

Hybrid Re-engineering

Title: Hybrid Re-engineering
Presenter(s): Linda H. Rosenberg, Lawrence E. Hyatt
Track: Track 10
Day: Thursday
Keywords: Re-engineering, reverse engineering, forward engineering, risks, COTS, metrics
Abstract: Traditional re-engineering applies reverse engineering to existing system code to extract design and requirements. Forward engineering is then used to develop the replacement system. Due to limited resources, many organizations, NASA included, are looking at the use of COTS software packages as a means of decreasing development time and costs. This paper briefly describes traditional re-engineering then discusses the emerging process of hybrid re-engineering. Hybrid re-engineering uses a combination of translation of existing code, COTS, and custom code to produce the replacement system. This paper discusses the advantages, disadvantages, potential risks, and metrics for each of these tracks in hybrid re-engineering.

1. Introduction

Re-engineering is the examination, analysis, and alteration of an existing software system to reconstitute it in a new form, and the subsequent implementation of the new form.[7] The process typically encompasses a combination of reverse engineering, re-documentation, restructuring, and forward engineering. The goal is to understand the existing software system components (specification, design, implementation) and then to re-do them to improve the system's functionality, performance, or implementation. The primary risk is that the new target system will not have the same functionality as the existing system and will be lower in quality.[2]

Although objectives of a specific software re-engineering task are determined by the goals of the owners and users of a system, there are two general re-engineering objectives [10]:

1. Improve quality—Typically, the existing software system is of low quality, due to many modifications. User and system documentation is often out-of-date or no longer in existence. Re-engineering is intended to improve software quality and to produce current documentation. Improved quality is needed to increase reliability, to improve maintainability, to reduce the cost of maintenance, and to prepare for functional enhancement. Object-oriented technology may be applied as a means for improving maintainability and reducing costs.
2. Migration—Old working software may still meet users' needs, but it may be based on hardware platforms, operating systems, or languages that have become obsolete and thus may need to be re-engineered, transporting the software to a newer platform or language. Migration may involve extensive redesign if the new supporting platforms and operating systems are very different from the original, such as the move from a mainframe to a network-based computing environment.

This paper first looks at the generally accepted method for re-engineering, then describes an emerging form of re-engineering that the Software Assurance Technology Center (SATC) at NASA Goddard Space Flight Center (GSFC) has given the name “hybrid re-engineering.” After defining three “tracks” used in hybrid re-engineering, this paper discusses each track in depth, looking at potential risks and how software metrics can be used to identify and mitigate these risks during the re-engineering process.

2. General Model For Software Re-engineering

Re-engineering starts with the source code of an existing legacy system and concludes with the testing and implementation of the target system source code. Figure 1 diagrams a general model for software re-engineering that indicates the processes for all levels of re-engineering based on the levels of abstraction used in software development.[3]

