

FINAL REPORT to the NASA/Goddard Space Flight Center

Grant Number: NAG⁵-3156 *FINAL IN-61512*

PROJECT COMPLETION DATE: December 14, 1996 *ORIT*

PROJECT TITLE: Applied Research Study *CR-550*

PRINCIPAL INVESTIGATOR: Ronald J. Leach

Department of Systems & Computer Science
College of Engineering, Architecture, and Computer Sciences
Howard University
Washington, DC 20059
(202) 806-6650
rjl@scs.howard.edu

Project Summary

The purpose of this project was to study the feasibility of reusing major components of a software system that had been used to control the operations of a spacecraft launched in the 1980s. The study was done in the context of a ground data processing system that was to be rehosted from a large mainframe to an inexpensive workstation. The study concluded that a systematic approach using inexpensive tools could aid in the reengineering process by identifying a set of certified reusable components.

The study also developed procedures for determining duplicate versions of software, which were created because of inadequate naming conventions. Such procedures reduced reengineering costs by approximately 19.4 percent.

Introduction

The IMP-8 spacecraft is used to collect scientific data. It is so reliable that it is used by NASA's Goddard Space Flight Center to calibrate instruments and data transmission equipment for much newer spacecraft. IMP-8 is solar powered, has few moving parts to wear out, and is expected to remain operational for many years. All data transmission must be processed in real-time.

A spacecraft in orbit around the earth must transmit data to ground antennas or relay systems. The position of the spacecraft can be determined exactly by knowing the precise time and the orbit of the spacecraft. Because of the distances involved, extreme accuracy is important in order to determine where cameras are pointing.

The relative position of the cameras, antennae and the body of the spacecraft are essential to allow clear vision as the spacecraft rotates. This calculation is called "attitude determination." Errors in orbit are detected by complex calculations using the known positions of the Sun, Moon, several stars, and the planets; this is known as "ephemeris information."

The cost of operating the spacecraft from the ground has become uneconomical in recent years. The primary reasons are the need to maintain the mainframe computers

needed for the initial, "front-end" processing and the expensive collection of tape drives and controllers. The mainframes and tapes are used to process the data obtained from relay antennas. The data comes into the sensor data collection area electronically using the Internet and ftp.

The primary data collection and storage is done in two large adjacent rooms with raised floors and high performance air conditioning. Other relevant sites are a control center where commands are sent to the IMP-8 spacecraft to send data to the data collection area and several off-site locations where scientists analyze the archived data. No changes in data format are allowed because of the need to process analog data. Analog data is to be transformed to digital, with several quality checks and other data operations. Other changes may call for the replacement of reels of tape with other units.

Input data is of two types:

1. McMurdo station in Antarctica uses the Internet and ftp to send input data in digital format into a Pentium 3 processor.
2. Data from Wallops Island and two other sources is sent to magnetic tape units. This is analog data.

A set of two IBM 3090s (1980s technology) is used to take both types of data and to produce two primary actions:

- Step 1 includes pre-edit processing to perform a pre-quality check in which timing information is extracted and put onto a tape, with the science information also extracted.
- In step 2, data is arranged into chronological order. The data is grouped into 4.333 "day groups" and other processing is done.

Most of the software is written in a combination of FORTRAN and Unisys assembly language. The FORTRAN version is pre-FORTRAN77, and may in fact be as early as FORTRAN2.

The control center is 1970s technology with proprietary hardware and software driving computers with large screen terminals that are essentially color alphanumeric displays. The data communications model is neither TCP/IP nor a proprietary NASA protocol called NASCOM.

Current technology uses one of two configurations for projects of this complexity, with UNIX workstations far more likely to be used:

1. HP workstations, HP-UX, Motif, ANSI C, TCP/IP, Berkeley UNIX sockets, Oracle database, and other COTS products integrated with both reused and special-purpose software.
2. Pentium PCs, ANSI C, TCP/IP, Oracle database, and other COTS products integrated with both reused and special-purpose software.

