We have gone over the IBM Instruments Engineerings Scientific Computing System and I would like to finish with a coup of words about where we see this product being used. In general most people are looking at it for at least three areas. One, as a relatively low cost intelligent controller usually with some data acquisition component to it. The IICS is a candidate for process modeling and simulation in control situations. A second application area is as an Engineering Scientific work station. This is a very powerful processor at roughly 10,000 floating point instructions per second with the current level of the operating system and that is without an optimizing compiler for FORTRAN or Floating Point hardware. Along that line with up to 5 megabytes of memory and the speed of the FORTRAN it obviously is very attractive at a FORTRAN application engine and we see a signficant amount of interest as a FORTRAN work station.

3/C/ear/7

SESSION REPORT

61	A053	A Field Study of Human Factors Involved in Debugging FORTRAN Programs	55
SHARE NO.	SESSION NO.	SESSION TITLE	ATTENDANCE
Human Factors		Jim Lipkis	NYU
PROJECT		SESSION CHAIRMAN	INST. CODE

Courant Institute, 251 Mercer St., New York, NY 10012 212-460-7166
SESSION CHAIRMAN'S COMPANY, ADDRESS, and PHONE NUMBER

A FIELD STUDY OF HUMAN FACTORS INVOLVED IN DEBUGGING FORTRAN PROGRAMS

Richard Halstead-Nussloch and Vance Sutton
IBM Corporation
Department C42/701-S
P.O. Box 390
Poughkeepsie, NY 12602

Phone: 914-463-6196

SHARE Installation Code: IBM

SHARE Human Factors Project

Session A053
Thursday 25 August 1983 10:15 A.M.

Permission to publish:

ABSTRACT

Thirty-two professional scientists, engineers, and programmers were interviewed about debugging FORTRAN application programs. The study found that debugging involves the four tasks of symptom location and classification, bug location, hypothesis generation and testing, and information gathering and selection. All study participants want to debug from the level of the FORTRAN source code, but are currently forced to lower, machine oriented levels. When progress towards bug resolution stalls, programmers will either try a different debugging tack or rewrite the problematic source code; whereas, scientists and engineers are likely to drop the program as a method to solve their scientific or engineering problems.

INTRODUCTION

Seidner and Tindall (1) have outlined requirements for interactive debugging facilities (IDFs). Their requirement report indicates that IDFs are seldomly used on large systems, but that debugging is an extremely important part of program development. If IDFs are to be integrated into the program development process, it is evident that their characteristics must conform to user patterns. Ehrman and Hamilton (2) elaborate on many of the features and factors required to integrate debugging and IDFs into program development.

One step towards this integration involves determining how debugging currently is done. To take this step, we conducted a field study of professionals who develop application programs in their work. The study attempted to analyze the tasks and procedures the professionals performed in the debugging phase of program development. The analysis was based on empirical data obtained through interview and survey. The study results are presented here to provide guidance for integrating debugging into program development.

STUDY OBJECTIVES

The objectives of our study were threefold:

1. Analyze the tasks that professionals now perform in program debugging.

2. Identify the information used in debugging, and the sources of that information.

3. Assess the current use of IDFs and other debugging tools.

From the objectives, we identified two specific topics to be included in the survey:

1. Specific debugging tasks performed

2. Information required to do those tasks

DEFINITIONS AND DISTINCTIONS

For our purposes, a _task_ is an integrated unit of user activity having specific goals. For example, compiling a FORTRAN program is a task, but FORTRAN programming is not, because it lacks specificity in its goals. We define the _job of debugging_ as starting with a program that does not work and ending with a working version of that program. Furthermore, the survey was limited to programs that had already successfully compiled, and could be loaded for execution.

We distinguish between symptoms and bugs. A _bug_ is an error in program specification, for example, labeling a COMMON as VARSI instead of VARS1. A _symptom_ is the result of a bug, for example, attempting to divide by zero. From this perspective, the job of debugging involves tasks of determining where and what bugs are present in programs based on the symptoms observed when one attempts to execute the program.