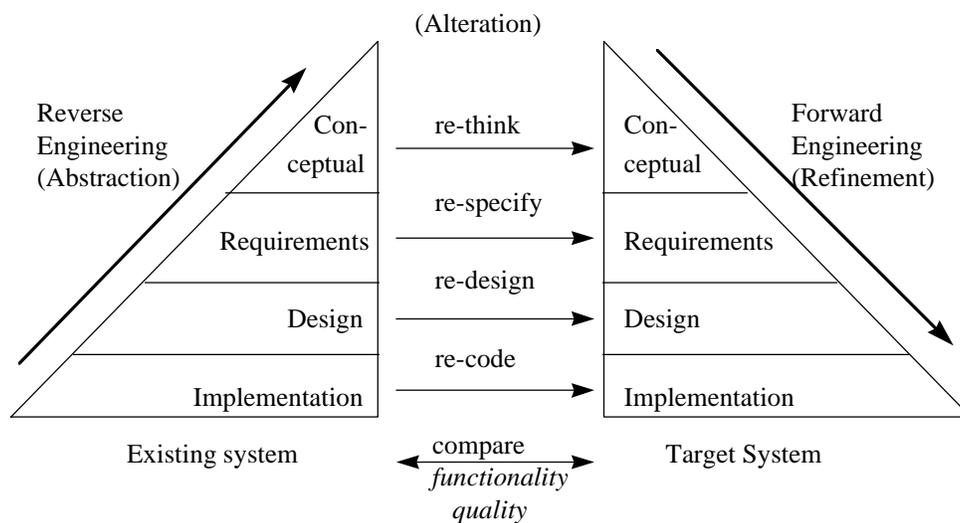


Figure 1: General Model for Software Re-engineering

Re-engineering starts with the code and comprehensively reverse engineers by increasing the level of abstraction as far as needed toward the conceptual level, rethinking and re-evaluating the engineering and requirements of the current code, then forward engineers using a waterfall software development life-cycle to the target system. In re-engineering, industry reports indicate that approximately 60% of the time is spent on the reverse engineering process, while 40% of the time is spent on forward engineering.[9] Upon completion of the target system, most projects must justify their effort, showing that the necessary functionality has been maintained while quality has improved (implying improved reliability and decreased maintenance costs).

Re-engineering projects, like all projects, have limited funds and time and thus need to minimize the re-engineering cost while still maintaining system functionality and improving software quality. Many organizations, NASA included, are investigating the use of reusable software packages as a means of decreasing development time and costs and reducing development risks. Because the majority of reusable software packages are Commercial Off The Shelf (COTS), the term COTS will be used in this paper to represent this category of software.

In trying to decrease the cost of re-engineering, managers are considering the industry “rule of thumb” that 20% of the software causes 80% of the errors [4], and that approximately 15% of the system contains 85% of the functionality.[2] They are questioning the need to apply all re-engineering phases to the entire system, as opposed to just the portion of the code causing most of the problems or containing most of the critical functionality. This thought process, combined with time and budget constraints, is leading to a new form of re-engineering, one that applies different levels of abstraction in reverse engineering (and different methods of alteration) to selected sections of existing code.

3. Hybrid Re-engineering

The SATC is working with NASA re-engineering projects to identify and minimize software development risks while improving the quality of products. These projects are applying a combination of abstraction levels based on the needs of the project and its budget and schedule. The SATC has coined the phrase “hybrid re-engineering” to indicate a combination of abstraction levels and alteration methods used to transition an existing system to a target system.

Figure 2 is a diagram of a “typical” software system that has been in use for some time. In this re-engineering example, we assume the project is moving from FORTRAN to C++ (but not necessarily to an object-oriented design).

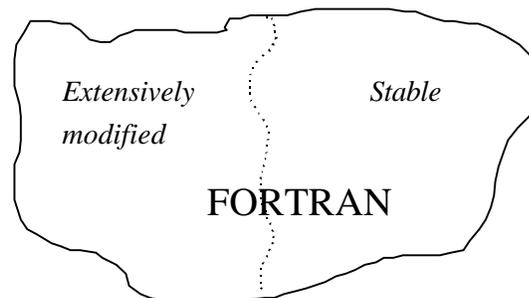


Figure 2: Current Software System

In looking at the current software system, the manager sees two classifications of code: some stable code that has had minimal modifications and whose requirements have not changed, and some code that has undergone multiple changes and has become unstable, unreliable, and costly to maintain. Re-engineering the stable code may not require total reverse engineering—tracking back to the conceptual level and rethinking the requirements and then forward through all stages. It might be feasible to simply re-code this portion into the new language or the new environment.

Because, as stated previously, reverse engineering constitutes 60% of total re-engineering costs, if some portion of these tasks can be omitted, a savings in time and cost results. Recognizing the importance of savings, a new re-engineering structure emerges as shown in Figure 3. This is an adaptation of the General Model for Software Re-engineering shown in Figure 1.

