Why Reengineering Projects Fail

John Bergey
Dennis Smith
Scott Tilley
Nelson Weiderman
Steven Woods

April 1999

Why Reengineering Projects Fail

John Bergey
Dennis Smith
Scott Tilley
Nelson Weiderman
Steven Woods

April 1999

Product Line Practice Initiative

This report was prepared for the
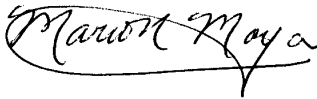
The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

Mario Moya, Maj, USAF
SEI Joint Program Office

Table of Contents

Abstract

The purpose of this report is to highlight some of the most important reasons for failures in reengineering efforts despite the best of intentions. We support our observations with examples from a variety of experiences over many years. Readers may recognize some of the situations presented here and be tempted to conclude that the examples are taken from their own organizations, but similar missteps occur quite frequently.

1 Introduction

One of the primary functions of the SEI is to transition software technology. An important area of software technology is the process for migrating legacy systems to a more desirable target system, especially to a product line. To help the software community migrate its legacy systems, we have published the *Enterprise Framework for the Disciplined Evolution of Legacy Systems* [Bergey 97] (hereafter referred to as the Framework) and "System Evolution Checklists Based on an Enterprise Framework" [Bergey 98] (hereafter referred to as the Checklists). The Framework describes a structure and context for exploring reengineering decision analysis and disciplined approaches to systems evolution. It attempts to improve the reengineering state of the practice, to evaluate reengineering projects, and to characterize reengineering initiatives.

The Framework and Checklists describe a structure for reengineering systems and asks questions about the current state of the legacy system and the transition process. In practice, many things can and do go wrong. Organizations moving from a legacy system to a new system are, in many instances, highly dysfunctional. This has been made clear from our many experiences interacting with both government and corporate clients.

The SEI has attempted to bring to light case studies of exemplary organizations that are doing leading-edge work in software engineering. One such case study is the product line work being done by Celsius Technology, from which we produced a comprehensive technical report [Brownsword 96]. But just as it is important to learn from exemplars, it is also important to learn from mistakes.

The situations in this report are taken from real experiences known first hand to the authors or related directly to us, but care has been taken not to reveal the sources. In some cases, the examples may be combinations of two or more different situations, or the "facts" may have been altered in insignificant ways. It is expected that readers attempting a reengineering effort will recognize potential hazards from among these real cases and will be able to redirect their efforts to avoid most of them. This report will have succeeded if it raises the general awareness of the potential problems that are most likely to occur in real reengineering efforts.

Ten Reasons That Reengineering Efforts Fail

1. The organization inadvertently adopts a flawed or incomplete reengineering strategy.
   [We have a bulletproof strategy.]

2. The organization makes inappropriate use of outside consultants and outside contractors.
   [We rely on experts to help us get there.]

3. The work force is tied to old technologies with inadequate training programs.
   [We're known for our on-the-job training.]

4. The organization does not have its legacy system under control.
   [We are on top of it—we know the system inside and out.]

5. There is too little elicitation and validation of requirements.
   [Our needs are simple and straightforward.]

6. Software architecture is not a primary reengineering consideration.
   [Anybody can specify an architecture.]

7. There is no notion of a separate and distinct "reengineering process."
   [We have our best people working on it.]

8. There is inadequate planning or inadequate resolve to follow the plans.
   [We're too busy to plan.]

9. Management lacks long-term commitment.
   [Tomorrow is another day.]

10. Management predetermines technical decisions.
    [If there's one thing we're good at, it's giving orders.]

For each of these ten reasons, we now will provide a brief description of the problem, followed by several examples from our experience that illustrate the problem, followed by highlights from the Framework where the issues associated with the problem are covered. No prior knowledge of the Framework or Checklists is assumed, but the reference section provides links to our Web site where the complete publications can be viewed or downloaded.

2 The Top Ten Reasons

2.1 Reason #1: The organization inadvertently adopts a flawed or incomplete reengineering strategy

While most organizations have a long-range strategy when they embark on a reengineering effort, these strategies may be seriously flawed or incomplete due to poor assumptions or lack of attention to detail. In some cases, problems can occur because the wrong problem is being addressed. In other cases, not all of the components and steps are considered. An example of a flawed strategy is when an organization chooses to "replace" rather than "repair" a major subsystem while at the same time abandoning corporate knowledge about the legacy system. Another example of an incomplete strategy is when an organization chooses a "big-bang" implementation approach that ignores how to deploy and transition the system into operational use.

Clearly, high-level strategic choices have substantial impact on the success or failure of a reengineering project. Just as architectural decisions have long-lasting impact on the structure and operation of a system, these early strategic reengineering decisions are difficult to change and have repercussions on the overall reengineering result. When the first steps an organization takes toward a new system are inherently flawed, the result will tend toward disaster.

2.1.1 Examples

A flawed transition strategy

An organizational decision was made to move from a flat file telemetry system on obsolete hardware to a client-server architecture on modern computer resources. No flexibility was allowed as to the changeover date. Parallel operation of the two systems was not possible and the old system was scrapped immediately. As a result, the new system did not have all the capability of the old system. It had some new features and a fancy new user interface, but users complained that it did not have some of the old functions. More than a year passed and the system still did not have its planned initial operational capability. To make matters worse, software development and maintenance positions were cut shortly after installation because the client/server system had been justified in the first place as easier to maintain. As a result, full operational capability was delayed indefinitely. The organization has reaped some benefits from the new architecture, but has inadequate personnel to achieve full operational capability and properly maintain it.

A flawed environment integration strategy

A project with a set of chronic problems was reorganized with new personnel and a new strategy. The new strategy involved reengineering current code to a scaled down set of requirements. As part of this strategy, a new software engineering development environment was employed. The environment consisted of a set of proven tools, each of which was well regarded as among the leaders in its class. However, the plan depended on integrating the tools in a seamless way and on using this environment for production of the system. The schedule called for the environment to be brought up over a weekend. The integration did not work, and production came to a grinding halt. Eventually, a scaled down and more feasible strategy was developed, but the project lost precious time and millions of dollars in getting back on track. The lesson is that even a relatively small part of an overall plan can cause problems if it is on the critical path, and even when individual components are proven, integration aspects can come back to haunt a project.

A flawed strategic process

An organization had the goal of eliminating a mainframe system and replacing it with a new set of workstations. The new system would receive input from external facilities. This data would be decoded and some preprocessing would take place on the workstations. The data would then be sent to a supercomputer system that would do quality control as well as final processing and edits. Most of the complex models would run on the supercomputer system. Some of the models would then be transferred to the external facility while others would be sent back to the workstations for additional processing.