METHOD AND PARTICIPANTS

To meet the study objectives, we conducted a survey through face-to-face interviews at the participants' places of work. A survey form was used to ensure each interview covered the questions in a standardized manner. The study participants were all professional scientists, engineers, or programmers who developed application programs (usually in FORTRAN) as a part of their jobs.

RESULTS AND DISCUSSION

We visited ten major industrial and university computing complexes. The following describes the study results and discusses our interpretations of their meaning.

PARTICIPANT DEMOGRAPHICS

The study included thirty-two professionals. Nineteen were scientists or engineers; thirteen were programmers. Of the thirteen programmers, three were trained as engineers but were now programming to the exclusion of engineering.

USE OF AN IDF OR OTHER DEBUGGING TOOL

When debugging a program, fourteen of the professionals in the study said they do not use a debugging tool; instead, they place WRITE statements throughout the program to trace its execution and determine the values of critical variables as it executes to completion. Table I shows the distribution of scientists/engineers and programmers according to whether an IDF or other debugging tool is used or not:

Table I: Use of a debugging tool by profession

	Scientists/Engineers	Programmers
Use debugging tool	7	11
No debugging tool	12	2

The Chi-square value for table I is 7.16, and with one degree of freedom, is significant at $p<0.01$. The Chi-square indicates that scientists and engineers tend not to use a tool in debugging; whereas programmers tend to use debugging tools. This finding is not surprising given comments when the study participants described their job responsibilities. The scientists and engineers use the computer to perform scientific studies and engineering analyses; the computer is only one of many tools at the disposal of scientists and engineers. If the computer does not deliver the necessary help in the time required, an engineer or scientist will choose another tool instead of trying to overcome the difficulty with the program. Since a programmer is responsible for making programs work, he or she will invest more time and effort in debugging, and consequently will be more likely to use an IDF or other debugging tool.

INFORMATION SOURCES IN DEBUGGING

Figure 1 shows the proportion of respondents who use four information sources in debugging. The information sources are ordered on the horizontal axis according to their increasing semantic richness with respect to the FORTRAN program. A striking feature of the graph is that all the study participants use the source listing (from the compiler) in debugging. As the information moves further away from the source towards the HEX level, its use percentage decreases. No differences were observed between scientists/engineers and programmers on these percentages.

Figure 1 shows that the source listing is the most valuable information source in debugging; indeed, the participants who used, for example, the PSW, indicated being forced to that level in order to obtain necessary information to simply locate a bug in the code. Some additional comments made by the participants about the information sources provide good insight into the value of information sources.

The source listing appeared to play three key roles in debugging.

1. A map

2. Notebook

3. Source of hypotheses

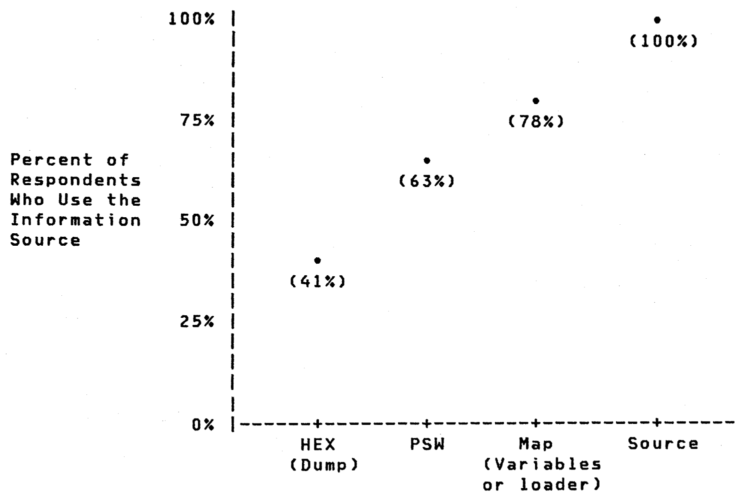

Figure 1: Use of information sources in debugging

The source listing functions as a map to guide the overall debugging job. The respondents said they used the source code to determine both how the program got into subroutines (CSECTS), and where it could potentially go following resumption of execution. They indicated the source code allowed the use of subroutine names as major landmarks. To them, this represented a significant improvement over being restricted to hex addresses for landmarks. The users said that they often make notes on the source listing when debugging. The notes serve to remind them of what transpired during the debug session, and what corrections are required to fix the program. Finally, the source listing provides users with information to develop hypotheses about the bugs.