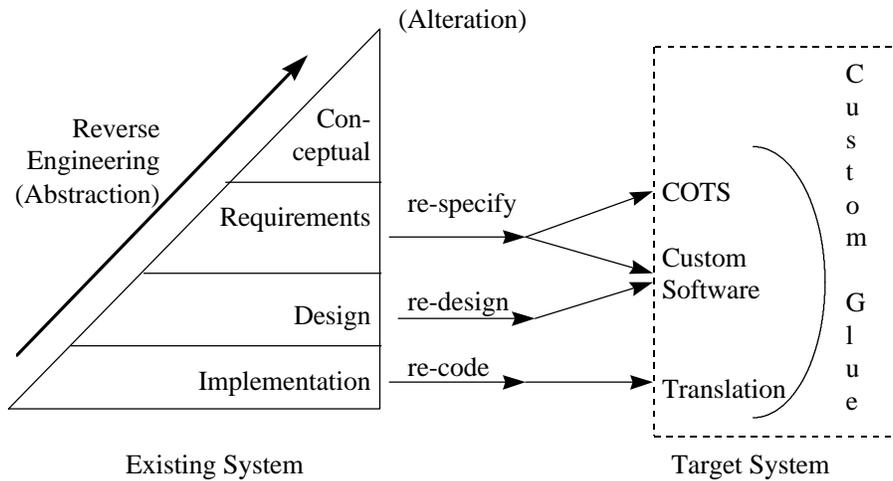


Figure 3: Hybrid Re-engineering Tracks

Hybrid re-engineering requires an approach similar to that of traditional re-engineering, but with additional considerations. When starting to re-engineer, initial justifications for re-engineering—such as costs and quality—are developed and expectations—such as return on investment—are stated. An analysis of the legacy system should be done to determine the feasibility of hybrid re-engineering. The analysis of the legacy system should provide a guideline to identify optimal strategies (translation, COTS, and custom) and project the cost of the target system. Once the decision to use a hybrid re-engineering approach is made, additional analysis is needed.

The first step in a hybrid re-engineering approach is to investigate the requirements and constraints of the software system and of the re-engineering project. Project factors include the time available for reverse and forward engineering. Time must be built in to investigate available COTS, including hands-on testing of the COTS. While forward engineering development time should decrease with the use of COTS and the translation of code, additional time will be needed to test the integration and interface of the products of the three tracks. Budget constraints must also be considered—how much can be spent on COTS that provide required features versus those that provide desired features. Management-mandated and organizational needs must also be identified. As the three tracks are developed, tradeoffs will be necessary, so it is important to prioritize requirements.

The next step is to do an in-depth analysis of the legacy system, focusing on functionalities and code segments suitable for each of the three tracks (Translation, Custom, COTS). In generic re-engineering, an analysis of the existing system is usually done to provide an evaluation of its quality and maintenance costs. This information is used to justify the costs and improvements at the conclusion of the re-engineering effort. While these reasons are still relevant in hybrid re-engineering, additional features of the legacy system must now be investigated. During the assessment of the legacy system, code sections that implement related functions must be identified. These sections must be further assessed to determine what documentation is available to identify required features versus what is no longer needed or what users have become accustomed to. Code sections must be ordered by the cost of maintenance and the quality of the current structure. Functions that are unique to this project, and thus very unlikely to be replaceable by COTS, must be identified. All this information will be used to identify which hybrid re-engineering track will be applied to the code section.

The process of selecting the track for each code section is described below in the paragraphs pertaining to that track. Once the functionality has been distributed, each track can proceed independently. The schedule for track completion will differ based on the complexity of the tasks in the track. As the tracks yield parts of the deliverable system, the merging of these products can begin. For example, custom glue needed to integrate COTS with translated and custom code can be started once the COTS packages have been selected. User training on the COTS packages can also begin.

Any re-engineering project has inherent risks such as schedule, functionality, cost, and quality. Hybrid re-engineering was developed to decrease some of these risks, because COTS packages are expected to have high reliability and require minimal development time, and translated code should consume fewer resources than custom development.

Because hybrid re-engineering combines three distinct re-engineering efforts, the risks generally associated with re-engineering can be increased by the addition of the risks inherent to each track. Because hybrid re-engineering combines products from different development tracks (COTS, custom software, and translated software), an example of a new risk is the interface and interoperability of the products. For example, data transfer between products can cause compatibility and timing problems. Another new risk is that COTS packages may not work exactly as anticipated.

Metrics can be used by management to improve its insight into software development efforts and to minimize risks. Metrics can indicate how well a project is meeting its goals.[5] In hybrid re-engineering, metrics can also support the justification for decisions on track selection for different software functions and components.

The following sections describe each of the three hybrid re-engineering tracks. After each track is described, the risks associated with the track are identified and appropriate metrics are defined.

3.1. COTS Track Hybrid Re-engineering

In the COTS track of hybrid re-engineering, shown in Figure 4, requirements and functions must be identified that can feasibly be implemented using COTS.