The job was a complex engineering effort consisting of several stages. In order to get to the future state, several intermediate states needed to be reached. The installation of the workstations required hardware installation and testing, development of new software, integration testing, and operational installation of the software. In addition, there were ongoing corrective fixes to the existing system, which needed to be placed under change control, while the changes were prioritized and fixed. Despite the complexity of the task, it was initially viewed as a simple task of replacing processors that could be handled in conjunction with the normal maintenance activities of the maintenance staff. The task overwhelmed the staff, and eventually the project was canceled.

2.1.2   Related Framework Highlights

By adopting a high-level structure for decision making early in the reengineering process, problems such as these can be more easily avoided. The inputs driving the reengineering decision analysis should include the enterprise strategy, programmatic issues, economic issues, and technical issues. The strategic issues include the value of the effort, the corporate impact, and the timing. Programmatic issues include resources, priorities, contracting, deliverables, schedule, and risk. Technical issues include feasibility, approach, architecture, tools, and risk. Economic issues include cost, make/buy decisions, and return on investment.

2.2 Reason #2: The organization makes inappropriate use of outside consultants and outside contractors

Outsiders can often offer substantial benefits a project for a number of reasons, such as understanding of the domain, technical expertise, objectivity, or simply the ability to bring extra personnel to a project quickly. However, if used unwisely, they can also contribute to the failure of reengineering projects. Since outsiders rarely know the business as well as insiders, their role needs to be carefully defined and monitored. Organizations and outside contractors often have conflicting interests. The former obviously wants to minimize the cost of external resources, while the latter wants to maximize it. Sometimes the contracting organization relinquishes all control to the contractor. However, it is important for the contracting organization to retain sufficient insight into the work to know if the project is headed for trouble.

Often, three, four, or five sets of consultants will have looked into a problem over a period of as little as a year. Each group often finds similar problems, but the problems persist even after being brought to light. Sometimes the consultant's reports are rejected as being biased in some way. Sometimes the consultants don't have the right experience or credibility. Sometimes they are not given the time to do an adequate job. Sometimes the management just wants to give the impression that they are addressing problems in some way by stirring the pot. In these cases the problem is not with the consultants, but with management.

Conversely, reengineering efforts can also fail when they shun outside help when they actually need it. Outsiders often bring a fresh perspective or additional manpower that is otherwise unavailable within the organization. The attitude that all knowledge exists within the organization can be just as damaging as the converse.

2.2.1 Examples

Giving up control to consultants and contractors

A parent organization brought in two groups of consultants to recommend options on whether to replace or rebuild a legacy system that was no longer able to meet emerging business needs. One group was to investigate the repair option and the other was tasked to investigate the replace option. A detailed joint report was produced with personnel from the contracting organization recommending that some of the system be repaired and some of the system be replaced. The organization did not have a strategy or set of decision criteria established ahead of time. Essentially, they ignored the recommendations of both consultants, and moved to a third contractor who took on the work of replacing the systems. Later, when the third contractor ran into difficulties, they scaled back the contract, but they still did not achieve the desired reengineering result. Perhaps each of the contractors could have offered some benefit. However, since the company's strategy shifted dramatically depending on internal political currents, they failed to gain from any of the outside contractors.

Getting the outside contractor that you are paying for

Contracts for system evolution often cost tens or even hundreds of millions of dollars. Often, because of political considerations, there are tens of contractors in dozens of locations working on a single system upgrade. Because some contractors are so big and operate out of so many different locations, it is hard for a contracting organization to know what it is buying when it signs so many contracts. In one case the contractor highly touted itself as a Capability Maturity Model® (CMM®) level three organization, but in fact was only CMM level three in limited pockets of the organization. When the contractor started getting behind schedule and asking for vastly increased payments for the contracted work, it was clear that the people actually doing the work were not up to the standards that were available only in the small pockets of the company. The contracting organization demurred on the requests for more time and money and stopped work on the contract. This resulted in re-planning the effort and long delays as a result of the transition of ongoing work to other contractors.

Time and materials contracts often don't conserve time and materials

One organization relied on a very reputable outside contractor to implement an entire contract. The contract was on the basis of time and materials. The contracting organization had a complex organizational structure, where the end users were isolated from those who were monitoring the contract. The end users had a strong voice (and in fact veto authority) over the requirements. Continually changing the requirements resulted in confusion, unnecessary expense, and led to specifying a system that was not feasible to build. The contractor, whose incentive was more billable hours, gladly accepted each modification to the system.

Although the contractor fell way behind schedule, the contracting organization did not realize this until it was too late. Several interrelated symptoms contributed to serious delays and a system that failed to meet many of its system tests. These included: gold plating of requirements without an analysis of tradeoffs or costs; failure by the contracting organization to establish meaningful milestones for monitoring the contractor; and failure of the contractor to inform the contracting organization when new requirements were reaching the breaking point due to complexity or performance constraints.

2.2.2   Related Framework Highlights

Outside consultants and outside contractors can be used effectively. They do provide manpower that may not be available inside the organization and they can provide an unbiased evaluation of a situation. When consultants are brought in, it is important to understand what role they are expected to play, and what skills they can bring to the table that are not available within an organization. The Framework and its associated Checklists can provide a starting point for assessment and analysis activities by illuminating relevant areas of inquiry. We know of at least one consulting firm that uses the Framework as a vehicle for organizing an inquiry and for probing and evaluating planned and ongoing system evolution initiatives.

® Registered in the U.S. Patent and Trademark Office.

"The model serves to draw out important global issues early in the planning cycle and provides insight for developing a synergistic set of management and technical practices to achieve a disciplined approach to systems evolution" [Bergey 97].

2.3 Reason #3: The work force is tied to old technologies with inadequate training programs

The lack of training for the work force can cause reengineering efforts to fail. In most cases, the reengineering effort will be taking advantage of newer technology as compared to the legacy system. Frequently, the hardware will be changed and updated and new programming paradigms will be adopted. New vocabularies will be introduced for new ways of doing business. For example, many new systems will be based on the Internet, the Web, and distributed component technologies. Old systems are often based on a functional style of programming using mainframe computers and radically different file systems. New programming languages and new user interfaces are commonly adopted. It is simply not possible to continue to do business as usual while at the same time bringing the same work force up to speed on the new technologies. Either there must be a conscientious and persistent effort to upgrade skills of the existing work force, or there must be a replacement of the existing work force, or there must be new workers added to the work force, or some combination of the three.

2.3.1 Examples

Middle managers see the trees, but not the forest

The federal government uses language standards such as Ada and architecture standards such as Defense Information Infrastructure Common Operating Environment (DII COE) [DISA 97] and High Level Architecture (HLA) [DMSO 98]. But many government employees are not trained to fully understand these standards and appreciate what they do and what they fail to do. Some understand the terminology at superficial levels.