The mainframe computers are 1970s vintage Unisys machines with an expensive proprietary operating system that costs in excess of \$200,000 per year in licensing fees. This fee will be raised next year because the current fee is pro-rated across several different projects also using this expensive proprietary operating system and these other projects will either terminate or else switch to newer technology. The estimated licensing and support cost is in the area of \$500,000 annually. This is exclusive of operations.

There are approximately 25-30 tape drives and controllers, each of which is about five feet high and occupies a square of approximately 2.5 feet per side. The tape drives accommodate large reels of nine track magnetic tape with a density of 6250 bpi, an inter-record gap of 3/4", and are approximately 2000 feet long. The tapes are controlled by two IBM 3090 mainframes, of 1980s vintage. (The tape backup system was updated from the original system.)

The amount of data that can be stored on the tape is difficult to estimate because it depends upon the size of a record. The amount of data cannot be larger than 150,000,000 bytes per tape and is probably closer to 120,000,000 bytes per tape.

The backup tapes are stored in storage cases that are over ten feet long and six feet wide, with three or four shelves. There are at least 8 of these cases. They are kept in the same very large computer operations room as the tape drives. Operators are need to load and remove tapes from the individual tape drives. There is not sufficient disk capacity to keep the contents of all the tapes on-line. Tapes are archived for two years.

Not surprisingly, there are often errors on the tapes caused by degradation of the magnetic media due to abrasion and other factors. Reducing the handling of tape is a major goal.

Writing drivers for newer storage devices such as 4 millimeter, 8 millimeter, or DAT cartridges appears to be an impossibility due to the age of the operating system and the lack of personnel. The enormous yearly software and hardware maintenance cost does not include facilities for new software development for software that is likely only to have a single user.

The Traditional Reengineering Approach

Because much of the original software development and software maintenance was done by private contractors, it was initially decided to have the same contractor that developed the software perform the reengineering. Of course most of the project personnel had been reassigned or retired, so there was little institutional knowledge. (Much of the maintenance had been done by a second contractor and there was some knowledge of the source code.)

Since requirements, design, rationale, test plans, or discrepancy report data were not available, the analysis focused on the source code.

The system consisted of 33 source code files, totaling 108161 lines, which were grouped into 14 "programs." In modern terminology each "program" would represent source code for distinct concurrent processes. It was difficult to determine which source files were associated with which "programs." Only rudimentary naming conventions were used, reflecting the state of software engineering at the time the software was developed.

Most of these files contain both FORTRAN and Unisys assembly code. No attempt was made to measure the size of the system in source lines of code of other (locally) standardized measure.

In order to determine the cost of the reengineering effort, a decision was made to reengineer one "program," accounting for approximately one fourteenth of the total system size. The cost of this "program" was to be multiplied by 14 to obtain the cost of reengineering the entire system.

The Reengineering Approach Used in This Project

The most important questions to ask during any reengineering effort are:

1. What software components should be reused?
2. Which software components should not be reused?
3. In what capacity are the components reused?
4. Which methods are likely to be successful at reengineering existing code?
5. What inexpensive tools are available reengineering?
6. What are the total life cycle costs of reengineering?
7. How do these costs compare to new development?
8. Do the perceived benefits outweigh the cost?

The goal of the first phase of the reengineering effort was to understand the software using publicly available software utilities. The intention was to determine the amount of information that could be obtained from such utilities. Since no documentation of the requirements, design, or rationale were available, the work focused on the source code. The software was transferred electronically to a HP workstation running HP-UX at Howard University.

The software consists of 33 files comprising 108,161 lines of combined FORTRAN and Unisys assembly language. Most files contained both FORTRAN and assembly code. No attempt was made to measure the size of the system in source lines of code or other (locally) standardized measure.