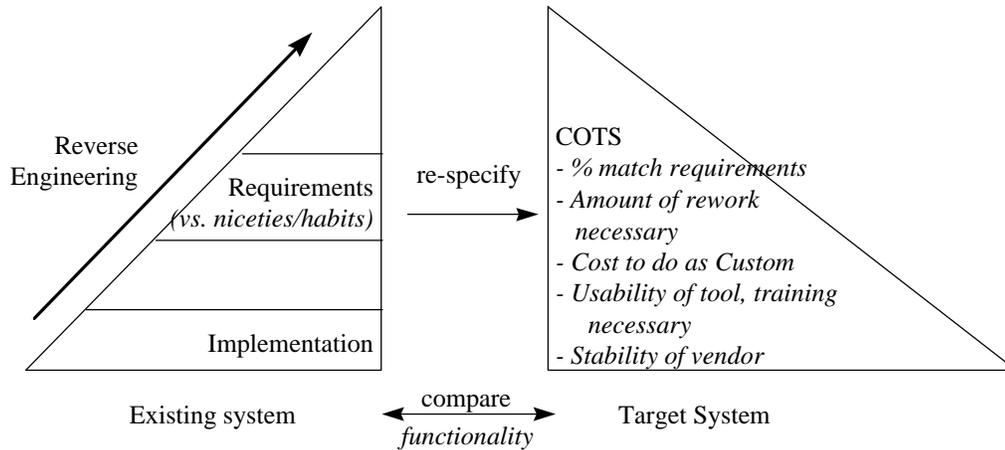


Figure 4: COTS Track Hybrid Re-engineering

After progressing through the reverse re-engineering abstraction levels (Figure 3) to identify requirements, it is important to separate those requirements that must be contained in the target system from those requirements that users want in the new system because they are comfortable with those features or because using them has become a habit. This separation of requirements into “necessary” and “nice” is critical to COTS selection. It is important to then identify what percentage of the desired requirements the COTS totally meets and which requirements the package partially meets. This information can then be used to determine how much rework or supplementation the package requires in order to totally fulfill the system requirements. The usability of the COTS must be evaluated to determine the amount of additional training that will be needed. The stability of the COTS vendor must also be considered. After all these evaluations are complete, the cost to develop from scratch should also be estimated, including testing time, and compared to the total costs of the COTS.

The benefits of using COTS are the reduced design and development time and costs. Because COTS products are usually commercially tested products, reliability is generally high and training and documentation are good. An additional cost savings is that COTS maintenance is done by the vendor.

Although the use of COTS software decreases development time and increases reliability, COTS software also introduces additional risks. A major risk is that the package will not perform as anticipated or advertised, or that will be unreliable, immature, or incomplete. The package may also undergo frequent manufacturer version enhancements requiring constant upgrading. In the worst case, changes may alter or remove functions needed for the software system of which the COTS has become a part. The COTS may require modifications or supplementation to match the requirements, causing increases in schedule and decreasing reliability. In addition, the use of a COTS product may limit further enhancements to the system, because changes in the COTS-provided functions may not be possible due to legal or contractual

issues. An additional training cost may be incurred due to unfamiliarity with the COTS. Even as simple a change as new icons may require additional training time.

Two sets of metrics are applicable when evaluating the COTS track. The first set addresses the expected functionality by identifying the percentage of requirements completely satisfied, partially satisfied, and not satisfied. The second set of metrics evaluates the decision “Make vs. Buy.” The functionality of the new system is compared to the dollars expended starting with product evaluation through testing. The savings to the schedule in design and development compared to the time spent in specification and evaluation, modification, and testing must be calculated.

3.2. Translation Track Hybrid Re-engineering

The translation track of hybrid re-engineering applies tools to translate legacy code automatically into the language of the target system. For this track, shown in Figure 5, the code in the existing system that is relatively stable, having had minimal changes to the original design and architecture, must be identified. This can be accomplished by an analysis of the code and through change reports. The quality of this code must then be evaluated to determine if it will be maintainable. Many source code translators are available to support the transport of code from one language or operating system to another.[12]