For example, one might naively expect that the HLA is an architecture that solves high-level enterprise problems rather than one that serves primarily to federate a collection of disparate simulations. But a mid-level manager rejected a domain-specific architecture for the organization's test and evaluation domain, because he thought that HLA was sufficient for the task and that no high-level architecture was needed for reengineering the enterprise. This decision was due to lack of understanding of what the new technology can do and what it cannot do.

The result is that a major initiative to upgrade and combine missions of disparate laboratories is proceeding without an appropriate domain-specific architecture.

An aging work force that would rather not learn much

One organization has a centralized maintenance group whose work force consists of hundreds of employees who are highly unionized. The vast majority of the workers have been working on the same maintenance jobs for twenty or more years. They have little turnover and few

have graduated with degrees in computer science. They know the legacy system extremely well and want to continue maintaining it until they retire. They are very resistant to change.

A major system upgrade will totally replace an existing system with new technology including a new language, a new operating system, and a new software engineering environment. The existing work force viewed the system upgrade within the context of the existing system, and their operational support plan was inadequate. It did not provide for retraining the existing staff to learn new technologies and new approaches or adding staff with the required skills. As a result, the organization will need to have a long-term maintenance contract with an outside vendor that will tie the organization to the vendor until either the existing work force retires or a newer work force takes over.

A culture dependent on maintaining the status quo

A large corporation had two different material management systems as the result of an earlier failed migration effort. These systems were deployed on two different platforms and had a significant amount of functionally redundant code with separate evolution paths. One system was used primarily for online data entry with a local version of the entire database. The second system, which was asynchronously updated by the data entry system, stored the permanent records and performed the primary business functions of materials management. This arrangement resulted in chronic code consistency and runtime synchronization problems between the two systems. As a result, a staff of 10 maintenance programmers was required to spend full time (and substantial overtime), fixing bugs and correcting data inconsistencies.

A study was performed that analyzed the work of the maintenance staff, the types of problems being encountered, and the programs that were affected. This study determined, not surprisingly, that data synchronization problems between the two systems were responsible for 80% of the maintenance costs, so it recommended a short-term effort to migrate away from the front-end system. However, the recommendations were turned down. A whole culture had grown up around fixing database inaccuracies, and the associated overtime pay that was required. The existing programmers felt that their jobs were threatened and the management information systems (MIS) manager did not see any clear gain for himself because organizational budgeting considerations made funding of (even short-term) reengineering efforts far more difficult than funding of maintenance. The status quo carried the day and unwarranted maintenance costs continued in the organization.

2.3.2   Related Framework Highlights

Cultural issues such as those described in the examples above are difficult to change in the short term. Training helps to change the culture, but it is not sufficient. The Framework has a lot to say about the organization and technologies (two of the seven components of the framework), which influence the culture over time. Organization is divided into management activities and infrastructure support activities. Infrastructure support includes "training and technology transition." One of the checklist questions is, "Are the training needs of the sys-

tems engineers and software engineers identified?" Among the checklist questions in the Technology component are the following:

- Is the cost, schedule, and impact of applying the new technology acceptable?
- Is adequate training available?
- Are key members of the project team already well versed in the technology?
- Can they act as mentors to other team members?

2.4 Reason #4: The organization does not have its legacy system under control

Before a system can be managed effectively, a system baseline under configuration management should be in place to aid in disciplined evolution. The system needs to be well documented, with an understanding of the priority of change requests and their impact on the system. In addition, the following items need to be in place: data on the costs of maintaining the system, adequate configuration management, and planning and project management capabilities. If these capabilities are not available, the maintenance effort becomes crippled and chaotic, and long-term planning becomes problematic.

The heritage of a legacy system can have a large influence on reengineering failure. Many legacy systems are not under adequate control, because the systems are poorly documented, have inadequate historical measurements, and inadequate change control processes. As a result, it is difficult to understand the current system, to manage it, and to manage changes or plan evolution.

One barometer of whether a system is under control is the way in which change requests are handled. If change requests can be given a priority for both importance and difficulty, rational decisions can be made for new releases. Historical data on similar types of changes, as well as on the ease (or difficulty) of changes to the affected modules, provide an important baseline for being able to make changes within schedule and resource constraints.

Another indicator of control is the availability of historical metrics. This includes changes that have been made, costs of the changes, and any problem areas that have occurred. Since every organization is unique, it is important for the data to reflect the unique process and personnel of that organization. For example, data on the initial cost of each component, the size of the component, change history, types of errors, and costs of making changes provide an important part of this baseline.

When such data is not available, it is impossible to make meaningful cost projections for various classes of changes to the system, or to be able to plan on any kinds of long term changes. This results in new releases coming in late, without adequate functionality. It also makes migration efforts impossible to plan.

2.4.1   Examples

Guesses substitute for historical data

An organization had a legacy system that was inadequate for dealing with rapidly expanding business needs. A large scale reengineering effort was undertaken to migrate to a new database and to incorporate major new functionality. The existing legacy system did not track any historical data. As a result estimates were made based on guesses. These guesses, as might be expected, were grossly inaccurate. The result was that the reengineered system came in several years late, and that several MIS directors lost their jobs as the deadlines were missed.

The legacy system had inadequate change processes

A large system had a history of chronic problems. There was a substantial backlog of trouble reports, with little discernible progress in working down the backlog. When an analysis was done, it was found that the change requests did not have any metrics associated with them, nor was there any indication of ease of change or severity. For example, a typographical error on a page of documentation appeared equal to an error indicating the system could not be initialized. In addition, there was no prior analysis of which modules in the system were relatively error free versus those that had many errors. There was not a repeatable process for making changes. As a result it was virtually impossible to estimate how long minor changes would take to accomplish, much less long-term evolutionary changes. (After a major effort at developing data for the trouble reports, a certain amount of stability was established for the system.)

Informality of all system management processes

In a large migration effort, there was not an organized process for implementing change requests, and no historical data on the costs of making changes was available. Essentially, changes were made in a sequential order with some rough ordering of priorities based on the intensity of user requests or complaints. The only data that was tracked was based on the number of maintenance programmers on the staff. Configuration management was minimal. Project plans were not developed, and milestones were very informal. Documentation was sketchy and outdated so that the maintenance staff viewed it as almost useless. As a result, the organization did not have control of the legacy system. When a major change to the system was attempted, it failed because the organization had not established the capability of making either minor or major changes to the system.