Over one-half of the respondents used information about variables, e.g., a variable cross-reference map. The primary use of this information was for bugs involving typographical errors and labelled COMMON areas.

About one-half of the respondents used information from loader/linker and operating system (PSW, hex dumps, load maps, etc.) in debugging FORTRAN programs. In about one-half of the cases, the respondents indicated use of information at the hex level was a function of necessity rather than desire.

Although not listed in Figure 1, all of the respondents said they used information about program execution, either in the form of program output, or the output of a formal debugging tool. Through the program output, the behavior of the program can be traced. Symptoms appear in the output to aid in error and fault diagnosis. The comments indicated the program output has information value that is second only to the source code.

Overall, the primary sources of information for debugging appear to be source listing and the program output. Most respondents said they try to focus on the source code in debugging, and all use it as a map and notebook in fixing bugs. The primacy of the source code in debugging is reinforced by the responses received to a question about the level that the respondents would desire to work when debugging (see below). All the respondents indicated a desire to stay at the FORTRAN source level, and not be required to work at the machine level, for example, decode PSWs and compute hex offsets.

PURPOSES OF DEBUGGING

The professionals included in the study were asked whether they used a debugging procedure for the following five purposes:

1. Bug location

2. Data error location

3. Execution-path error location

4. Program testing

5. Error patching (temporary)

Almost all the respondents use debugging techniques to locate bugs, data errors, and execution-path errors. This makes sense because debugging techniques are designed to locate bugs and errors.

About one-half of the professionals in the study regularly use a debugging technique for program testing. The other one-half either do not test programs or use some other methodology. These data show that more scientist/engineers regularly test programs with debugging techniques (eleven of nineteen) than do programmers (four of thirteen). This difference is not statistically significant, however.

The single statistically significant difference between the scientist/engineers and programmers occurred for temporarily patching an error. Programmers tend to temporarily patch errors and continue debugging (eight said they did so), whereas scientist/engineers do not (fourteen said they never do so). (For this difference, Chi-square is 3.97, with one degree of freedom, and is significant at $p<0.05$). The scientist/engineers said they would rather edit the source code and recompile for each error. The programmers who did use an IDF for this purpose indicated that it increased their efficiency in debugging--they could find and fix more than one bug in a single session. In their minds that added to their productivity, by reducing the amount of their time and processor time required for program development. In short, the programmers who employed a debugging technique to patch faulty code did so to better complete their professional responsibilities.

SPECIFIC USE OF AN EXISTING FORTRAN IDF

The study also examined the use of specific commands from the FORTRAN interactive debug for CMS and TSO (program number 5734-F05, called, here, TESTFORT). Of the thirty-two study participants, sixteen (50%) had used TESTFORT. Of the TESTFORT commands, only the AT, LIST, and GO commands were reported to be necessary in every debugging session by all sixteen professionals who had actually used TESTFORT. Other commands, although not as heavily used as AT, LIST, and GO, were reported as necessary to complete all of the functions required of an IDF. Since AT, LIST, and GO are used by all who use an IDF, and they simulate a WRITE statement (which is used by those who do not use an IDF), they appear to constitute the minimum set of commands for an IDF.

USE OF ABEND DUMPS IN DEBUGGING

For the study, we collected information on the value the professionals in our study found in ABEND dumps. The responses to that question are in Figure 2:

```
   Value                Scientist/Engineer       Programmer
   -----                ------------------       ----------

ABEND not recognized                     ( 0%)  |||        ( 9%)
Never use dump        ||||||||||||||     (44%)  ||         ( 6%)
Information fair      ||||               (13%)  |||||      (16%)
Information good      |                  ( 3%)  |          ( 3%)
Information excellent                    ( 0%)  ||         ( 6%)
```

Figure 2: Information value of ABEND dumps

Figure 2 shows that scientists and engineers do not use ABEND dump information; programmers gave responses that were distributed throughout the scale. Overall, it appears ABEND dump information is of greater value to programmers.