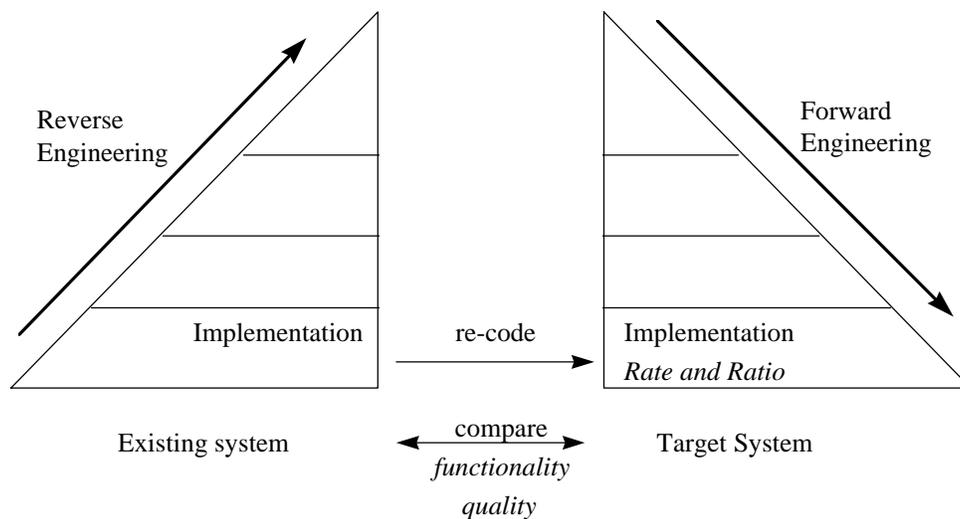


Figure 5: Translation Track Hybrid Re-engineering

An advantage of simply translating code is the decrease in time and cost because minimal, if any, reverse engineering is needed. Maintenance personnel will still understand the program flow because it remains unchanged. This straight translation will also aid programmers in becoming effective in the new language. Savings will also occur in testing because many test cases should be reusable with minimal revisions.

Two risks are associated with the translation track. Source code translators may not be adequate alone, because line-by-line translators don't take advantage of target language semantics, constructs, etc., often resulting in code known as “C-TRAN” (C syntax on FORTRAN structures) or Adabol (Ada and COBOL).[10] Generally, source code translators only succeed in translating 90-95% of the program, and translated code may not produce the correct output, resulting in loss of functionality. The second risk is

in the selection of the code for translation. Poorly designed or poor quality legacy code will result in worse design and quality of the target system.

Prior to re-engineering, in identifying candidates for translation, metrics such as logical and calling complexities provide valuable information on structure and coupling and suggest candidates for translation. In identifying components that have been extensively maintained, change or problem reports supply the data. Tracking the criticality of functions will assist in making tradeoff decisions. A reliability evaluation is needed using the number of errors located per source lines of code. On an on-going basis during translation, this information may be used to determine the success of the translation, e.g., if less than 80% of the code is translating without errors, the tool may not be effective [11]. The size of the legacy system should be compared to the size of the target system and the functionality of the two systems should also be compared.

3.3. Custom Track Hybrid Re-engineering

The custom track of hybrid re-engineering, shown in Figure 6, is similar to traditional re-engineering because new code is derived from the existing legacy system.