2.4.2   Related Framework Highlights

The Framework is essentially about the processes that facilitate the evolution of systems. It defines the legacy system as one of the seven building blocks for a successful system evolution. The legacy system is subdivided into three parts: the core system, the operational environment, and the support environments. The core system is the software intensive system that is the candidate for evolutionary improvement. Its architecture, products and services, and functionality characterize it, as well as its quality attributes (e.g., usability and performance). The operational environment includes the users, the interfacing systems, and network

communications. The support environments include tools for development, integration, and testing. The Framework provides a context for assessing the health of a legacy system. The questions in the Checklist focus on the current state of the three parts of the legacy system. It helps with the decision to repair or replace a legacy system and points to the weaknesses that could be early indicators of failure of a migration effort.

2.5 Reason #5: There is too little elicitation and validation of requirements

System failure can be caused by too little elicitation and validation of requirements for the reengineering effort, as well as by significant flaws in the requirements elicitation and validation process. There is often no documented concept of operations for the target system that has the buy-in of key stakeholders (e.g., external and internal customers, external and internal users, domain experts, and project team).

Requirements specification is a thorny problem even for "green field" developments (i.e., development from scratch). There are functional requirements and non-functional requirements, user requirements and customer requirements, hardware requirements and software requirements, architecture requirements, maintenance requirements and logistical requirements. All of these reflect the fact that requirements are not unidimensional. Requirements are not only an expression of the needs of the intended users, but of the many stakeholders who have a vested interest in the system.

For reengineering efforts there are additional problems in eliciting and validating requirements because a requirements baseline for the legacy system frequently does not exist. In the relatively few cases where there may be one, the requirements are typically out of date and do not correspond well to system functionality. Assuming that the existing requirements baseline is in good shape, or assuming that there is a small requirements delta when there is actually a large delta, can be deadly.

2.5.1 Examples

Starting from unsatisfactory baseline requirements

An organization based its requirements approach (and planning) for reengineering a mainframe-based system on the premise they simply wanted to "migrate" the existing mainframe functionality and its processing capabilities to a distributed system of workstations using a client-server architecture. They believed they could forego a formal requirements elicitation and validation process, and just concentrate on the "requirements delta." In their thinking, the delta corresponded to a few new features they wanted to add, along with the processing differences stemming from migrating from a batch-oriented system to an interactive one. This incremental approach to eliciting requirements was rendered more challenging by lack of documentation. There was no user's guide, and the minimal system and software documen-

tation that did exist was quite out of date due to years of software modifications and a legacy of changes to the system.

After a series of setbacks in implementing the migration effort and the breakdown of a remedial planning effort, the organization adopted a recommendation to develop a concept of operations (CONOPS) as a first step in the requirements process. They subsequently produced a CONOPS that aptly described the proposed system from a user's operational perspective. The results were very revealing. When the CONOPS document was circulated among the stakeholders for comment, the organization was caught off guard by the feedback they received. It was the first time the stakeholders had clearly understood the operation of the system and it forced them to completely rethink their prior approach to requirements because major issues had been overlooked.

Failing to recognize a large requirements delta

The requirements elicitation and validation approach used by an organization to reengineer its voluminous, record-based transaction system was predicated on developing a concept of operations and preparing a detailed requirements specification. The concept of operations was not developed from scratch. They chose to describe the target system in terms of their own familiar processing sequence that corresponded closely to the legacy system's batch-oriented processing paradigm that was to be replaced by a data-centric system with a customer focus. Over 75 distinct artifacts, which were to be produced and/or processed by the system, and 30 specific agents (who were to be users of the system) were identified in the concept of operations. However, the CONOPs did not systematically describe the role of these agents, or the function of the artifacts, and how they were used, by whom, and when.

It was not clear if the CONOPS was describing a capability of the current system or the proposed one. This reinforced the notion that the proposed system was a moving target. In addition, the CONOPS did not include any end-to-end, operational scenarios which were needed to provide examples of how the system would conceptually operate and how it would fulfill the diversified needs of the users. Furthermore, there were many instances of internal inconsistencies, and external inconsistencies with other documents such as the more detailed requirements specification. While the CONOPs was viewed by some as satisfying a developmental milestone, it failed as a high-level requirements document and it became "shelfware." There were two reasons for this failure: 1) the development of the CONOPs was largely contracted out without the active participation of experienced users, domain experts, or other system stakeholders, and 2) the document review consisted of a perfunctory sign-off by the project leader and upper management.

2.5.2   Related Framework Highlights

Today there are many practices aimed at doing a better job of defining requirements such as creating user scenarios, rapid prototyping elements of the system, or developing storyboards to better define the user interface and obtain greater insight into the desired system features

and functionality. For example, Ivar Jacobsen's "use cases" [Jacobsen 92] are now often used to capture requirements in a database and can be used effectively to elicit deltas and to validate requirements. Each of these practices is as appropriate for reengineering as for new development. The Framework attempts to draw attention to the important role requirements play, and serve as a catalyst for considering some of the more promising techniques for requirements elicitation and validation. It does this by asking a set of questions such as: "Is there a concept of operations to describe the proposed target system?" "Have operational scenarios been developed to describe how the proposed system will operate?" "Have the concept of operations and operational scenarios been validated with customers, users, and key systems personnel?"

2.6   Reason #6: Software architecture is not a primary reengineering consideration

Failure can occur when a methodical evaluation of the software architectures of the legacy and target systems is not a driving factor in the development of the reengineering technical approach. In the first place, the evaluation is necessary to determine whether the legacy software architecture is viable at all as a base for further development. It may turn out that the best decision is to throw out the existing system and start from scratch.

Software architecture is defined by Bass, et al. [Bass 98] as follows:

"The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them."

If the existing architecture represents a viable starting point, the reengineering technical approach must be grounded in that architecture. To do otherwise is to start a "green field" development and abandon the previous heritage. Most architectures are by nature long-lived and slow to change. Unless the legacy system architecture is well understood, it becomes very difficult to build a new compatible architecture. If the old architecture is well understood, it becomes possible, for example, to use the existing interfaces to wrap components for use in the new architecture. If the old architecture is well documented, it is possible to use the same types of documentation for the new architecture. Failure to evaluate the existing architecture will lead to gratuitous inconsistencies between the legacy and target systems. It will also lead to more work and more troubles.

2.6.1   Examples

Combining subsystems into federations is not always the best solution

This organization had a large number of independent systems that tested various aspects of the avionics for aircraft. Each of these systems worked well independently and the plan was to combine them into a large test system in which independent avionics systems could be

---

tested simultaneously under more realistic scenarios in a real-time environment. The plan called for each of the systems to be upgraded independently and to define interface specifications to insure that all the systems would work together when completed.