Since the original source code files contained functions in multiple languages, we used the UNIX `csplit` utility to separate the files. The `csplit` utility looks for a user-defined delimiter (in this case the delimiter was `./ADD NAME=`) and breaks each file into separate files. The shell script was:

```
for i in *.SEQ
do
echo $i
csplit -k -f $i $i './ADD NAME/+1' '{99}'
done
```

This operation had to be performed twice because some of the delimiters had an extra space between the `'/'` and the beginning of the word `'ADD.'` The delimiter was removed by using the UNIX `sed` text editing utility in the shell script

```
for i in *[0-9]
do
sed "1,1 d" $i > $i\.f
done
```

The final result of this sequence of splits was a set of 764 source code files, each with the `.f` extension, regardless of whether they were FORTRAN or Unisys assembly files.

The UNIX `wc` utility was run on these 764 files to determine if there were any files of exactly the same size. Such pairs were then compared using the `diff` utility to determine if there were any duplications. We determined that 180, or 19.4 %, of the functions being reengineered were duplicates, probably grouped because of the lack of naming conventions and makefiles.

Of the remaining 584 source code files, 90 were rejected for reuse because they were written in Unisys assembly language and would have to be rewritten entirely.

The next step was to certify the remaining 494 FORTRAN source code modules as being correct enough to be potentially reusable. The modules were tested for several potential problems each of which would render the module unsuitable for reuse:

- The code failed to compile using the portable gnu `g77` compiler.

- The McCabe cyclomatic complexity was greater than 10. This is especially relevant because the current software standards and practices manual requires only branch testing. We used the fsca (FORTRAN Source Code Analyzer) previously developed at Howard University [8].
- The interconnection from a function to others is "large." This metric was also computed by the fsca tool.
- If the FORTRAN source code can be translated directly to C in order to use the code on more modern computers.

The McCabe cyclomatic complexity was checked by using the UNIX awk utility to determine which, if any, files exceeded the pre-set limit. The awk script was an examination of the output of the fsca utility. The awk script was

```
if ($3 == "CYCLOMATIC")
{
  getline
  getline
  print $2
}
```

The flow of control was determined by the floppy and flow utility. These utilities are robust in the sense that they ignores any foreign (e.g., Unisys assembly) code encountered.

The data on the McCabe cyclomatic complexity, size of function interface, and control flow information provide an assessment of the current state of the software. This data can be compared to the result of reengineered software to determine if anything was omitted in the reengineering effort.

Examples of the output of the fsca tool used for certification are shown below. Several types of coupling [3] were calculated by the fsca tool [8], but only the total amount of coupling was used.

module name: NORVEC
cyclomatic complexity: 3
lines of code: 18
of interconnections: 2

Module Name	Number Times	None	Weak	Stamp	Control	Common
VECTOR	1	FUNCTION NOT DEFINED IN SOURCE CODE				
SQRT	1	INTRINSIC FUNCTION				

module name: ENGCAL
cyclomatic complexity: 57
lines of code: 189
of interconnections: 6

Module Name	Number Times	None	Weak	Stamp	Control	Common
ITABLE	28	FUNCTION NOT DEFINED IN SOURCE CODE				
COEFF	1	FUNCTION NOT DEFINED IN SOURCE CODE				