LEVEL AT WHICH DEBUGGING IS PERFORMED

The study included questions about the level of detail and semantic richness that the professionals found it necessary to work when debugging and the level they preferred to work. The responses to the question on the level of necessity were distributed as in Figure 3.

```
Necessary level         Scientist/Engineer       Programmer
---------------         ------------------       ----------

Mathematics/logic                        ( 0%)  |          ( 3%)
Source code           ||                 ( 6%)  ||         ( 6%)
Number representation |||||||||||||||    (47%)  ||||||||   (25%)
Machine operations    ||                 ( 6%)  ||         ( 6%)
```

Figure 3: Level of detail necessary during debugging

The responses to the question on the desired level were distributed as in Figure 4.

Preferred level	Scientist/Engineer		Programmer	
Mathematics/logic	|||	(9%)	|||||||	(22%)
Source code	|||||||||||	(34%)	|||||	(16%)
Number representation	||||	(13%)	|	(3%)
Machine operations	|	(3%)		(0%)

Figure 4: Level of detail preferred during debugging

Overall, Figures 3 and 4 show that almost all study participants are forced to debug at the level of how FORTRAN represents numbers, and this is a more detailed and machine-specific level than generally desired. Most scientists and engineers would prefer to work at the source-code level; whereas, programmers show a small tendency toward the mathematics and logic level.

TASKS IN DEBUGGING

The study attempted to describe the tasks involved in debugging by constructing a composite list of the activities the participants said they performed when determining why a program does not work. The analysis shows that debugging involves four critical tasks, regardless of who is performing the debugging or whether an IDF or other debugging tool is employed. All the tasks appear to be necessary to successfully debug a program; none of the identified tasks appears sufficient in and of itself for debugging. The four tasks are:

1. Symptom location and classification

2. Bug location

3. Hypothesis generation and testing

4. Information gathering and selection

Symptom Location and Classification

We have distinguished between symptoms and bugs. (A bug is an error in program specification, for example, labeling a COMMON as VARSI instead of VARS1. A symptom is the result of a bug, for example, attempting to divide by zero.) When performing the symptom-location subtask, the user tries to answer the question: Where in the program execution did things go wrong? In the symptom-classification subtask, he examines the question: What went wrong in program execution?

Bug Location

The bug-location task poses this question to the user: Where is the program incorrectly specified?

All the respondents in the survey start debugging by classifying the symptoms they can observe and then (or simultaneously) determining where in the program the bug is located. They look for this information in program output and source listings. In some cases this information can not be found in program output, for example, where the program executes to completion but produces incorrect answers; here, the user will attempt to gather information about the symptom's and bug's location using either an IDF or inserting WRITE statements into the source code.

Hypothesis Generation and Test

The hypothesis-generation subtask involves establishing a set of the possible specification errors (bugs). The user answers the question, what could be wrong in the source code? From the survey results, this task appears to be based upon the symptoms and information from the source listing.

The hypothesis-testing subtask poses the question, which of the possible specification errors were not made? It involves an iterative reduction in the size of the set of possible bugs down to the actual bug.

Not surprisingly, the results show great variety in the respondents' approach to these two subtasks involving hypotheses. For example, some start at the level of the underlying mathematics of the application in developing possible explanations; whereas others immediately go to the pseudo-assembler code to generate hypotheses. Regardless of the level where hypothesis generation and test activity starts, respondents uniformly indicated they were required to deal with the bug at a level below source code, for example, hexadecimal representation or machine operations, if they wanted any chance of success.