However, insufficient time or emphasis was placed on a comprehensive architecture to define the optimal framework into which all the independent systems would fit. In addition, the integration work was delayed due to funding cutbacks. The result was that the independent systems were upgraded without a strong emphasis on how the system would finally be integrated. As the upgrades proceeded, the independent systems became more inflexible and less amenable to change. The interfaces became further refined, but without testing of those interfaces, the chances of them working correctly declined. The end result was inevitably a system that was harder to maintain and with significantly increased total cost of ownership over its lifetime.

In a similar case, there was an attempt to combine three different types of testing (pure modeling, hardware-in-the-loop, and open-air range) across a wide geographical area. One scenario used a real platform for a weapon system, with the weapon system itself on a laboratory bench, together with modeling of the trajectory of the weapon. Connecting the three pieces through satellite or ground links and well-defined interfaces could test the integrated system. Such a federation is supported by an architecture such as the Defense Systems and Modeling Office (DMSO) HLA for integrating independent simulations. But there was not an integrating framework architecture to clearly define rules, standards, and protocols for the individual federates. As a result, the solution was sub-optimal relative to what could have been achieved with an architecture that was had provisions for the broader domain. HLA had facilitated a translation layer between inherently incompatible systems.

Architecture is in the eye of the beholder

There is hardly a word in software engineering that is as widely misused and abused as the word "architecture." As a result, different organizations often present very different ideas of the concept of an "architecture." One organization presented detailed views of the components in a fault-tolerant scheme for switching from one set of hardware and software to backup hardware and software (that was the driving factor of the architecture). Other organizations present a "wiring diagram," giving the major components and their interconnections. The "architecture" was described by a list of commercial products that were used in the construction of the system. The "architecture" is sometimes described by giving a model of how the user interacts with the system. The problem with all these different versions of "architecture" is that it slows and distorts communication. The definition of architecture and the architecture itself must be clearly articulated so that all the stakeholders can communicate with one another. Various views of the architecture are important, but they must be clearly defined and compatible with one another. These views need to bridge the gap between the technical and the non-technical stakeholders.

2.6.2   Related Framework Highlights

The Framework defines the legacy system and the target system as two of its seven elements. Both of these systems consist of their core system and their operating environment. Unless there is a deep understanding of the core system and its operating environment, the requirements of the system evolution effort cannot be understood. The legacy system has customers, customer sites, and user groups. Interfacing systems, networks, and both internal and external users and usage patterns must be considered. In addition, interoperability considerations, security measures, and logistics need to be evaluated. While the Framework does not deal explicitly with more detailed architecture concerns, it helps to formulate the procedural and organizational underpinnings for defining and expressing an architecture. Boehm [Boehm 99] describes the hazards of model clashes in architecture descriptions.

2.7   Reason #7: There is no notion of a separate and distinct reengineering process

The means by which a legacy system evolves can have a large influence on success or failure. The existence of a documented life cycle process and corresponding work products are often wrongly viewed as being evidence of a sound reengineering process. Although work products are a necessary outcome of a reengineering process, there needs to be a set of tasks and guidance to perform each step, as well as an understanding of how the whole fits together. In addition, it is necessary to take a broad reengineering-in-the-large view that integrates the processes and work products for the entire project. When these processes and products are absent, or are simply hollow exercises, failure of a reengineering project is a natural consequence.

Four critical elements of any reengineering undertaking are the people, the technology, the process, and the resources available. Quality people, with ample resources, employing suitable technologies rarely produce a quality product without using a quality process. Although reengineering may be viewed as a relative newcomer on the technical scene, it is no different from any other engineering discipline from the standpoint of being dependent on proven processes. The elements of a software reengineering process closely parallel those of the classic software development process. Reengineering, though, involves a higher degree of "constrained problem solving" by virtue of the fact that the legacy system is the starting point. Typically this entails reverse engineering the legacy system software to obtain comprehensive program understanding as a precursor to reengineering the system.

2.7.1   Examples

A generic paper-driven process is not the solution

As part of an internal improvement program to mature its software practices, an organization developed a comprehensive life cycle model that prescribed a high level process covering all aspects of software development from initiation through deployment and operations. The ini-

tial reengineering project produced a project plan, activity network diagrams, and elaborate activity charts for all the prescribed phases. Although it appeared that everything was progressing normally, something was awry. The project team followed the exact detailed phasing of the model and created all the required documents. However, except for terminology that was unique to their effort, the charts were generic and could have applied to almost any project. The effort gave the appearance that "they really had their act together." However, major pieces of the job fell through the cracks. Critical dependencies in the sequencing and phasing of the tasks were overlooked, key analysis tasks such as reverse engineering and baselining the legacy system and performing an architecture evaluation were not included, and key reviews were treated as perfunctory signoffs with no provision for feedback and incorporation of results.

Management did not involve any of the project personnel in developing the model, nor did they pilot the model, or solicit input from the organization before requiring its use. As a result the process model did not have buy-in from the participants.

By blindly following the model using the prescribed tools, the project adopted a superficial and highly fragmented process that emphasized the production of the prescribed documents instead of the specific systems and software engineering tasks needed to incrementally reengineer the system. Since the model was foreign to the way the project leaders and practitioners were accustomed to doing their jobs and was unilaterally forced upon them, they only half-heartedly agreed to follow it. The results were not surprising to anyone other than top level management.

Metrics require a process to use them

An organization had collected a large amount of system performance data. However, there was not a clear understanding of the problem they were trying to address or the relationship of the particular metric to their business goals and objectives.

One critical problem was how to contain the growing number of trouble reports that were being submitted by users experiencing operational problems. Although data were available on the problems being reported by users, there was not a single, well-defined process for logging, categorizing, validating, prioritizing, and resolving problem reports. The reports were submitted by the external user community, internal system users, and developers. They were subsequently processed by various departments and projects within the organization.

The organization later implemented a centralized, enterprise-wide problem reporting process to obtain quantitative data on the problems being experienced and insight into where they should be investing their limited resources in fixing system problems. It provided insight into the types of maintenance and reengineering strategies that would be most effective for countering the growing number of system problems and implementing new products and production capabilities. Nevertheless, without the tie back to the business goals and objectives, the

organization struggled with what to do with the all the data they collected. This distracted them from dealing with critical system problems.

2.7.2   Related Framework Highlights

To sort out these types of problems, the Framework looks at efforts from a "reengineering in-the-large" perspective as opposed to considering more narrowly focused aspects such as the reengineering of specific software applications and components. While reengineering the software applications is a critical element, it is a lower level task that can be defined once the major reengineering considerations such as defining the operational system concept, migration strategy, and software architecture for the target system are resolved. The object of taking an enterprise approach is to systematically evaluate the major elements that contribute to the reengineering problem and solution space and evaluate how well the organization and project are equipped to perform the reengineering tasks. The Checklists can help to accomplish this.

