer industry, in general, and the SAS software community specifically.

The last example in figure 1 addresses the process of changing and testing programs. SAS software is a tool which supports a rapid coding and debugging cycle. No separate and laborious compiling and linking is required when new code is developed or an existing routine is modified. We believe that this ease of development subverts the testing and change management steps of development and maintenance. When the university team changed programs, the usual method of testing was to change the output dataset(s) to a similarly named file, sometimes a numbered sequence of files (e.g. PHYSDAY PHYSDAY2 ...PHYSDAY6). For a one- or two-person programming staff, this is not a big problem since communication of the current version is limited to one phone call or yelling to the next office down the hall. We feel that this style still makes the process vulnerable to human error, especially to the possibility of using the wrong dataset in such a sequence. From the time the first datasets were loaded onto the SUN machine, we differentiated what was test, or not validated, code and data and what was considered production. Our purpose was to create a data system which could be maintained by any experienced SAS programmer and which would require a minimal learning curve. The easiest way to provide this was to clearly delineate the test and production environments, storing them in separate directories which started as identical copies. It seems almost a cliché to emphasize the testing and validation of code changes, but these are easy steps to bypass when the language is as dense and flexible as SAS software.

**Figure 2**

Storage of Glasgow Coma Scale components and creation of the SAS format used to assign the neurology severity score:

Dataset: GLASGOW.SSD
Variables: EYEOPEN, MOTOR, VERBAL ($1) and NEURSCOR 3.

Format Creation:

LIBNAME PHYSFMT 'SAS format directory';
LIBNAME PHYSTBL 'SAS data directory';

DATA LOOKUP;
  LENGTH START $ 3 LABEL $ 2;
  RETAIN FMTNAME 'SNEUMAP';
  SET PHYSTBL.GLASGOW
  END=ALL_DONE;
  START = EYEOPEN || MOTOR || VERBAL;
  LABEL = PUT(NEURSCOR,2.) ;
  OUTPUT;
  IF ALL_DONE THEN
    DO;
    START = 'OTHER';
    LABEL = '0';
    OUTPUT;
    STOP;
    END;
RUn;

PROC FORMAT LIBRARY=PHYSFMT
  CNTLIN=LOOKUP;
RUN;

Format Application:

NEURSCOR = INPUT(PUT(EYEOPEN || MOTOR || VERBAL, NEUMAP.), 2.) ;
When a working data system is inherited by a business from a less structured environment, it is difficult to devote the resources needed to upgrade it according to a set of demanding standards. It becomes easier when the process is juxtaposed with most engineering processes in most industries. Even within a single pharmaceutical company or one manufacturer of jet engines, the research and development scientists take a far different route than the eventual process engineers when producing a new drug or a modified propulsion system. They must differentiate because rapid trial and error is the trademark of the successful researcher, and consistency that of the production engineer. There is no difference in the engineering process which creates new software.

After so much discussion of the utility of formal structure when considered in the framework of a programmer's goals, it is necessary to state some near absolutes. We have found that good programming practices, such as avoiding repetitive code, using efficient data storage and manipulation techniques, and generally working towards accurate and reliable results, always reward the developer with a shorter, less frantic development and maintenance cycle. The researchers described above are professionals who attempted to perform their task in the best way possible. They fulfilled their goal of discovery. We achieved our ends by providing a data system which was ordered, reliable and fit for use as a model for future (unknown) data collection. It is important to stress that each party learned from the other. It is not always better to impose structure when ease of use is paramount. Conversely, the speed and flexibility of adhoc work is often enhanced by working with an easily understood database and a tested toolbox of structured programming techniques. SAS software provides such tools as user-defined formats for table-lookup, a macro language used to isolate repetitive code and to pass parameters that add flexibility, and a dataset header as a built-in data dictionary. SAS users in all computing environments will benefit by learning and using these tools effectively.

CONCLUSION

Perhaps the best way to show the benefits of our conversion method is the fact that halfway through our validation process a consortium of 28 hospitals (with a total of 39 ICUs) purchased quarterly performance reports (read: You need these WHEN??!). An input/edit system was written to read in raw data supplied by the hospital and to allow our client's staff nurses to make corrections. Previous data collection procedures were enhanced due to new time constraints. We developed semi-automated procedures to be used by staff nurses for data handling, tracking, and communication with the hospitals. We coordinated and tested changes to the data collection software (provided by another consultant). The bulk of the consortium's customization needs involved reporting by individual ICU and by hospital (for those with multiple ICUs). Additional reports compared the consortium members with national, regional, and similar-type hospital norms. All of the deliverables were generated quarterly, and also required was the ability to combine them for a set of yearly reports. We also made some patient record tracking enhancements; hospitals needed to account for every patient, including records that were disqualified for a variety of reasons. In the end, the reports went out on time and correct, and a very flexible, easily modified system that consumed a total of 45 megabytes was in place to handle any number of new customers and new requirements with a minimum of programming time. Without the planning and effort involved in the conversion, we could easily be sitting there today asking "which program runs next?".