A Program Walk-through for a SAS Application
Robert G. Hall
ARC Professional Services Group

INTRODUCTION

This discussion concerns SAS program walk-throughs. I will briefly address walk-throughs in general; then I will present some code to a panel of expert reviewers. They will examine and discuss the code. The discussion is unrehearsed. The intent is to learn what the panelists think about as they proceed through a walk-through—what they are looking for, what they are not looking at, and how do they draw conclusions from what they see.

Walk-throughs have been a part of software development mythology for more than twenty years. A walk-through consists of one or more people reviewing code or program documentation (or both) that has been written by someone else. Often times walk-throughs represent an in-depth technical review of code. They can occur at any time during the life cycle of a software development project. Typically, walk-throughs check for completeness, consistency, and the suitability of the product for requirements.

Walk-throughs serve multiple functions. Primarily, however, this is a technique for detecting errors. This use of the walk-through mirrors the process individuals utilize when they try to debug code inherited from other programmers. First, you determine what the code is suppose to do, then you step through the code (PROC by PROC, DATA step by DATA step) trying to identify problems. The walk-through simplifies this process by having the author present the code, reducing the amount of time required to determine what the author was trying to do. It also moves the process earlier in the development cycle. This is important since we all know that the earlier you can detect problems, the less expensive it is to fix the problems.

Walk-throughs have additional functions, however. They act as milestones; a point in time in which developers can say that their code has reached some level of acceptability. This is especially important in larger projects. In larger projects there is a tendency for programmers and reviewers to endlessly cycle through a draft-review-draft cycle. By focusing the review into a single session, this cycle can be reduced considerably.

Walk-throughs are also used to increase project communication. A walk-through enables programmers of other parts of the project to become aware of all the issues confronting the project team. Presumably, it will also help the developers to conform to similar coding strategies throughout the project.

Walk-throughs have traditionally dealt with code. They represent one of the few opportunities individuals, other than task leaders, have to see real-life code that other people have generated. Walk-throughs can also focus, however, on documentation. In particular, requirements documents and design documents are excellent candidates for walk-throughs—especially in those rare cases where they exist and they are prepared before the code is actually written. Less frequently, test cases can be the focus of a walk-through. In this discussion, however, we will be focusing only on code walk-throughs.

CLASSICAL APPROACH

This discussion is based on the parts of the book Software Engineering Concepts by Richard Fairley. He discusses walk-throughs as comprised of the reviewer and three to five reviewers. This can be scaled down. The walk-through formalizes the process of explaining the reviewer's work to his or her colleagues. Attendees would be the project leader, members of the project team, a representative from the QA group, and the technical writer. Customers (end-users) usually attend requirements and preliminary design walk-throughs but usually are excluded from subsequent walk-throughs.

Walk-throughs should have an open, non-defensive atmosphere. This is critical. The goal is to discover and make notes of problem areas. Problems should not be resolved during the walk-through session. Some guidelines include: all code should be reviewed on a scheduled basis, the emphasis must be on detecting errors, the walk-through should focus on major issues not on coding styles, it should have a two hour limit.

Most importantly, walk-throughs must not be used as vehicles for employee evaluation.

MY OWN APPROACH

The defensive nature of the walk-through can be greatly reduced through using a technique I learned about ten years from a former boss, Bill Croughwell. His approach was to remove any possible confrontation between the developer and the reviewer. In this approach, the walk-through is lead not by the coder by but a third-party "walker." There is very little hand-holding in the version of a walk-through. Materials are distributed ahead of time and there is only a very brief overview of the code/document given by the "walker." The "walker" is typically an experienced programmer who is working on the project, but working on an unrelated portion of the project. For example, the Task Leader for Adverse Experience Reporting may lead the walk-through of the Laboratory Data Reporting software. The walker should be an experienced programmer with no personal involvement in the programming.

This type of walk-through is a very controlled meeting and thus is relatively formal. Typically, the "walker" simply goes around the room one person at a time and solicits comments. The focus of the discussions is on problem detection not problem resolution; problems are simply pointed out. No discussion of various solutions takes place. This sometimes requires a strong-willed "walker." The author simply takes notes of the potential problems. Later, the author determines whichever problems she or he wishes to resolve. It is the project leader's responsibility.
to go over the problems with the author to ensure an appropriate response.

There is no confrontation because there is no discussion between the author and reviewer. In actuality, of course, this doesn’t always work. At the "walker’s" discretion some problems may require some discussion to help delineate the exact nature of the problem. It may even be helpful to discuss possible resolutions of the problem. Those cases are exceptions, however, and in general there is little or no discussion of the problems or possible resolutions.

EXAMPLE CODE

The code used for the purpose of this discussion is some interesting SAS/A code which is part of an Executive Information System that ARC developed for a client. The programmer wanted to display results of a SAS procedure to a user interactively; give the user several different options using function keys; and then respond to the selection by the user. All of this must be done without giving the user an opportunity to interact with SAS directly. The code copies the report output into the SAS notebook and defines the function keys; the program calls a second display which handles the interaction with the user; and the program then performs the actions that the user selected.