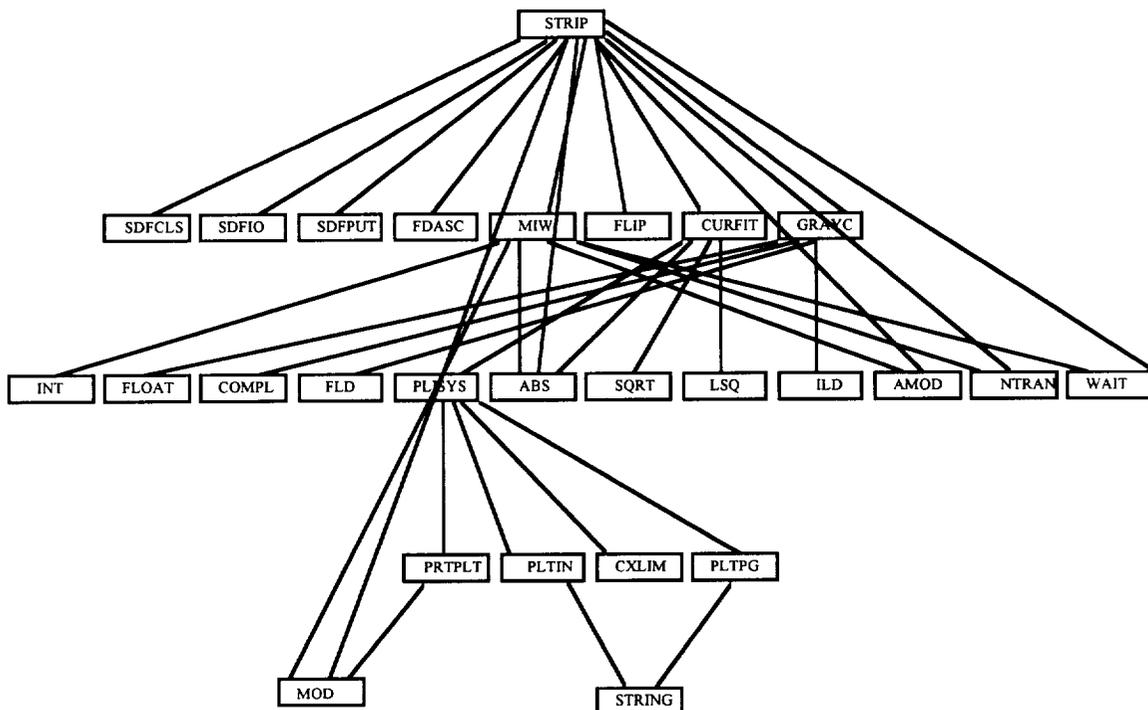
ACSLOG	1	FUNCTION NOT DEFINED IN SOURCE CODE				
CALTAB	3	N	N	14	7	8
OR	1	FUNCTION NOT DEFINED IN SOURCE CODE				
INTERP	1	N	N	27	15	0

We found that 209 files exceeded the recommended limit of 10 for the McCabe cyclomatic complexity. These functions are likely to require special testing efforts if they are changed in any way. Indeed, the first rehosted "program" crashed on its first execution because a new, previously untested execution path led to an uninitialized array on the new host workstation. Even though the IMP-8 system runs essentially error-free in the present hardware and software environment, individual modules clearly have to be certified as correct before reuse. Any module having any of the above characteristics will not be certified as error-free.

The analysis also determined that 2% of the functions in the system are of abnormal length (functions with more than 500 lines of code), 4% utilize abnormal FORTRAN semantics (columns 73-80), 25% contain assembly language code, and 6% have a very large number of interconnections (functions which have greater than 10 interconnections).

New documentation was developed using the public domain tools floppy [4] and flow [5] to obtain both graphical and textual representations of the detailed software design. Floppy and flow are used in unison on FORTRAN programs to produce structural flow graphs as well as metrics information on the number of source code lines and the number of commented lines. Unfortunately, the number of commented lines may be somewhat skewed because the floppy and flow consider any lines they cannot interpret to be comment lines. A sample of the output from the flow tool is shown below.

Flow Diagram for ATTITUDE.SEQ



The documentation and metrics for the software provides a basis for comparison to make sure that newly written or ported modules are functionally equivalent to the original modules in their implementation details.

Domain Analysis

“Domain analysis is a generalization of systems analysis, in which the primary objective is to identify the operations and objects needed to specify information processing in a particular application domain. In addition, domain analysis will identify precisely domains and software artifacts within these domains that are good candidates for reuse.” [7, pg. 35-36] We used a bottom-up approach using basic building blocks (a faceted classification scheme) for domain analysis.