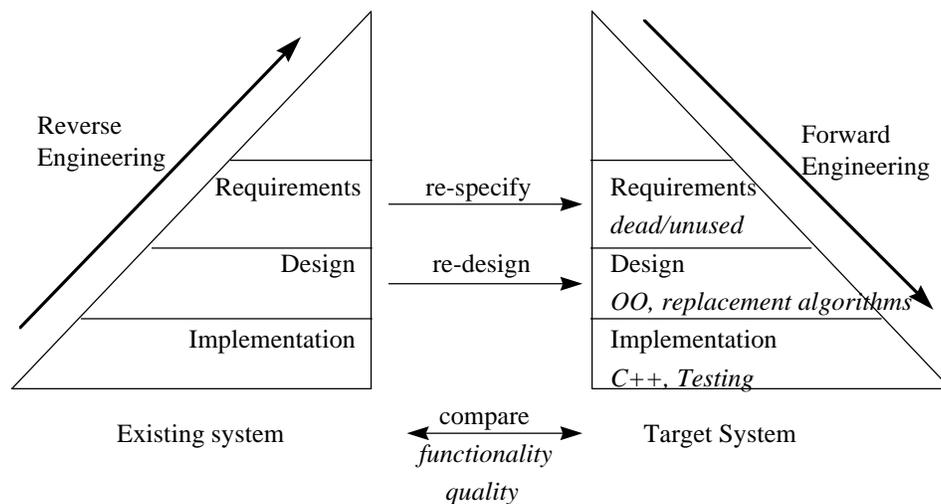


Figure 6: Custom Track Hybrid Re-engineering

In this track, reverse engineering is first performed. Those functions that are not satisfied by COTS packages or through translated code must be identified, and the requirements and design must be extracted. Forward engineering is then performed. This begins with requirements analysis, with the objective of identifying requirements that are not needed. The process is then similar to any development process, beginning with developing a new design, applying object-oriented structure if desired, then implementing the code and doing comprehensive testing.

The benefit of the custom track is that the resulting code should fulfill its requirements exactly. The design should satisfy the new structural requirements, such as moving to an object-oriented environment. This is not the same as adding new functional requirements, but is done in preparation for those enhancements at a later time. The developed code should be of high quality and well structured, requiring little maintenance. Features of the new language should be implemented where possible.

Custom code has the same inherent risks as any software developed: quality, reliability, and schedule. Because most of the functions of the legacy system identified as unique to the system will be done with custom code, the risk is that one of the unique features will not be identified and hence the functionality of the new system will be incomplete. The quality of the code might be poor, resulting in costly maintenance. Features identified as critical to the system that are accomplished with custom code will require extensive testing but reliability might still be poor. As with any development, the risk to schedule and budget remain in this track also.

Metrics for this track are a combination of both process and product metrics. Prior to reverse engineering, the quality of the existing system should be evaluated for later comparison to the target system (as discussed previously). Effort expended should be tracked to assist in the evaluation of the cost to re-engineer. This can also be used to approximate schedule completion using the estimate that 60% of the time is involved in reverse engineering. Once requirements are re-specified, their quality can be evaluated to determine ambiguity and testability.[13] Code analysis tools can be used to evaluate the quality of the code as it is being developed and to identify the risks.[5] In testing, discrepancy or error rates help in evaluating reliability. In this track, both functionality and quality are compared between the existing system and the target system.

3.4. Hybrid Re-engineering Custom Glue

The purpose of the custom glue is to connect the three “systems” resulting from different development tracks. While sounding minor, the success of the project can rest on this development effort. Regardless of how good the products are from the three tracks, if they can’t “talk” to each other and accurately transfer data between themselves at an acceptable speed, the entire project is in jeopardy. This code must be developed with the same precision and rigorous testing as the custom application code discussed previously.

4. Hybrid Re-engineering Track Convergence

Once the system is complete and all tracks are merged, two tasks remain: testing and justification. First, comprehensive system and integration testing must be performed to ensure that all components work together as a cohesive unit and to ensure that all functionality of the existing system was transferred to the new system. Second, justification for the re-engineering is usually required—do the benefits gained justify the cost? Some anticipated benefits, such as improved maintenance and operational costs, can be demonstrated only indirectly through improved quality. Improved quality can be demonstrated initially by a metric analysis of the legacy system compared to the new system. As the new system is put into operation, additional metrics can be used to verify the improvements.

5. Hybrid Re-engineering Benefits

Benefits for hybrid re-engineering are derived both in the application of re-engineering and in the hybrid approach. In general, re-engineering is performed as opposed to building a new system because of the invisible business application procedures and logic that are built into the existing software. These processes might be deeply embedded in business procedures as simple as a field length or as complex as a mathematical algorithm; the only source of this information is in the legacy code. One benefit, therefore, is the capturing of these procedures and their transference to the new system. A second justification for re-engineering versus building is the development and maintenance costs of the legacy system; the time

spent developing logic and components should not be wasted. In re-engineering, the existing system is re-implemented and instilled with good software development methodologies, properties, and new technology while maintaining existing functionality. Reliability and maintainability should also be improved with re-engineering.

Hybrid re-engineering has the additional benefits of a reduced development schedule and hence reduced costs. The development schedule is shortened first by minimizing the amount of reverse engineering (recall, reverse engineering is 60% of the effort). The translation track uses minimal reverse engineering time because work is done at the lowest level. The use of COTS decreases forward engineering development and test time and thus costs. The use of properly selected COTS also increases reliability because these packages have been extensively tested.

