SOFTWARE REENGINEERING

Ernest M. Fridge III
Deputy Chief, Software Technology Branch/PT4
NASA/Johnson Space Center
Houston, Texas 77058
(713) 483-8109
Nasamail: EFRIDGE

ABSTRACT

Today's software systems generally use obsolete technology, are not integrated properly with other software systems, and are difficult and costly to maintain. The discipline of reverse engineering is becoming prominent as organizations try to move their systems up to more modern and maintainable technology in a cost effective manner. The Johnson Space Center created a significant set of tools to develop and maintain FORTRAN and C code during development of the space shuttle. This tool set forms the basis for an integrated environment to reengineer existing code into modern software engineering structures which are then easier and less costly to maintain and which allow a fairly straightforward translation into other target languages. The environment will support these structures and practices even in areas where the language definition and compilers do not enforce good software engineering. The knowledge and data captured using the reverse engineering tools is passed to standard forward engineering tools to redesign or perform major upgrades to software systems in a much more cost effective manner than using older technologies. A beta version of the environment was released in March, 1991. The commercial potential for such reengineering tools is very great. CASE TRENDS magazine reported it to be the primary concern of over four hundred of the top MIS executives.

INTRODUCTION

Programs in use today generally have all of the functional and information processing capabilities required to do their specified job. However, older programs usually use obsolete technology, are not integrated properly with other programs, and are difficult to maintain. Reengineering is becoming a prominent discipline as organizations try to move their systems to more modern and maintainable technologies. Johnson Space Center's (JSC) Software Technology Branch (STB) is researching and developing a system to support reengineering older FORTRAN programs into more maintainable forms that can also be more readily translated to a modern language such as FORTRAN 8x, Ada, or C. This activity has led to the development of maintenance strategies for design recovery and reengineering. These strategies include a set of standards, methodologies, and the concepts for a software environment to support design recovery and reengineering.

This document provides a brief description of the problem being addressed and the approach that is being taken by the STB toward providing an economic solution to the problem. A statement of the maintenance problems, the benefits and drawbacks of three alternative solutions, and a brief history of the STB's experience in software reengineering are followed by the STB's new FORTRAN standards, methodology, and the concepts for a software environment.

STATEMENT OF THE PROBLEM

Based on trends in the computer industry over the last few years, it is clear that computer hardware, languages, and procedures are not static. The software industry recognizes that a large existing software base must be dealt with as new software engineering concepts and software technologies emerge. The old systems use outdated technology and are costly to maintain. At JSC, as in industry at large, there is a large investment in existing FORTRAN software. These FORTRAN systems do not consistently use modern
software practices that can increase maintainability. Yet these systems must be maintained for perhaps the
next 20 years. Management is seeking ways to reduce maintenance costs.

In the 1960s-70s many FORTRAN programs were developed at JSC, each with its own sizeable software
development team, and its own input/output format. These programs could not communicate readily and
eventually were "wired" together in a very crude semblance of integration. Standards could not be enforced
because FORTRAN did not enforce them and some were not visible by just looking at the code. The
problem was aggravated by the lack of training of new developers plus a 50 percent turnover in the very
large development staff every two years. In addition, the user organizations had more people doing
development than the development group, and these other organizations were not always aware of the
standards and support tools available. This history has left JSC with the following problems:

- Many programs are large and difficult to understand, resulting in maintenance problems.
- The problems in maintenance led to users keeping their own versions of programs, resulting in
tremendous duplication.

Many of the FORTRAN programs have already been converted from their original dialect of FORTRAN to
the FORTRAN 77 standard. Additional conversions will periodically be required even if only to new
FORTRAN standards. It is necessary to consider the question, where will that code have to be in five or ten
years? Three possible answers come to mind:

- FORTRAN 77 is the current standard, but this will be replaced by newer Fortran standards. As
  vendors stop supporting FORTRAN 77, existing FORTRAN will have to move to the new
  standard or to another language.
- Much of the code may move to the Ada language. This will be particularly true on Space Station
  Freedom work.
- With C being the language of choice for Unix, some of the code might move to the C language.

ALTERNATIVE SOLUTIONS

Three alternative solutions to the problems identified above have been identified: complete redevelopment
of the program, code translation to a more modern language or version of a language, and reengineering.
Each of these is illustrated in figure 1 and discussed briefly in the following paragraphs.

Redevelopment of a system from scratch is very expensive. Redevelopment includes all of the same phases
of the life cycle as new development, from requirements through integration and testing. Extensive domain
analysis is required, and there is a risk of incomplete requirements. All too often it is reported that a large
program will be redeveloped from scratch to a more modern style only to find out that the new developers
did not understand all of the functions and necessary information requirements of the existing system.

