INTRODUCTION

In the development of large-scale systems such as a set of programs which interpret a survey, programmers and analysts need to develop ways of testing code and verifying output. This becomes important when a large system is being developed by several people. SAS provides features which can be used to test code and verify output, for example, the FREQUENCY, PDS and SOURCE procedures. These SAS procedures can be incorporated into packages called subsystems which operate independently of the main system and provide information about the system to the programmer, analyst, and management. The subsystems are written in SAS; however, the large system can be written in SAS, PL/I, FORTRAN, or COBOL.

The life cycle of a set of software can be divided into three phases as shown in Figure 1: the system design and development phase, which incorporates the writing and preliminary testing of code; the system execution testing phase, which involves using the source code with test data to check syntax; and the system results verification phase, during which the "live" data is used with the source code and actual results are checked for accuracy. Four subsystems have been developed by ORI, Inc. for the purpose of testing code and verifying output in the three phases of the software life cycle (Figure 2). The first subsystem maintains source code in partitioned data sets through the use of PROC SOURCE and PROC PDS. The
second subsystem verifies successful execution of the system and deletes unused OS files allocated by the system. The third subsystem presents results to be hand checked by the programmer or analyst after the system has updated the data base. The fourth subsystem extracts information from the SAS log during system execution to provide management with statistics such as CPU time used in each step and space allocated.

In addition to aiding the programmer, analyst and management, these subsystems can enable non-SAS users to perform generalized system testing. The operating system can be used interactively to prompt the user for parameters and information necessary to run a subsystem and to automatically execute the SAS programs with the user's parameters. This enables the same generic program to be run on many different components of the data base, with minimal input and SAS knowledge required of the user.

**PROGRAMMING EXAMPLES**

**Subsystem 1 - Maintain Source Code**

SAS programs that reside as members of a partitioned data set (PDS) can be maintained using this subsystem. During the design and development phase of a large-scale system, the programmer needs at different times to:

a) Delete members from a PDS.

b) Rename members. (especially for testing purposes)

c) Obtain source listings of selected members from the PDS.

---

**PROGRAMMER**

**ANALYST**

**MANAGEMENT**

**PRODUCTION**

**SYSTEM FOR SURVEY ANALYSIS**

**FIGURE 2.** The Use of Subsystems to Provide Information about System to Programmers, Analysts, and Management

**FIGURE 3.** Simple Output of Subsystem 1.
**Subsystem 2 - Verify Successful Execution**

The actual execution of the system is tested for successful completion and for usage of all files allocated by the system. This subsystem scratches unused files and informs the user whether execution was successful by printing a special, enlarged message of job status.

The condition from a previous job step is examined in the SAS program. All allocated files are examined to determine if data were written to them. PROC EXPLODE is used to produce the enlarged "SUCCESS" or "FAILED" message (See Figure 4).

**Subsystem 3 - Generate Output for Manual Review**

Results of system execution can be verified by the use of frequencies on selected variables to determine whether the variables to be updated by the system have been updated.

A SAS macro was written to acquire user-requested variables from the data base and execute PROC FREQ for the selected variables. In order to avoid voluminous output of numeric variables, the data were recoded using PROC FORMAT into a valid range, zero, and missing categories. For an example of the output, see Figure 5.

**FIGURE 4. Sample Output of Subsystem 2.**

**FIGURE 5. Sample Output of Subsystem 3.**
**GENERAL INFORMATION**

<table>
<thead>
<tr>
<th>OBS FILE</th>
<th>NAME CODE</th>
<th>OBSW</th>
<th>YARW</th>
<th>CPW</th>
<th>ORDER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 SYSIN WORK.A DATA SET TYPE</td>
<td>33</td>
<td>5</td>
<td>0.06</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>2 SYSIN PRINT PROCEDURE TYPE</td>
<td>.</td>
<td>.03</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 SYSIN SORT PROCEDURE TYPE</td>
<td>.</td>
<td>.06</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 SYSIN PRINT PROCEDURE TYPE</td>
<td>.</td>
<td>.03</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5 SYSIN PRINT PROCEDURE TYPE</td>
<td>.</td>
<td>.03</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 SYSIN WORK.B DATA SET TYPE 28</td>
<td>3</td>
<td>.06</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7 SYSIN SORT PROCEDURE TYPE</td>
<td>.</td>
<td>.03</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8 SYSIN PRINT PROCEDURE TYPE</td>
<td>.</td>
<td>.03</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**LEGEND**

- **FILEW** = input file
- **NAME** = data set name or procedure name
- **OBSW** = the number of observations
- **VARN** = the number of variables
- **CPUW** = CPU time used
- **ORDER** = the number of data sets generated within a data step minus one

The controlling program that prompts the user for input parameters also accepts "conditions" under which certain observations would not be included in the frequencies.

**Subsystem 4- SAS Log Analyzer**

Another SAS program can be used to analyze the SAS log (See Figure 6). (This provides statistical information concerning the SAS run flow.)

This SAS program is executed as a separate job step following the main SAS program. The SAS log is read, and the CPU times (for each Data Step and Proc Step), data set names and number of observations and variables are saved in a SAS data set. Efficiency concerns can be addressed using this data, and observation and variable counts can be tracked.

**CONCLUSION**

Evaluation of large scale systems can be achieved using generalized validation subsystems written in SAS. In this way, the components of a large system can be tested using a single set of tools, and programmers do not have to spend time writing programs to test programs. These subsystems can often be run by technicians and reviewed by analysts. If these tools are used throughout the design, development and implementation of a system, the results can serve as early warning indicators for faulty algorithms or misinterpreted design specifications.


The author can be contacted at:
ORI, Inc.
Software Applications and Training Program Office
7910 Woodmont Ave., Suite 1405
Bethesda, MD 20814
(301) 856-5276