6. Hybrid Re-engineering Risks

All software development has inherent risks to schedule and cost. Hybrid re-engineering, as a software development methodology, is also susceptible to them. Hybrid re-engineering, because it is composed of three diverse development tracks, is subject to all the risks discussed in each track description. Also, hybrid re-engineering as a unique software re-engineering methodology has risks to functionality and quality—the functionality of the existing system must be preserved in the new system, and the quality must improve, implying a decrease in operational and maintenance costs. With all the risks in hybrid re-engineering, why bother? Why not just treat it as a new software development effort and omit the re-engineering all together? Because of the benefits.

7. Hybrid Re-engineering Metrics

Metrics, when properly applied, provide managers information about their project to help mitigate risks.[5] It is logical, therefore, to discuss some of the re-engineering phases in which metrics provide valuable information. Previously we have identified metrics applicable for each track in hybrid re-engineering. In this section, we will discuss metrics applicable to the entire project, not just one track. Metrics provide information on quality, functionality, and hybrid track selection, a prime area of risk.

At the start of the re-engineering effort, the legacy system must be quantified. The objective is to identify the amount of functionality and the quality of the existing system. By quantifying the functionality, scheduling estimates are more accurate; during development, completion can be estimated by the percentage of functionality transferred to the new system. Functionality is also important at the conclusion of the project, measuring how much functionality is contained within the new system. The SATC and others are working with function points as a means of estimating functionality. Function points are comparable across languages, and time estimates based on function points are available.[1,6] For COTS packages, functionality might be measured by the number of requirements satisfied.

Quality is harder to measure and few software developers agree totally on the appropriate metrics. The SATC has a group of metrics it applies to projects to evaluate quality. These metrics evaluate the project at the module level (procedure, function, class, or method). Size is measured by counting the number of executable statements. Readability is measured by the comment percentage. Complexity is measured by cyclomatic complexity (McCabe). Coupling is measured by the calling complexity (fan in/fan out).[8] Although there are many other metrics available, these have been successfully applied to many projects at GSFC NASA. One final measure of quality involves the reliability, the number of errors found, and the projected number of errors remaining. These metrics can be used for both the translation track and the

custom code track. When the components are combined, the number of errors found and the projected number of errors remaining can be applied to the whole system.

8. Summary

The number of large systems being built from scratch is diminishing, while the number of legacy systems in use is very high. Rapid changes in the computer industry continually introduce new hardware and software, making older systems obsolescent and difficult to maintain. Businesses do not want to re-develop from scratch; too many business decisions are built into the legacy systems. Hence re-engineering. But project development is always short on time and money, making it necessary to look at alternatives. The use of COTS packages is seen as a way to increase reliability while decreasing development and test time. Translation of code is a means of decreasing time and cost. This results in a combination of development methods into a form of hybrid re-engineering. Although any type of re-engineering has associated risks, metrics are available to help identify and mitigate the risks, leading to successful software development.

9. Future Work

The field of software re-engineering is new, and its risks and metrics are not well understood. The newest area, hybrid re-engineering, is even less well known. The SATC is working to define metrics to quantify the quality of legacy systems and re-engineered products, and to provide reliable measures of progress. It is in this last area, the measure of progress, that we are experimenting with function points.

References

- [1] Albrecht, A., Gaffney, J. "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," *IEEE Transactions on Software Engineering*, 11/83.
- [2] Arnold, R., *Software Reengineering*, IEEE Computer Society Press, 1993.
- [3] Byrne, E., "A Conceptual Foundation for Software Re-engineering," Conference on Software Maintenance, 1992.
- [4] Grady, R., *Practical Software Metrics for Project Management and Process Improvement*, Prentice Hall, 1992.
- [5] Hyatt, L., Rosenberg, L., "A Software Quality Model and Metrics for Identifying Project Risks and Assessing Software Quality," 8th Annual Software Technology Conference, UT, 4/96.
- [6] Jones, C., *Applied Software Measurement*, McGraw Hill, Inc., 1991.
- [7] Manzella, Mutafelija, "Concept of Re-engineering Life Cycle," IEEE, 1992.
- [8] Rosenberg, L., Hyatt, L., "Developing a Successful Metrics Program," European Space Agency Symposium, 3/96.
- [9] Ruhl, M.K., Gunn, M.T., "Software Re-engineering: A Case Study and Lessons Learned," NIST Special Publication, 9/91.
- [10] Sneed, S., "Planning the Re-engineering of Legacy Systems," IEEE Software, 1/95.
- [11] Software Technology Support Center, "STS Reengineering Technology Report, Vol 1," Hill Air Force Base, UT, 10/95.
- [12] Software Technology Support Center, "STS Reengineering Technology Report, Vol 2," Hill Air Force Base, UT, 10/95.
- [13] Wilson, W., Rosenberg, L., Hyatt, L., "Automated Analysis of Requirements Specification," Fourteenth Annual Pacific Northwest Software Quality Conference, Oregon, 10/96.