After careful evaluation of the code, it was decided that only the commented regions of the code would be considered when looking for objects and verbs. The commented regions contain a description of the function, the purpose of the function, the English name of the function, methods of the function, modularity of the function (coupling, cohesion), implementation notes for the function, variables used by the function, calls made by the function, and which other functions call the function. By utilizing this strategy, the amount of code to be considered was drastically reduced, however, the domain analysis results were not greatly compromised because of the rich content of the commented regions.

In some cases the function comments did not include all of the above information. If that were the case, the contents of the particular function were examined to determine the objects and actions.

A small program, ‘comment.c’ was written to pull out the commented regions as well as other significant lines from each module in the source code and placed them in an output file. Next, the output files containing the commented areas served as input to the Concordance tool [2]. This tool, designed for the Macintosh, produced an index of all the words contained in the input file along with the number of occurrences of each word and the line numbers in the input file where the words are located. The generated index expedited the domain analysis process by generating a word list which was used to determine object, verb, and system agents. Approximate values for the possible reuse of each file in the system were determined. We also used a software tool from the ASSET library (source.asset.com) to create a “concordance,” or cross-reference of the source code by treating the embedded comments as a text document.

The Results of the Software Analysis

The software consists of 33 files comprising 108,161 lines of combined FORTRAN and Unisys assembly language. Most files contained both FORTRAN and assembly code. The original source code files contain 764 functions. Using the UNIX wc and diff utilities indicate that there are 180 functions that are duplicated, with a total of 20,333 lines that are duplicated. Thus the reengineering effort can be reduced by approximately 19.4% over the cost of reengineering the entire project with no loss of quality.

Of the 765 files, 209 exceed the recommended limit of 10 for the McCabe cyclomatic complexity. These are likely to require special testing efforts. For example, the following functions in the ATTITUDE.SEQ file require special attention:

function number	McCabe cyclomatic complexity
-----------------	------------------------------

6	25
9	16
10	55
11	29
18	13
20	44
21	17

We concluded that approximately 60% of the IMP-8 system is a candidate for reuse. Much of the software can be used as is, with extensive testing of converted FORTRAN modules necessary for those with large interfaces of high cyclomatic complexity.

Lessons Learned About Reengineering Analysis

There were no similar systems located to assist in evaluating the reuse potential of the IMP-8 software. Therefore, it was necessary to implement an original methodology for performing the domain analysis, the metric analysis, the certification, the reengineering and the economic analysis of the system.

There were several lessons learned throughout the analysis process. First, since the functions of the system were large in number of lines of code, separating the functions into smaller parts made analyzing the code much more manageable. This separation allowed us to determine which files were duplicates. It is unlikely that these duplicates would have been found without this separation. In addition, the separation improved the efficiency of the code analysis tools used for program understanding, because several of these tools could not be run on large source code files because of memory constraints.

Second, in analyzing the data, no single public domain tool should be relied upon, due to the lack of testing that would be appropriate for a . All tools utilized for conducting the analysis had to be used together. For example, the 'fsca' tool determined the module calling tree which was similar to the results produced from the 'floppy' tool, which produced a graphical module calling representation.

Third, there were several methodologies available to perform the above mentioned analysis tasks. However, the simpler approach was always taken because it would be easier to understand the results. Utilizing the bottom-up approach made domain analysis easier since we were not familiar with the code.

Comparisons to a Traditional Reengineering Approach

The reengineering approaches suggested in [1], [7], [9], and [10] all focus on the need to determine a correct view of the system being reengineered. Even though many program understanding tools are limited, the use of tools is highly recommended as a general practice. This is the most common approach to developing an abstract view of the system, which is necessary if portions of the system are to be reengineered into new systems with (at least) the same functionality.

In this project, the public domain floppy and flow tools were used for program documentation, together with the locally-developed fsca tool for metrics. This provided the high level view of the software system; obtaining this view is the first step in the processes recommended in [1], [7], [9], and [10].

