REENGINEERING LEGACY SOFTWARE TO OBJECT-ORIENTED SYSTEMS

S23-61
44829

C. Pitman, D. Braley, E. Fridge
NASA/JSC/PT4
Houston, TX 77058

A. Plumb
LinCom, Inc.
Houston, TX 77058

M. Izygon, B. Mears
I-NET, Inc.
Houston, TX 77058

ABSTRACT

NASA has a legacy of complex software systems that are becoming increasingly expensive to maintain. Reengineering is one approach to modernizing these systems. Object-oriented technology, other modern software engineering principles, and automated tools can be used to reengineer the systems and will help to keep maintenance costs of the modernized systems down. The Software Technology Branch at the NASA/Johnson Space Center has been developing and testing reengineering methods and tools for several years. The Software Technology Branch is currently providing training and consulting support to several large reengineering projects at JSC, including the Reusable Objects Software Environment (ROSE) project, which is reengineering the flight analysis and design system (over 2 million lines of FORTRAN code) into object-oriented C++. Many important lessons have been learned during the past years; one of these is that the design must never be allowed to diverge from the code during maintenance and enhancement. Future work on open, integrated environments to support reengineering is being actively planned.

1. INTRODUCTION

Like many other organizations, NASA has a legacy of complex software systems that were developed during the past three decades and are still being used today. These systems are critical to NASA's mission; they represent an enormous investment; and they are very expensive to use, maintain, and enhance. The aerospace community can continue to maintain the systems, using improved tools and techniques, but at some point that will become ineffective: the skills for old languages will be harder to find; the few people who understand how to maintain the systems will retire; or the number of patches on top of patches will drive maintenance costs beyond all reason.

The systems could be scrapped and redeveloped, but it would take enormous amounts of time and money to redevelop them from first principles, because of the vast quantity of software involved, because the necessary engineering skills would have to be found and reassigned to the tasks, etc. More importantly, the systems themselves may be the only repository for detailed engineering knowledge of the application at the algorithm level, because system requirements documents are not kept up-to-date by most organizations as new requirements are continually added. If the current systems were scrapped, all of this engineering knowledge of the current algorithms and interfaces would be lost, and for mission-critical systems this knowledge was gained at great expense.

Another option is software reengineering, i.e., modernize the systems using up-to-date software technologies, in order to improve maintainability, without losing the embedded engineering knowledge or current capabilities. However, software reengineering itself is not a trivial task. It is a young field, and organizations need support (training, tools, etc.) in order to realize benefits. In particular, methods and tools to help reengineer software systems are becoming increasingly important.

2. REENGINEERING TERMS

Reengineering terminology is not always used in the same way by different authors, and so it is best to begin with brief definitions of some terms (Figure 1). Forward engineering refers to the usual direction of software development (whether a waterfall, spiral, incremental, or rapid prototyping process): begin with requirements, then design , code, test, and deliver. In contrast, reverse engineering refers to starting with the old code and recovering from it the essential semantics, the design, and/or the requirements. Reengineering is NOT synonymous with reverse engineering. Reengineering is the combination of reverse engineering followed by forward engineering into the new, modernized software system. (Actually, this definition is itself oversimplified: in many reengineering processes, the reverse engineering is not done completely in a single phase up front, but is done as needed and fed back into the forward engineering effort at various times.) Note that reengineering that goes all the way back to recover requirements is very similar to rebuilding the software system from scratch, but the difference is that the reverse engineering portion of the reengineering process helps to assure that all of the critical requirements which are captured only in the old code will appear in the new system.

3. OBJECT-ORIENTED TECHNOLOGY

Object-oriented methods provide a modern approach to building software. There are many methods available on the market today, such as those by Rumbaugh, Booch, Shlaer-Mellor, or Wirfs-Brock. Some-second generation methods, such as Fusion, are also beginning to emerge.

Probably, the most widely used method today is the Object Modeling Technique (OMT) by James Rumbaugh, et al. [1] OMT provides a method for incorporating good software engineering principles during the requirements analysis and design phases. Many of these principles, such as data abstraction, encapsulation and modularization, commonality, and reuse, have been discussed for many years now, but are still all too often ignored when the pressures of project deadlines loom close at hand.