THE CODE

The following programs are of interest. The RPT_VIEW program which displays the notepad with the report in it. The GET_CMD program which processes the user commands.

REFERENCES


ACKNOWLEDGEMENTS

The author wishes to thank the authors of the code: Bill Lawson and Jeff Phillips from the ARC staff for contributing their code to the project; to Ben Steffen for reviewing this document; and to Judie Mopsik for her general support of ARC’s continuing SUGI activities.

SAS and SAS/A are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. © indicates USA registration.
select (usercmd);
when (opt1) do;
    _msg = "Help Option not available."
    refresh;
end;
when (opt2) do; /* GOBACK */
    call execcmd("notepad;source.cf; end");
end;
when (F3) do; /* GOBACK */
    call execcmd("notepad;source.cf; end");
end;
when (opt3) do; /* print */
    call method("methods.cf","rep_print");
    call method("methods.cf","rep_print_catalog, report");
    call symput("mode","SCREEN");
    rc = fileneme("BSPRNT.");
    call display("prntmsg.program");
    rc = field("cursor", "opt2");
    refresh;
end;
when (F6) do; /* print */
    call method("methods.cf","rep_print");
    call method("methods.cf","rep_print_catalog, report");
    call symput("mode","SCREEN");
    rc = fileneme("BSPRNT.");
    call display("prntmsg.program");
    rc = field("cursor", "opt3");
    refresh;
end;
when (opt4) do; /* UP */
    refresh;
    call execcmd("notepad;backward &scroll");
    rc = field("cursor", "opt4");
    refresh;
end;
when (F7) do; /* UP */
    refresh;
    call execcmd("notepad;backward &scroll");
    rc = field("cursor", "opt4");
    refresh;
end;
when (opt5) do; /* DOWN */
    refresh;
    call execcmd("notepad;forward &scroll");
    rc = field("cursor", "opt5");
    refresh;
end;
when (F8) do; /* DOWN */
    refresh;
    call execcmd("notepad;forward &scroll");
    rc = field("cursor", "opt5");
    refresh;
end;
when (opt6) do; /* LEFT */
    refresh;
    call execcmd("notepad;left");
    rc = field("cursor", "opt6");
    refresh;
end;
when (F10) do; /* LEFT */
    refresh;
    call execcmd("notepad;left");
    rc = field("cursor", "opt6");
    refresh;
end;
when (opt7) do; /* RIGHT */
    refresh;
    call execcmd("notepad;right");
    rc = field("cursor", "opt7");
    refresh;
end;
when (F11) do; /* RIGHT */
    refresh;
    call execcmd("notepad;right");
    rc = field("cursor", "opt7");
    refresh;
end;
when (F12) do;
    call display("get_cmd program", user_id, usercmd);
    if symput("exit_rc") = 1 or
        symput("end_rc") = 1 then goto term; /* test return */
    /* from GET_CMD program */
end;
when (EXIT) do;
    call display("get_cmd program", user_id, usercmd);
    if symput("exit_rc") = 1 or
        symput("end_rc") = 1 then goto term; /* test return */
    /* from GET_CMD program */
end;
otherwise put "otherwise statement executed";
end;
term;
    _status = "14";
    * call execcmd("scroll on");
    return;
*/ from GET_CMD program */
/* MODULE: GET_CMD.PROGRAM */

FUNCTION: Processes user's function key, PMENU, or navigation radio button command.

CALLED BY: All full screen where users have data and command entry capability.

CALLS: HELP.PROGRAM (Displays help based on screen id of original calling program).

EXIT.PROGRAM (Displays EXIT YES/NO dialog box).

SETS MACRO: STATUS used to control SCL_STATUS_variable.

END_RC used to indicate GOBACK condition.

EXIT_RC used to indicate QUIT condition.

USAGE:
CALL DISPLAY('GET_CMD.PROGRAM', screen_id, usercmd);

*/

entry screen_id 47 usercmd 110;

init:
   if usercmd = 'F12' then
      (* DO: user has selected 'EXIT' */
      call display('exit.program');
   end;
   if usercmd = 'F11HELP' then
      (* DO: user has selected 'HELP' */
      call display('nullnode.program');
   end;
   return;
end:

main:
return;
term:
return;

/* the call method block */
catlist:
   method catalog 85 let 88 ;
   refresh;
   submit continue;
   proc catalog catalog = 'query.' & Catalog entrytype = source;
   contents out = 'files (keep = name dest)';
   exit;
   endsubmit;
endmethod;

rpt_print:
   method;
   if ffile('BISPRINT') ne 0 then
      do:
         submit continue;
         ffilesame BISPRINT SYSDOUT = 'A DEST = N2R10';
      endsubmit;
      end:
      call symput('mode','_PRINT_');
      endmethod;

rpt_run:
   method catalog 80 report 88;
   /* make sure preview is empty */
   rc = preview('clear');
   rc = preview('copy' 'query.');
   trim(catalog); trims(source); trim(report); trim(source);
   submit continue; /* force preview to submit */
   endsubmit;
   call symput('mode','_SCREEN_');
endmethod;

*/

126