The study indicates that the hypothesis generation subtask is critical in debugging. Two specific situations uncovered by the survey illustrate this critical nature of the hypothesis-generation subtask. One expert programming consultant resolves bugs in programs developed by engineers that have already gone through an initial review (by another programming consultant). The expert requires that the engineer provide a current compiler source listing, a program input listing, and a program output listing. The expert may or may not add information obtained from an IDF to provide the engineer with a list of hypotheses about the bug. Also, the consultant provides the engineer with directions on how to fix the bug, appropriate for each hypothesis. This case illustrates that since the expert provided the hypotheses, and they were necessary to solve the bug, hypothesis generation is a critical debugging subtask.

The second specific situation originated from comments provided by many of the engineers. They indicated a reliance on programmer consultants to provide "guidance and direction" in interpreting the symptoms and developing hypotheses about the bug. One group of engineers also described a specific need to obtain hypotheses about the bug in mathematical and not computer terms and that they seek only those consultants who could make the necessary translations. The survey results indicate engineers perceive value in appropriately termed hypotheses about bugs.

Information Gathering and Selection

According to the survey respondents, the information gathering and selection subtasks are performed for three reasons:

1. To locate the bug and determine symptoms.

2. To ensure progress towards bug resolution.

3. To eliminate hypotheses about the bug.

Sometimes the user can not locate the bug and/or determine the symptoms from the source listing and program output alone. In these cases, an initial round of information gathering is performed to do so.

An important purpose for gathering information is to ensure that progress is being made towards bug resolution. In cases where progress is perceived to be too slow, professional programmers said they would either try a different debugging tack or rewrite the offending source module. In contrast, professional engineers and scientists said they would either rewrite the source code or completely drop the program as an alternative method to solve the engineering or scientific problem. Engineers and scientists cite time pressures as the reason for abandoning programs.

Information gathering and selection to eliminate hypotheses about a bug is undertaken in a simple and basic approach in most every debugging session. (Some elaborate on this approach for more difficult bugs.)

> After the bug has been tentatively located, the symptoms classified and hypotheses generated, breakpoints are established at a number of statements (usually less than five). At those breakpoints the contents of critical variables are displayed as the program executes and the symptoms develop.

The survey results indicate that regardless of whether a formal tool was employed, users performed the four tasks described above in debugging a FORTRAN program. The survey results showed great uniformity across respondents in approach to symptom location and classification, bug location, and information gathering. Great variety was observed in approaches to hypothesis generation and testing.

SUMMARY

In summary we see three major points. First, source-level debugging has primacy. All professionals in our survey want to "fix it in the language in which it broke" (2). The thought that goes into debugging depends on the source code to a very large degree.

Second, regardless of whether IDFs are used or not, four tasks are regularly applied in the job of debugging. They include symptom location and classification, bug location, hypothesis generation and testing, and information gathering and selection.

Third and finally, when debugging becomes complicated and difficult, or progress slows significantly, programmers will continue debugging by trying alternative tacks. In contrast, scientists and engineers currently will abandon the program as a method to solve their substantive problems.

REFERENCES

1. Seidner, R.I., and Tindall, W.N.J. Interactive Debug Requirements. IBM Technical Report TR-03.218, March 1983.

2. Ehrman, J.R., and Hamilton, H. The thrill of programming--the agony of debugging. SHARE 57, 1981.

SHARE

SHARE SESSION REPORT

61	A054	An Empirical Study of the Use and Effectiveness of Online Documentation	65
SHARE NO.	SESSION NO.	SESSION TITLE	ATTENDANCE
Human Factors		June Genis	STU
PROJECT		SESSION CHAIRMAN	INST. CODE
CIT, Stanford University, Stanford, CA 94305 (415) 497-4422			
SESSION CHAIRMAN'S COMPANY, ADDRESS, AND PHONE NUMBER			

An Empirical Study of the Use and Effectiveness of Online Documentation:

Final Report

Presented at:

Session A054
SHARE 61
New York, N. Y.

Joan M. Winters
Stanford Linear Accelerator Center (SLA)
P.O. Box 4349
Stanford, California 94305

Keith J. Sours
SPSS inc. (SPH)
Suite 3300
444 N. Michigan Ave.
Chicago, Illinois 60611

August, 1983