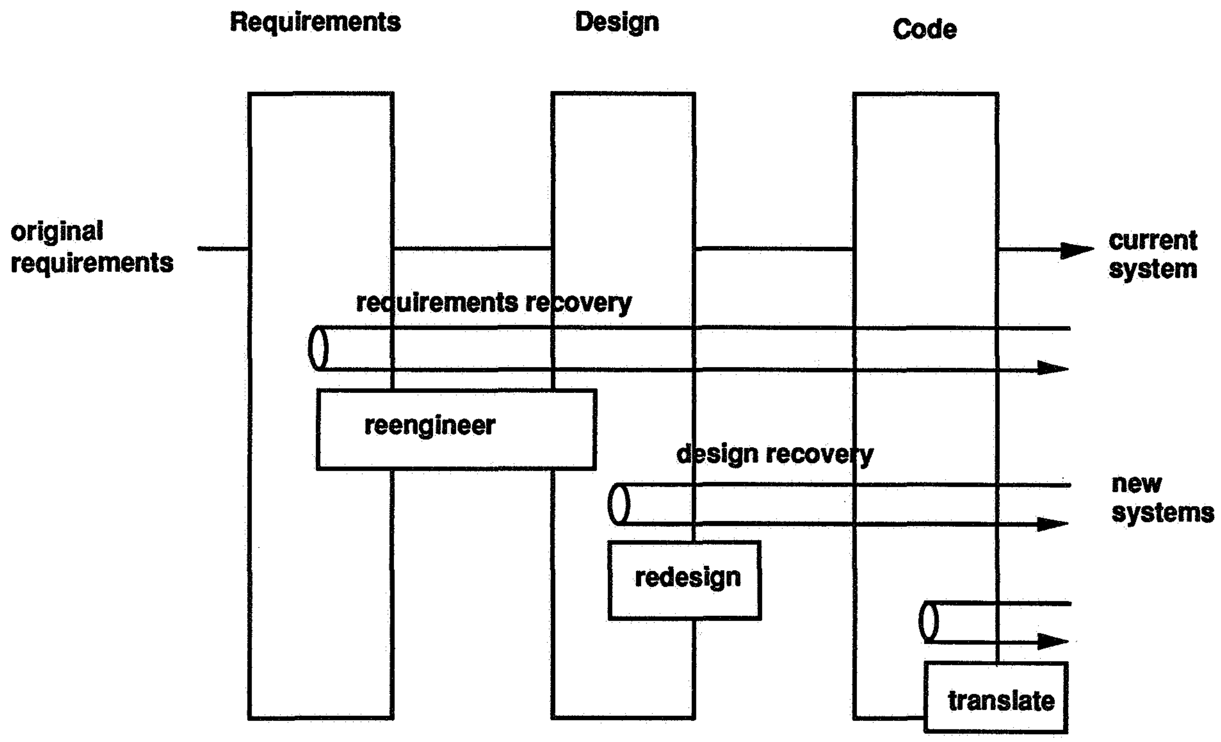

Figure 1. Reengineering is the combination of reverse engineering (analyzing the current code to recover its meaning, design and/or requirements) followed by forward engineering into the new, modernized software system. Three levels of reengineering are shown: translation, redesign, and complete reengineering.

OMT also helps to smooth the transition between life cycle phases. For example, OMT models provide a standard language that can be used for defining guidelines for transforming designs into code; indeed, some tools exist today that can automatically generate C++ "header" files directly from the OMT models.

Perhaps most importantly, large software engineering projects that are done by teams are inherently a social enterprise, and OMT models are an extremely effective communications tool for software development or reengineering teams.

4. TECHNOLOGY TRANSFER SERVICES

The Software Technology Branch (STB) helps teams to learn, adopt, and use reengineering and object technology. The STB has led seminars on reengineering and on OMT, and is providing mentoring and consulting support to some JSC reengineering projects. The STB also performs evaluations of Computer-Aided Software Engineering (CASE) tools for these projects, and has developed some evaluation criteria as a part of these efforts. The STB in turn gains from these experiences because they also offer opportunities for testing and refinement of reengineering methods on real applications.

5. REENGINEERING EXPERIENCES

The Software Technology Branch has been developing and testing reengineering methods and tools for several years. [2] The first project involved a large orbital mechanics program named the Orbital Maneuver Processor (OMP). The project recovered the design of the OMP FORTRAN code, modified the design into an object-based one, and then implemented the new design in Ada. [3, 4] Several tools that aid in the recovery of the design from FORTRAN code (e.g., COMMON block structures, calling trees, ...) were used and enhanced for this and following projects; these tools were originally developed by JSC during the period of development of the Space Shuttle flight planning software.

Work on support environments for reengineering has resulted in the Reengineering Applications (REAP) environment, which provides a uniform presentation and invocation of reengineering tools and a suggested sequence for their use. As mentioned above, the STB is providing reengineering support to other projects: a command system (Mission Operations Computer), a database reconfiguration system (Reconfiguration Tools), and a solar thermal analysis and optimization system (in conjunction with the University of Houston, for Sandia National Laboratories).

6. REENGINEERING SUPPORT FOR THE ROSE PROJECT

Another of the more significant reengineering projects with which the STB is associated is the Reusable Objects Software Environment (ROSE). This project is seeking to recover the requirements from the flight analysis and design system (over 2 million lines of FORTRAN code), enhance the requirements for reuse if necessary, and then redesign the system into an object-oriented one and implement it in C++. The first-year phase of this four-year project will be concluded in February, 1994; so far, very promising results have been obtained.

On the ROSE project, the reverse engineering method used for FORTRAN design recovery was developed over the course of a few years and then tailored to the ROSE project. Several Computer-Aided Software Engineering (CASE) tools are used to facilitate this process, such as data and control structure analysis tools, complexity metrics tools, and restructuring/refining tools (to a limited extent). The forward engineering part of the ROSE reengineering project is using the Object Modeling Technique (OMT), an Object-Oriented Analysis and Design (OOAD) method developed by James Rumbaugh, et al. OMT is excellent for Analysis, providing the basis for communication and clarification of the problem to be solved and of the requirements. (Section 3)

7. FILLING THE GAPS IN COMMERCIAL TOOLS

Sometimes commercially available tools do not support all of the steps in an organization's software development or reengineering process; this leaves holes or "gaps" in the process where custom-built tools might be needed. Usually, these gaps occur because many organizations' processes are more complex than those found in text books, and it is the text book processes that are most often supported by commercial tools.

Custom-built "gap" tools make sense when all of the following conditions occur:
(i) available tools do not support a specific part of the organization's process;
(ii) the gap tools will be used to automate process steps that would otherwise be performed manually;
(iii) the gap tools must save more time than they take to build and maintain; and
(iv) there should be easy, stable methods for interfacing to the commercial tools used by the organization.

For example (Figure 2), the ROSE project is reengineering a large FORTRAN system. The project has selected Reasoning Systems' Refine/FORTRAN as a FORTRAN front end, working with Software through Pictures (STP). However, Refine/FORTRAN does not completely handle the COMMON formats that ROSE has encountered, and one of the JSC Maintenance Aids, CREATE, was designed precisely for that type of COMMON format. The STB built a gap tool to integrate this information into the Refine/FORTRAN - STP communications, and thus to correctly show the internals of the COMMON blocks.

8. INTEGRATION TECHNOLOGY

As one of its more forward looking research projects, the STB has also begun investigating open, integrated reengineering environments. (Figure 3) Today, many companies are working on integrated support environments for forward engineering, but not too much work has been done on reengineering environments, even though the requirements are very similar. An integrated environment should provide presentation, data, control, and process integration. The National

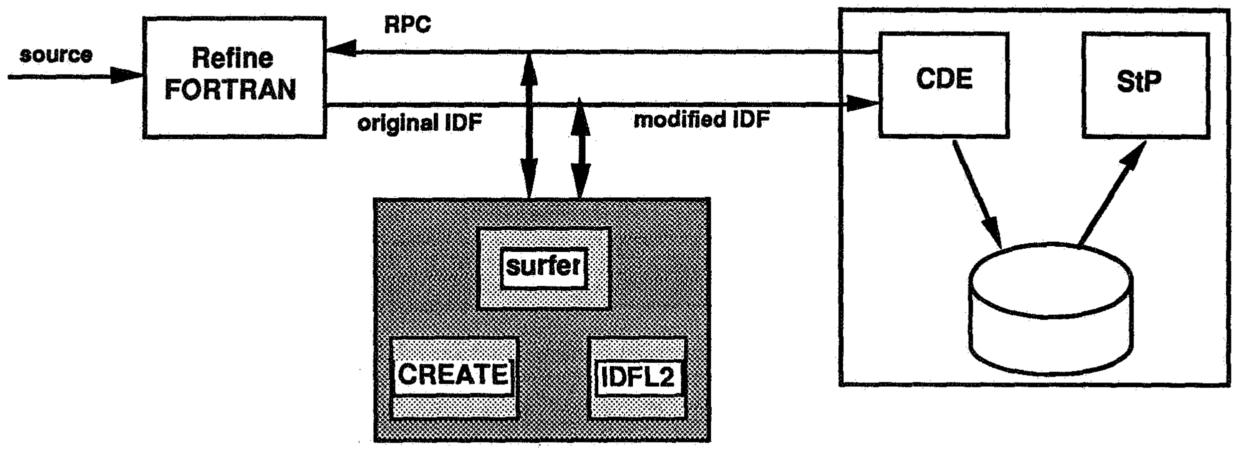

Figure 2. Custom-built "gap" tools (in the shaded area above) can be used to fill gaps, i.e., steps in an organization's software development or reengineering process where commercially-available tools do not provide automated support.

Institute of Standards and Technology (NIST) and the European Computer Manufacturers Association (ECMA) have proposed a standard model for discussing such an integrated environment. The STB, in conjunction with IBM, has worked to develop a reengineering data model for a repository, which would provide data integration in a reengineering environment based on the NIST/ECMA model. Much good initial work has been done to date, but this is a challenging problem and much work still remains. Nevertheless, the objective is to produce the REAP II environment, based on this integration work, within one to three years.

9. LESSONS LEARNED

One of the most important lessons learned over the past two decades of software development and maintenance at JSC is that the design must never be allowed to diverge from the code during maintenance and enhancement; i.e., the design must be kept closely tied to the code. For the ROSE project, it was discovered that, during design, OMT must be supplemented and extended to permit a closer tie between the design and the C++ code that will eventually implement it.

Another very important lesson learned is that it is critical to choose tools that fit the processes and data types of an organization and project, and not vice versa. All too often an organization will buy a tool and try to force-fit it into its processes, without considering the way it does business, much less if the processes themselves might need reengineering.

Finally, an important lesson learned is that it is critical to provide appropriate training in the methods, processes, tools, and language used for a project, BEFORE the project team members are expected to meet project deadlines and deliverables. The ROSE project adopted this approach and it is paying off. The STB was asked to help coordinate the ROSE training and to provide initial training in the OMT method.

10. CONCLUSIONS

The STB has been researching and developing reengineering of legacy software systems for many years, and it has pulled together some methods, training, and tool products that can greatly facilitate the tasks associated with reengineering legacy software. These products and services are proving useful for JSC software systems, and can be useful for reengineering software in other scientific and technical domains as well. The STB is committed to applying and transferring this

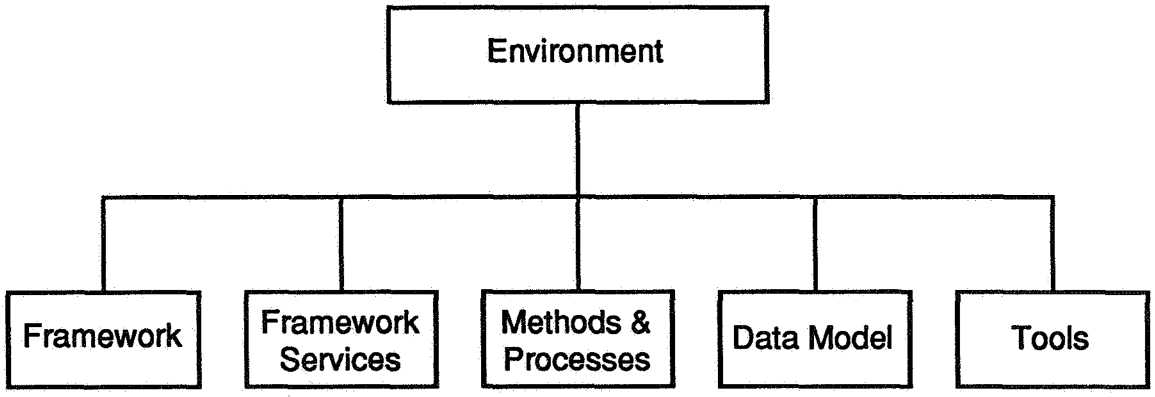

Figure 3. An environment to support software reengineering is composed of many parts. A framework is the backbone upon which an environment is built, and it supplies services like data repository services. Methods and processes for doing reengineering should drive the environment's configuration, and the data model, stored in the framework's repository, contains required information about the processes and tools. Tools automate some of the reengineering work.

reengineering technology to other projects, including those in industry. Please contact us if you have projects where this technology might help.

11. REFERENCES

1. Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorensen, W., *Object-Oriented Modeling and Design,* Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

2. Pitman, C. L., Erb, D. M., Izygon, M. E., Fridge III, E. M., Roush, G. B., Braley, D. M., and Savely, R. T., "The Development and Technology Transfer of Software Engineering Technology at Johnson Space Center", *Fifth Annual Workshop on Space Operations Automation and Robotics (SOAR ' 91),* Houston, Texas, July 9-11,1991.

3. Fridge III, E., Braley, D., & Plumb, A., "Maintenance Strategies for Design Recovery and Reengineering," Vols. 1-4, NASA Johnson Space Center, Houston, TX, June, 1990.

4. Plumb, A., & George, V., "A Method for Conversion of FORTRAN Programs," Barrios Technology, Inc., Houston, TX, March, 1990.