2.8   Reason #8: There is inadequate planning or inadequate resolve to follow the plans

When reengineering software intensive systems, projects often tend to get out of kilter by focusing on the low-level "software problems" and neglecting the intermediate-level tactical management planning and systems engineering planning aspects of the job. In developing a migration approach for reengineering legacy systems there are many global issues that often need to be resolved by an interdisciplinary team of engineers and domain experts in concert with the software reengineering decision-making. The team must develop a greater understanding of the legacy system, its mission, the operational environment it is deployed in, and the users of the system as well as a keen understanding of the organization's goals and objectives for reengineering the system.

Another symptom of this lack of planning is the absence of a documented project plan that has the buy-in of key stakeholders (e.g., organizational line managers, project team, domain experts, systems and software practitioners). Sometimes the vision and objective can be clear, but there is an inadequate roadmap for getting from the "as is" state to the "to be" state. Major reengineering changes require careful and extensive planning just as they do for "green field" developments. Such plans have many steps and involve actions on the parts of all the major stakeholders. Sometimes these plans are not written down and exist only in the minds of some key people, but with the passage of time plans decay. Sometimes management formulates them, but there is not promulgation of the plan throughout the organization. Sometimes, the plans are incomplete. Sometimes, the plans have inadequate resources for implementation. Sometimes the plans are changed frequently due to changing personnel, changing budgets, or changing whims. In each of these cases, the chances of failure are increased.

2.8.1　Examples

Assigning the planning to a committee without clear leadership and empowerment

A large organization in the middle of reengineering its computing facilities (to reduce their large maintenance and operating costs) began to experience some significant problems. Upper managers became alarmed at the status of the project when they learned that there was not a viable project plan. At the insistence of the organization, the department formed a team to develop a formal reengineering plan. The "planning team" was a loosely knit, *ad hoc* group that included four in-house personnel and two consultants (one internal and one external to the organization).

Each of the in-house personnel was responsible for some aspect of the legacy system and had a role in the reengineering effort. However, the reengineering initiative was not structured like a typical project. All the team members had their normal job responsibilities to contend with and were contributing to the reengineering and planning effort as a collateral duty. None of the team members assumed the position of team leader or acknowledged that they had overall responsibility for the effort; rather, they considered themselves as a "team of peers" who were drafted to develop the plan. Since none of them claimed to have planning skills, it was their desire that the consultants write the plan and they would review it. The consultants avoided taking responsibility for the plan, because (in addition to lacking profound knowledge of the domain and legacy systems) they were acutely aware that if they wrote the plan it would not be the team's plan and would soon become a "shelfware" document.

The result was that the consultants produced an outline for the plan, serving as a catalyst to stimulate discussions for assigning sections for the team members to write. The team members surfaced a lot of tough issues and concerns but floundered in developing the plan. Among the stumbling blocks were the lack of clear cut goals and objectives, confusion over roles and responsibilities, lack of direction and authority to assign resources and make decisions, and an inability to obtain commitments from other organizational units participating in the reengineering effort.

The lack of focused technical management oversight and control

An organization had several hundred projects (in various phases of planning or development) that constituted the near-term maintenance, reengineering, and development efforts to revitalize a legacy system. Each of these projects was independently initiated by a separate unit within the organization in response to a perceived need. Each of the "projects" focused on some aspect of improving the hardware and software system capabilities, but there was no coherent, enterprise-wide view of how all these efforts tied together and how they related to the operational deficiencies and needs. Since there was little oversight or coordination across the enterprise, the organization did not have a common understanding of the need, or its importance to the customer and user community, or its relationship to their own strategic goals and objectives.

From a systems engineering perspective, there was not a consistent and equitable means of determining the validity of the need, or determining its potential impact on the system (including cost and schedule). In addition, there was not a means for evaluating how the need could best be satisfied (e.g., through a maintenance action, or by an existing project working on a related problem or capability, or by creating a totally new project).

Consequently, the organization lacked a coherent approach for sorting out the potentially conflicting considerations associated with new needs and for providing cogent direction to ensure a unified reengineering approach across the organization. Many of the projects completed their efforts only to discover that their particular product (e.g., a revitalized system, subsystem, or component) could not be deployed due to system incompatibilities. These were the result of concurrent and uncoordinated changes to the legacy system baseline, which were occurring on an ongoing basis and precluded integrating new products without significant rework.

Recipe for a Y2K Disaster

A large, complex organization had been in the "thinking" stage about the year 2000 (Y2K) problem for a long period of time. They had issued statements to their collaborators and customers that the problem would be solved in plenty of time. Finally, at the corporate level, the organization issued a detailed "plan" to accomplish the necessary remediation. However, the organization had a set of far-flung business units that operated in a semi-autonomous manner. Because the central administration had little control over the day-to-day operations of the business units, the plan focused on the only area in which it had direct control, maintaining a complex database of each system and the status of its Y2K remediation efforts.

The corporate headquarters exerted substantial pressure on each of the business units to report its progress in status reports. However, there were no increases in budget or resources or other centralized support to deal with the problem. In addition to this "take it out of your hide" approach, the master plan did not provide for any training to define the prescribed database requirements so the business units could eliminate database ambiguities and ensure uniformity of reporting.

Personnel in the business units checked off milestones in the database in direct proportion to the amount of pressure they received from the central administration. However, the reliability of the checked milestones was highly suspect. A number of the major systems were in fact fixed and tested at an early date. However, it was still not clear at the beginning of 1999 whether the numerous less critical systems would survive or fail during the coming year. The data were simply not reliable because the plan was inadequate to deal with the problem.

2.8.2 Related Framework Highlights

The Framework can provide the skeleton for the plans, but it cannot provide the resolve. The resolve is provided when the stakeholders who are responsible for executing the plan are given the responsibility for developing the plan. What the Framework does is start the organization off on the right foot. It asks a set of questions. If the questions are addressed, the organization will be on its way toward a robust plan. In many cases, the lack of resolve can be traced to lack of confidence in the plan or a poor plan. If the plan is a good one and it can be carried out efficiently without a lot of bureaucratic red tape, then the resolve to follow it will come much more easily. Part of the plan is to get the buy-in from stakeholders. Once that has been achieved, the plan will roll on its own accord.

2.9 Reason #9: Management lacks long-term commitment

Management support of the project means careful monitoring and putting things back on track when they stray off track. If management gets distracted with other projects during the course of a major reengineering effort, it will not know when things go wrong. Of course, management commitment is a generic problem that is common to all large-scale projects, even those outside the domain of software engineering. What makes this particularly important to reengineering is that the consequences of missteps in early stages can be catastrophic because errors are so hard to correct when they are found late in the process.