Code translation, especially automatic code translation, costs much less. Some might then ask, why worry
about all of this now? We can use a translator when the time comes that we are forced to move the code for-
ward. Although this would be a nice solution, the truth is that code translators have proven unsuccessful due to several major reasons:

- Poor existing control flow is translated into poor control flow.
- Poor existing data structures remain poor data structures.
- Input/output translation usually produces hard to read "unnatural" code in the new language.
- Translation does not take advantage of the code and data packaging techniques available in the newer languages. Attempts to automatically translate some FORTRAN programs to Ada have failed.

Reengineering is the combination of "reverse engineering" a working software system and then "forward engineering" a new system based on the results of the reverse engineering. Forward engineering is the standard process of generating software from "scratch." It is composed of the life cycle phases such as requirements, architectural design, detailed design, code development, testing, etc. In each phase, certain products are required and the activities which produce them are defined. Each product is required to be complete and consistent. To progress forward to a new phase normally requires a new representation of the products which involve more detail such as new derived requirements, design decisions, trade off evaluation between alternative approaches, etc. Finally, code is developed which is the most complete, consistent, and detailed representation of the required product.

Reverse engineering is the reverse of forward engineering. It is the process of starting with existing code and going backward through the software development life cycle. Life cycle products are, therefore, obtained by abstracting from more detailed representations to more abstract ones. This process should proceed much faster than forward engineering since all of the details required are available. Reverse engineering starts with the most detailed representation, which has also proven to be complete and consistent since it can currently do the job required. Developing products in reverse involves abstracting out only the essential information and hiding the non-essential details at each reverse step.

How far to go backward in the reverse engineering process before it is stopped and forward engineering begins is a critical question and involves trade offs. It is important to understand all of what the program
does, all of the information it handles, and the control flow since these are probably required to get the job
done. This implies taking the reverse process far enough to understand what the "as is" program is. This is
usually more significant than how the program does its job since the how is usually the part that will be
changed in any following forward engineering process.

What a program does is called its requirements. How it meets those requirements is its design. For a
reverse engineered program it is the design that will be updated more often than what the program will do.
Modern software engineering techniques and technologies such as user interfaces, database management,
memory utilization, data structuring, packages, objects, etc. will affect the design, not what the program
does. Therefore, once it is understood what the program does and what is obsolete, then the forward
engineering process can begin with confidence.

Reverse engineering is referred to as "design recovery" when the reverse engineering process stops at the
recovery of the design of the implementation, rather than proceeding on to a higher level of abstraction to
include the recovery of the requirements. The basic process of this level of design recovery involves
recovery of information about the code modules and the data structures in an existing program. This
information will support the programmer/analyst who is maintaining an unfamiliar large FORTRAN
program, upgrading it for maintainability, or converting it to another target language.

However, a better job of redesigning a program can be accomplished with requirements recovery than with
design recovery. To carry the reverse engineering process beyond design recovery to requirements recovery
is difficult and requires higher levels of domain knowledge to do the abstractions. The whys of the
requirements, design, and implementation can only be provided by someone very familiar with the program
and the domain. This level of expertise is often very difficult to find and have dedicated to the reengineering
process. For this reason, the methods and tools that the STB has developed initially assume reverse
engineering only to the design recovery stage. Future development will be based on feedback from the JSC
software engineering community. The current standards, methods, tools, and environment are all designed
to be sufficiently flexible and extendible to enable the strategies to be extended to cover the full spectrum of
reverse engineering.

The overriding philosophy of this planned reverse engineering process is to capture the total software
implementation in an electronic form. This includes source code, documentation, databases, etc. Figure 2
illustrates the progression of data structures from COMGEN-compatible code (see section "Software
Technology Branch's Reengineering History") to reengineered code. This progression in electronic form
ensures that the total consistent and complete requirements representation is available. Software tools are
provided to support the generation of the more abstract products required for engineering in reverse as well
as capturing rationale and decisions of the engineer. By the continuing process of abstracting the
information about the program into the different representations, the engineer can remain more confident
that information is not being lost or inadvertently "falling through the cracks."
SOFTWARE TECHNOLOGY BRANCH'S REENGINEERING HISTORY

In the early 1970's, the Mission Planning and Analysis Division's (MPAD) Software Development Branch and TRW/Houston developed a tool, called COMGEN, that began as a COMMON block specification statement generator. It grew to include many other functions as new techniques were developed. Later COMGEN was broken up into a continually evolving set of tools with common data interface structures. This tool set supports the maintenance of FORTRAN programs today on Unisys and multiple Unix systems. People still refer to this tool set as COMGEN tools, and a program that complies with the MPAD standard COMMON concept as a COMGEN-compatible program. [1,2,3]

In the 1970's, MPAD performed a lot of software reengineering to meet the goal of combining many of the independently developed engineering programs, each with its own input/output formats. Many of the modern concepts such as separation of input/output processing from the applications, databases, data structures, packages, generics, objects, etc. were recognized and simulated to some degree. They were not called by the modern names, of course, but the design engineers were trying to do good engineering, modularization, and data handling. Even though these techniques were known in the 1970's, they are just now really becoming popular because of newer technologies such as database management systems, user interface tools sets, and modern languages that actually embed and enforce good software engineering practices.