Biography

LINDA H. ROSENBERG, Ph.D.

Dr. Rosenberg is an Engineering Section Head at Unisys Government Systems in Lanham, MD. She is contracted to manage the Software Assurance Technology Center (SATC) through the System Reliability and Safety Office in the Flight Assurance Division at Goddard Space Flight Center, NASA, in Greenbelt, MD. The SATC has four primary responsibilities: Metrics, Standards and Guidance, Assurance Tools and Techniques, and Outreach programs. Although she oversees all work areas of the SATC, Dr. Rosenberg's area of expertise is metrics. She is responsible for overseeing metric programs to establish a basis for numerical guidelines and standards for software developed at NASA, to investigate the role of metrics in risk assessment and management of software projects, and to work with project managers to use metrics in the evaluation of the quality of their software. As part of the SATC outreach program, Dr. Rosenberg has presented metrics/quality assurance tutorials at GSFC and IEEE and ACM conferences.

Immediately prior to this assignment, Dr. Rosenberg was an Assistant Professor in the Mathematics/Computer Science Department at Goucher College in Towson, MD. Her responsibilities included the development of upper-level computer science courses in accordance with the recommendations of the ACM/IEEE-CS Joint Curriculum Task Force and the advisor for computer science majors.

Dr. Rosenberg's work has encompassed many areas of Software Engineering. In addition to metrics, she has worked in the areas of hypertext, specification languages, and user interfaces. Dr. Rosenberg holds a Ph.D. in Computer Science from the University of Maryland, an M.E.S. in Computer Science from Loyola College, and a B.S. in Mathematics from Towson State University. She is a member of Electrical and Electronic Engineers (IEEE), the IEEE Computer Society, the Association for Computing Machinery (ACM), and Upsilon Pi Epsilon.

Dr. Linda Rosenberg
Goddard Space Flight Center
Code 300.1, Bld 6
Greenbelt, MD 20771
Voice: 301-286-0087
Fax: 301-286-0236
Internet: Linda.Rosenberg@gsfc.nasa.gov

LAWRENCE E. HYATT

Mr. Hyatt is a member of the Systems Reliability and Safety Office at NASA's Goddard Space Flight Center, where he is responsible for the development of software implementation policy and requirements. He founded and leads the Software Assurance Technology Center, which is dedicated to making measured improvements in software developed for GSFC and NASA.

Mr. Hyatt has over 35 years' experience in software development and assurance, 25 with the government at GSFC and at NOAA. Early in his career, while with IBM Federal Systems Division, he managed the contract support staff that developed science data analysis software for GSFC space scientists. He then moved to GSFC, where he was responsible for the installation and management of the first large-scale IBM System 360 at GSFC. At NOAA, he was awarded the Department of Commerce Silver Medal for his management of the development of the science ground system for the first TIROS-N Spacecraft. He then headed the Satellite Service Applications Division, which developed and implemented new uses for meteorological satellite data in weather forecasting. Moving back to NASA/GSFC, Mr. Hyatt developed GSFC's initial programs and policies in software assurance and was active in the development of similar programs for wider agency use. For this he was awarded the NASA Exceptional Service Medal in 1990.

He founded the SATC in 1992 as a center of excellence in software assurance. The SATC carries on a program of research and development in software assurance, develops software assurance guidance and standards, and assists GSFC and NASA software development projects and organizations in improving software processes and products.

Larry Hyatt
Goddard Space Flight Center
Code 302, Bld 6
Greenbelt, MD 20771
Voice: 301-286-7475 (voice)
Fax: 301-286-0236
Internet: Larry.Hyatt@gsfc.nasa.gov