When management abrogates its responsibility by throwing the problem "over the fence" and fails to stay closely involved and adequately support the project, reengineering projects are very likely to fail. When management is not fully committed to a reengineering effort, the project tends to lose its focus. By disengaging and entrusting this effort to others, the plan tends to deteriorate and go in different directions. If management gives up responsibility to lower level managers or outsiders, they will tend to take the project in different directions than were initially intended.

2.9.1 Examples

Managing to your expected lifetime in the position

In one example in the DoD, a new officer took over after his predecessor's three year tour. He immediately made made a reorganization, and put a successful reengineering project on hold. It is typical in the military command structure for a tour of duty to last up to three years, but rarely longer. However, major system reengineering efforts usually last longer than three years. This leads to a built-in conflict between the overall demands of good management of the project and the fact that the military officer will be judged only on what is visible during his or her "watch" and not on what happens several years later when the system is completed and enters maintenance. Operation and maintenance costs may be the next officer's problem. Human nature being what it is, we have observed several examples where judgments may have been clouded due to the military tour-of-duty and reward structures.

In another example at a bank in 1997, an operations vice president refused to give his operations manager the resources and personnel that he requested to attack the bank's Y2K problems. The vice president saw little reason to address that long-range problem when he had short-range problems to solve. He was scheduled to retire in 1999, so he could not be held accountable for any shortcomings on January 1, 2000.

Keep disengaging from the work so it can chart its own course

A large organization was faced with developing a prototyping facility. An interdisciplinary working group was formed to develop the prototyping process. The team consisted of internal personnel, outside consultants, industry and government experts, and local contractors.

The first team meeting resulted in a large turnout and expectations were high. But as the work became more clearly defined and tasking assignments were being made, the team members dwindled down to the team leader and five other active participants, with only the team leader and one of the active participants from the organization. After a few weeks, the last remaining active participant dropped out due to a conflicting job assignment, leaving the team leader as the only remaining organization participant. This left no working-level personnel involved to "champion" the process within the organization.

Despite these impediments, the team initially made reasonable progress in developing a repeatable process for evaluating candidate prototypes, including producing technical decision criteria and data necessary for sponsors, laboratory personnel, and contractors to determine how best to integrate the prototype capability into the target operational environment. However, other tasks and "fire drills" began to consume the team leader's time and effort and made the scheduling of working sessions difficult and very sporadic. The team effort was faltering due to lack of management commitment and follow through. Finally, management unilaterally decided to cut the funding of non-agency personnel without first determining the impact on work in progress.

2.9.2   Related Framework Highlights

There is a whole realm of management activities that are defined in the Framework. They include strategic and business planning, marketing and customer liaison, information technology planning, budgeting and managing resources, organizing coordinating projects, overseeing and evaluating projects, and managing infrastructure support. All these activities take long-term commitment and support on the part of management. There is no place for halfhearted efforts. There is no chance that management can just set the organization off in the initial direction and hope that its momentum will carry it to its ultimate goal. The framework provides detailed checklists to determine whether management is on track in its system evolution initiative. There are key work products that need to be produced, key organization processes that must be followed, and infrastructure support that must be built and maintained. Each of these key elements needs to be communicated through the management reporting chain.

2.10 Reason #10: Management predetermines technical decisions

Mandates or edicts issued by upper management that predetermine the technical approach or schedule, cost, and performance considerations without sufficient project team input or concurrence are frequently seen to cause reengineering failure. More often than we would like to admit, project schedules, costs, and deliverables are dictated by top management decisions. Software is a difficult business, and especially where one is dealing with legacy systems that may have poorly developed components and poor documentation. While top management does need to make decisions on the allocation of scarce resources, it is tempting for them to also determine specific deliverables and timetables. However, detailed planning of schedules and milestones can only be accurately determined through careful study of the technical parameters of a system, based on an understanding of the system, historical data, and knowledge of the specific skills of the staff. When top management prescribes these details with little data other than hunches, the results are usually disastrous.

In spite of the obviousness of this premise, our working experience with customers has shown this to be one of the most prevalent causes for the failure of reengineering initiatives. Although the outcome is very predictable, the cause of failure is often attributed to the technical decision process, the technology that was used, or the reengineering team.

2.10.1 Examples

Firing the manager is not the solution

A large commercial service company was in the third attempt to revitalize its system. The previous two attempts failed and had to be totally abandoned. The organization's chief executive officer (CEO) unilaterally mandated an unrealistic schedule for completing the effort. And to follow through on the "plan," the CEO appointed the organization's chief information officer (CIO) as the sole management agent responsible to oversee the work to completion. When the CIO eventually challenged the mandated schedule (based on the technical analyses performed by the reengineering planning team), the CEO tersely stated his position along the following lines: "If you can't complete the effort in the time allotted, I will find someone who can." It should come as no surprise that the average tenure of CIOs in the organization (and there were a procession of them) was 12 to 18 months. The resulting lack of continuity in management oversight (and support) only served to worsen the problems.

Management edicts cannot fix price, schedule, *and* function

As part of an agency-wide downsizing initiative, an organization was directed to absorb drastic cuts in its budget beginning with the next fiscal year. Management decided that the best way to reduce costs would be to migrate from a mainframe to a client-server system. However, management had predetermined the cost, schedule, and capability parameters for the project. Consequently, the only variables in the "migration equation" left to the project team were the actual implementation approach and the quality of the end product. It soon became obvious to the planning team that the effort and resources required for the system mi-

gration had been grossly underestimated. The effort was being viewed as a simple "migration" task when, in fact, it was a large scale reengineering effort that would have agency-wide impact. What is the difference? In a classic migration approach the existing software is "rehosted" to a new computing platform but the basic functionality of the system remains intact. Only the logical and physical design of the underlying system interfaces is affected. In this case, however, extensive changes affected the system's functionality, data processing algorithms, performance, and overall operation that reflected their moving from an archaic, monolithic system to a highly distributed one.

Edicts to save money now may increase future maintenance costs

At the organizational level, a decision was made to migrate to a new language and operating system. Funding was limited, so a decision was made by management to do the physical conversion first and to try to catch up with the documentation and design issues at a later date. Following this decision, the system was converted through brute force methods. However, the new language was structured in a significantly different way, and the resulting code required a different approach to maintenance and support. A serious problem was encountered when the next release of the system was scheduled. The maintenance programmers did not have strong expertise in the new language, and they could not understand the system without documentation. The release date was missed and a redocumentation and training effort was belatedly undertaken.