In the late 1980's, some of the personnel and the functions of the Software Development Branch were reorganized into the newly created Software Technology Branch (STB). The STB's reengineering history has put JSC in a better position with respect to the maintainability of its older software than many other organizations. The positive results of this experience include the following:

- Most of the software is reasonably modular.
- The data has some structure.
- Most of the software at JSC is reasonably compatible with the STB's tools, including the in-line documentation.
- The large complex programs that support many simulations have considerable software reuse and information sharing.
MAINTENANCE STRATEGIES

The strategies presented in this document are intended to help with design recovery in support of programmer/analysts who are required to maintain large FORTRAN programs that they did not develop. In addition, these strategies are intended to support reengineering of existing FORTRAN code into modern software engineering structures, which are then easier to maintain and which allow a fairly straightforward translation into other target languages. The STB is proposing standards, methods, and an integrated software environment based upon the significant set of tools built to develop and maintain FORTRAN code for the Space Shuttle. \[4,5,6,7,8\] The environment will support these structures and practices even in areas where the language definition and compilers do not enforce good software engineering practices.

New FORTRAN Standards

New standards, which allow modern software engineering constructs to be used in FORTRAN 77, have been defined by the STB. \[5\] These standards are added to existing standards defined by the former MPAD and still in use in the mission planning and analysis domain. The goal of the new standards is to improve maintainability and permit relatively automated translations to newer languages. In table 1, the standards and their benefits are summarized. These standards address documentation, longer variable names, modern control flow structures, grouping subprograms together as virtual packages, data structuring, and input/output encapsulation in separate subprograms. Where FORTRAN 77 does not provide the constructs, virtual constructs are provided along with a tool environment to support their development and maintenance. The existing core of FORTRAN programmers should have little problem with the standards and new FORTRAN code should adhere to them from the start.

Table 1. Standards Summary

<table>
<thead>
<tr>
<th>Standard</th>
<th>Benefit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Documentation</td>
<td></td>
</tr>
<tr>
<td>Header statement before code blocks</td>
<td>Understandability</td>
</tr>
<tr>
<td>Requirements in CD1 statements</td>
<td>Understandability and traceability</td>
</tr>
<tr>
<td>Rationale in CD7 statements</td>
<td>Design knowledge capture</td>
</tr>
<tr>
<td>Virtual package identification</td>
<td>Maintenance</td>
</tr>
<tr>
<td>Longer, more meaningful variable names</td>
<td>Understandability</td>
</tr>
<tr>
<td>Modern control flow structures</td>
<td></td>
</tr>
<tr>
<td>Block DO</td>
<td>Maintenance and understandability</td>
</tr>
<tr>
<td>DO WHILE</td>
<td></td>
</tr>
<tr>
<td>Grouping subprograms into virtual packages</td>
<td>Higher level of abstraction, understandability</td>
</tr>
<tr>
<td>Data structuring</td>
<td></td>
</tr>
<tr>
<td>Preferred use of calling parameters</td>
<td>Maintenance</td>
</tr>
<tr>
<td>Controlled use of COMMON blocks</td>
<td>Maintenance</td>
</tr>
<tr>
<td>INCLUDE</td>
<td>COMMON database concept</td>
</tr>
<tr>
<td>Preferably encapsulate input/output in separate subprograms</td>
<td>Maintenance and support to future conversions</td>
</tr>
</tbody>
</table>

Design Recovery and Reengineering Methodology

The reengineering methodology defines the steps, the skills required, and guidelines on how far to reverse engineer before deciding to rebuild. The key goal is to update to modern technology and software engineering concepts without losing required functions and data. Methods are provided that have the flexibility to meet multiple levels of conversion, each of which improves maintainability. Figure 3
illu...
Figure 4. Conceptual Architecture of the Design Recovery and Reengineering Environment

The environment will not be a completely automated environment since much work will still have to be done by a programmer/analyst. A person must be in the loop to provide the required puzzle-solving skills that are beyond the capabilities of state-of-the-practice tools. However, as an experience base is accrued in design recovery and reengineering, knowledge-based capabilities can be added to the environment.

Version 1 of the environment called REengineering APplications (REAP) was delivered in June, 1991. This integrated all existing JSC supported tools listed above, behind a common user interface built on the MOTIF standard. It contains major elements of all subsystems and encapsulates the capabilities that have been developed and used at JSC during the last fifteen years. A version with improved tool integration, user interface enhancements, and the commercial LOGISCOPE tool was delivered in October, 1991. The Fortran design recovery version should be available in February, 1992. MCC should also have delivered an evaluation prototype of a design recovery capability for the C language by that time. In parallel, the study of using CASE framework standards and tools to better integrate and manage this environment should be completed early in 1992 and the version 2 series will be delivered on one of these platforms. The plans and design of REAP are such, that all deliveries containing COTS products will be tailorable so that users can delete the COTS tools that they do not want to license. This policy even includes the framework integration tools. In most cases, similar functions might still be available but they would have less capability.

CONCLUSIONS

JSC has a large amount of existing code in FORTRAN that embodies domain knowledge and required functionality. This code must be maintained and eventually translated to more modern languages. Three primary alternative solutions have been identified to address the maintenance problems of these old FORTRAN programs: complete redevelopment of the programs, code translation to a more modern language or version of a language, and reengineering. Complete redevelopment is effective but very costly. Simple code translation is cheap, but usually ineffective since seldom do the old systems incorporate modern software engineering concepts such as good data structuring, good control structuring, packages, objects, etc., that should be present in the new system. Modern languages such as Ada have constructs for representing these features, but translators cannot determine these features in the original code to map them into the new system. Reengineering is being recognized as a viable option because the old systems, in
spite of obsolete technology, do contain all of the required functionality and can get the job done. However, at the present time there are only a few expensive Computer Aided Software Engineering (CASE) tools and no total system environment available in the COTS market to support reengineering FORTRAN programs.

The STB maintenance strategies provide standards, methods, and a tool environment for upgrading current FORTRAN systems without losing the embedded engineering knowledge and at a lower cost than for complete redevelopment of the program. A useful environment for reengineering FORTRAN software can be built fairly quickly by building upon the existing FORTRAN development and maintenance tools, COTS products, new software and hardware technologies, plus current research into reuse, design recovery, and reengineering. This environment will support reengineering existing FORTRAN code into more maintainable forms that can also be readily translated into a modern language including newer versions of FORTRAN.

Two versions of the environment were delivered in 1991 which integrate the existing JSC tools plus the commercial LOGISCOPE tool behind a common MOTIF user interface. A Fortran design recovery capability should be available in February, 1992 and the MCC should deliver a design recovery prototype for the evaluation of design recovery in the C language by that time. Plans are to integrate this capability on a CASE framework tool during 1992.

GLOSSARY

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>arbitrary FORTRAN</td>
<td>FORTRAN program that is not compatible with the COMGEN standards long in place for JSC's mission planning and analysis domain.</td>
</tr>
<tr>
<td>COMGEN-compatible</td>
<td>FORTRAN program that is compatible with the COMGEN standards long in place for JSC's mission planning and analysis domain. [1]</td>
</tr>
<tr>
<td>COTS</td>
<td>Commercial-Off-The-Shelf</td>
</tr>
<tr>
<td>design recovery</td>
<td>Reverse engineering, the first step for maintenance or reengineering.</td>
</tr>
<tr>
<td>environment</td>
<td>Instantiation of a framework, i.e., an integrated collection of tools. It may support one or more methodologies and may also provide a framework for third party tools.</td>
</tr>
<tr>
<td>framework</td>
<td>Software system to integrate both the data and the control of new and existing tools; usual components include a user interface, object management system, and a tool set.</td>
</tr>
<tr>
<td>FORTRAN 77</td>
<td>ANSI standards for FORTRAN in effect in June 1990.</td>
</tr>
<tr>
<td>FORTRAN 8x</td>
<td>Future ANSI standards for FORTRAN; expected to be approved and released soon; draft standards have been circulated; unofficially called FORTRAN 90.</td>
</tr>
<tr>
<td>forward engineering</td>
<td>Process of developing software from &quot;scratch,&quot; through the phases of requirements, design, and coding.</td>
</tr>
<tr>
<td>package</td>
<td>&quot;A collection of logically related entities or computational resources&quot; (Booch[9]).</td>
</tr>
<tr>
<td>reengineering</td>
<td>&quot;The examination and alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form&quot; (Chikofsky and Cross [10]); combination of reverse engineering and forward engineering.</td>
</tr>
<tr>
<td>reverse engineering</td>
<td>&quot;The process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction&quot; (Chikofsky and Cross [10]); the first step of maintenance or reengineering; reverse of forward engineering; process of</td>
</tr>
</tbody>
</table>
starting with existing code and going backward through the software development life cycle.

**software maintenance** Process of modifying existing operational software while leaving its primary functions intact (Boehm [11]).

**subject program** Program that is being maintained or reengineered.

**virtual package** Package concept as defined by Booch [9], but implemented either in Ada, which enforces the concept, or in a language in which the concept must be supported procedurally.

**REFERENCES**