However, our approach used these tools for high level analysis only after the system was decomposed into individual modules. We were able to reduce the cost of reengineering the entire system by nearly 20 percent, by selecting this approach. The

same approach would be effective when reengineering any legacy system that was developed before naming conventions were enforced. Such systems often have multiple copies of the same functions embedded into larger source code files.

There is one final note on this approach. The reengineering process traditionally is considered to be either a monolithic process, with an existing system as input and a reengineered system as output or an incremental approach. Most reengineering processes are tailored towards the monolithic approach. The separation process described here is also appropriate for incremental approaches, with little overhead cost.

Appendix

A number of scripts including those below were written to facilitate the analysis.

SHELL SCRIPT FOR SYSTEM ANALYSIS (Script name: analysis)

```
echo "STARTING ANALYSIS FOR THE IMP-8 SYSTEM FILES..."
mkdir $1
cp COPY/* $1
cd $1
for i in [A-Z]*
do
    echo "Creating Directory $i.DIR..."
    mkdir $i.DIR
    mv $i $i.DIR/$i.SEQ
done
for i in *.DIR
do
    cd $i
    echo "DATA ANALYSIS FOR <=====>"
    $i">>$i.DOC
    echo "FILES\t\t\tSTART DATE\tFINISHED
DATE\t\t\tCOMMENTS">>$i.DOC
    echo "-----"
">>$i.DOC
    for j in *.SEQ
    do
        csplit -s -k -f $j $j '/ADD NAME/+1' '{99}'
        rm -f $j
    done
    for j in *[0-9]
    do
        mkdir $j.DIR
        echo $j>>$i.DOC
        floppy -t -cn $j>>$j.txt
        flow -g $j.floptre>>$j.txt1
        rm -f *.floptre *.flopold
        mv $j* $j.DIR
        mv flow.ps $j.DIR
    done
    cd ../
done
```

SCRIPT TO COMPILE THE FORTRAN SOURCE CODE

```
for i in *.f
do
f77 -col72 $i
done
```

SCRIPT TO SEPARATE FORTRAN, ASSEMBLY, AND DOCUMENTATION

```
for i in *.DIR
do
cd $i
cp *.f REUSE/IMP8.18/FORTRANFILES/DISK
cp *.txt2 REUSE/IMP8.18/DOCUMENTATION
cp *.a REUSE/IMP8.18/ASSEMBLY
cd ../
done
```

References

1. Arnold, R. S. ,ed., *Software Reengineering*, IEEE Press, Los Alamitos, California, 1992.
2. Concordance Tool, www.sil.org/computing/conc/conc.html
3. Conte, S. D., Dunsmore, H. E., and V. Y. Shen, *Software Engineering Metrics and Models*, Benjamin Cummings Publ., Menlo Park, California, 1986.
4. Floppy,nic.fonet.fi/pub/languages/fortran/floppy7.tar.Z
5. Flow,nic.fonet.fi/pub/languages/fortran/floppy7flow3.tar.Z
6. IMP-8 description, <http://nssdc.gsfc.nasa.gov/space/imp-8.html>
7. Leach, R. J., *Software Reuse: Methods, Models, Costs*, McGraw-Hill, New York, 1997.
8. Leach, R. J., and D. M. Coleman, "A Software Metric for Logical Errors and Integration Testing Effort," Proceedings of the Computer Assurance Conference, COMPASS'97, Gaithersburg, Maryland, June 18-20, 1997.
9. Software Reengineering Analysis Handbook (SRAH), Hill Air Force Base, <http://www.stsc.hill.af.mil/RENG>.
10. Wilkening, D. E., Loyall, J. P., Pitarys, M. J., and K. Littlejohn, "A Reuse Approach to Software Reengineering," *J. Systems Software*, Vol. 30., No. 1-2, July-August, 1995, pp. 117-125.