2.10.2 Related Framework Highlights

A section of the Framework raises organizational issues and concerns that can have a significant impact on the conduct of a reengineering initiative. At a high level it describes management activities as well as management infrastructure support capabilities. It highlights some of the key organization processes and the key work products. Included among the checklist questions (under the section entitled "should avoid doing") are an itemized list of unfortunate —but all too common—organizational practices. Like the dysfunctional practices described in these examples, they can undermine the technical work and cause disastrous results. One checklist example of a "to be avoided" practice that directly relates to these examples is, "Have some aspects of the solution space been predetermined before analyzing the system and involving the project team?"

3 Summary

Reengineering efforts are replete with examples of failures. In fact, the documented record suggests that there are far more failures than there are successes. The SEI was founded in 1984, at least in part, to investigate why so many software-intensive system developments efforts failed to meet their stated requirements, were late, and went over their budgets. As more and more systems have been built, and more and more systems are evolved from existing systems rather than being built from scratch, the same questions have been raised relative to reengineering efforts. But the task of evolving a system from an existing system has similar pitfalls. There are just as many failures in trying to evolve systems as there are building them in the first place.

We have provided examples of some of the most common reasons for reengineering failures. We have documented the general modes of failure with examples from our direct experience. We expect that these failure modes and these examples will be familiar to most readers. We offer them not to denigrate or second-guess the organizations, but rather to provide awareness that these failures are not at all uncommon and that there is hope for a better way of doing business. Getting it right takes hard work, organization, and planning. By no means does the Framework provide all the answers, but it does provide a basis to start the planning of any migration effort in a wide variety of different organizations. Certainly one size does not fit all and system evolution is a non-trivial and long-range undertaking requiring a wide variety of resources.

The Framework attempts to provide tangible guidance to assist managers in avoiding the failures represented in this report. It does this by providing

- a global frame of reference for answering the questions (i.e., an enterprise-wide context)
- insight into contributing factors (i.e., the Framework elements)
- insight into related activities, practices, and work products
- a set of checklists for probing the relevant management and technical issues

In short, it provides a context for identifying and solving the high-level reengineering problems.

In the words of the Framework:

"Developing and fully validating effective management and technical practices for software evolution is a long-range undertaking. While it is not realistic for an organization to think it

can develop a 'one-size-fits-all' set of practices, it is reasonable to expect that an organization can reach a state where the practices they adapt and use achieve predictable and repeatable results. The enterprise framework represents a starting point for assessing the need for developing a synergistic set of management and technical practices and achieving a disciplined approach to system evolution."

Robert Glass has, over the last 20 years, documented numerous computing projects that failed (e.g., see [Glass 98]). Like Glass, we have found that many of the reasons for these failures can be traced directly to management rather than to technical shortcomings. We need to do a better job in recognizing specific failure modes and in empowering management to take corrective action. The first step is awareness. We hope this report plays a small role in raising the awareness of the reasons for failures.

4 References

[Bass 98]   Bass, Len; Clements, Paul; & Kazman, Rick. *Software Architecture in Practice*. SEI Series in Software Engineering. Reading, Ma.: Addison-Wesley, 1998.

[Bergey 97]   Bergey, John K.; Northrop, Linda M.; & Smith, Dennis B. *Enterprise Framework for the Disciplined Evolution of Legacy Systems* (CMU/SEI-97-TR-007). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1997. Available WWW: <URL: http://www.sei.cmu.edu/reengineering/pubs/97-TR-007/>.

[Bergey 98]   Bergey, John K. *System Evolution Checklists Based on an Enterprise Framework* (white paper). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, February 1998. Available WWW: <URL: http://www.sei.cmu.edu/reengineering/pubs/white-papers/Berg98/>.

[Boehm 99]   Boehm, Barry & Port, Ed. *Escaping the Software Tar Pit: Model Clashes and How to Avoid Them* (Technical Report USC-CSE-98-517). Los Angeles: University of Southern California, 1999. Available WWW: <URL: http://sunset.usc.edu/TechRpts/electronicopy.html>.

[Brownsword 96]   Brownsword, Lisa & Clements, Paul. *A Case Study in Successful Product Line Development* (CMU/SEI-96-TR-016). Pittsburgh, Pa.: Software Engineering Institute, Carnegie Mellon University, 1996. Available WWW: <URL:http://www.sei.cmu.edu/publications/documents/96.reports/96.tr.016.html>.

[DISA 97]   Defense Information Systems Agency. *Defense Information Infrastructure (DII) Common Operating Environment (COE) Baseline Specifications*, Version 3.1. Defense Information Systems Agnecy, March 29, 1997. Available WWW: <URL: http://dii-sw.ncr.disa.mil/coe/>.

[DMSO 98]   Defense Modeling and Simulation Office. *High Level Architecture Technical Specifications*, Version 1.3. Alexandria, Va.: Defense Modeling and Simulation Office, February 1998. Available WWW: <URL: http://hla.dmso.mil/>.

[Glass 98]   Glass, Robert L. *Software Runaways*. Upper Saddle River, N.J.: Prentice Hall PTR, 1998.

[Jacobsen 92]   Jacobsen, Ivar, et. al. *Object-Oriented Software Engineering: A Use-Case Driven Approach.* Wokingham, England: Addison-Wesley, 1992.

REPORT DOCUMENTATION PAGE

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (LEAVE BLANK)	2. REPORT DATE April 1999	3. REPORT TYPE AND DATES COVERED Final

4. TITLE AND SUBTITLE Why Reengineering Project Fail	5. FUNDING NUMBERS C — F19628-95-C-0003

6. AUTHOR(S)

John Bergey, Dennis Smith, Scott Tilley, Nelson Weiderman, Steven Woods

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	7. PERFORMING ORGANIZATION REPORT NUMBER CMU/SEI-99-TR-010

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) HQ ESC/DIB 5 Eglin Street Hanscom AFB, MA 01731-2116	10. SPONSORING/MONITORING AGENCY REPORT NUMBER ESC-TR-99-010

11. SUPPLEMENTARY NOTES

12.A DISTRIBUTION/AVAILABILITY STATEMENT Unclassified/Unlimited, DTIC, NTIS	12.B DISTRIBUTION CODE

13. ABSTRACT (MAXIMUM 200 WORDS)

The purpose of this report is to highlight some of the most important reasons for failures in reengineering efforts despite the best of intentions. We support our observations with examples from a variety of experiences over many years. Readers may recognize some of the situations presented here and be tempted to conclude that the examples are taken from their own organizations, but similar missteps occur quite frequently.

14. SUBJECT TERMS enterprise framework, legacy systems, reengineering	15. NUMBER OF PAGES 30 pp.
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL