Special Report
CMU/SEI-93-SR-5

2

Carnegie-Mellon University
Software Engineering Institute

Reengineering: An Engineering Problem

Peter H. Feiler

July 1993

416218

Special Report
CMU/SEI-93-SR-5
July 1993

Reengineering: An Engineering Problem

Peter H. Feiler

Software Engineering Techniques Program/
Technology Division

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This technical report was prepared for the

SEI Joint Program Office
ESC/ENS
Hanscom AFB, MA 01731-2116

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.

Review and Approval

This report has been reviewed and is approved for publication.

FOR THE COMMANDER

[illegible]

Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

Table of Contents

List of Figures

Reengineering: An Engineering Problem

Abstract: This paper discusses a plan that addresses how the Software Engineering Institute (SEI) may assist the Department of Defense (DoD) in reengineering its large software-intensive systems. This plan is based on a view of reengineering as an engineering problem to improve the cost-effective evolution of large software-intensive systems. This view of reengineering, which takes the whole software engineering process into account, fosters a growth path by leveraging promising emerging software engineering technologies. Reengineering also builds on the industry's improvement in its ability to manage the software engineering process, an accomplishment of SEI work in the Capability Maturity Model (CMM) and its key process areas.

1 Executive Summary

In the last several years, there has been significant discussion about the legacy of software systems and reengineering. As a result of this attention, reengineering tools have begun to appear. Their focus is primarily on deriving information from code of legacy systems (reverse engineering), on restructuring and retargeting code, and on mapping derived design information into a new implementation. In particular, a number of tools exist to migrate information systems implemented in COBOL to new platforms and to upgrade their data representation into a relational form.

While reengineering tools will help in certain aspects of reengineering, reengineering is no more about tools than engineering is about tools. Just as engineering implies a disciplined process supported by engineering methods and automated tools, reengineering practice requires a disciplined process supported by methods and tools. In short, reengineering is viewed as an engineering problem that requires a quantitative analysis of the problem and consideration of engineering tradeoffs in its solution.

In response to Advanced Research Projects Agency (ARPA) guidance, the SEI proposes to provide a leadership role in defining and advancing best reengineering practice, taking advantage of current SEI activities in software engineering technologies and related ARPA activities such as Domain Specific Software Architecture (DSSA), Software and Technology for Adaptable Reliable Systems (STARS), the Virginia Center of Excellence, as well as Air Force activities such as Central Archive for Reusable Defense Software (CARDS) and Software Technology Support Center (STSC).

The SEI proposes the following key activities to support the DoD's reengineering of large software-intensive legacy systems:

- Develop a reengineering maturity framework. This framework identifies and assesses the maturity of software engineering technologies in support of reengineering and establishes quantitative methods as decision aids in engineering tradeoff analyses. The technology results of this activity lead to products such as a guide to best reengineering practice, a reengineering technology roadmap, and a reengineering improvement strategy that leverages reuse and domain-specific architectures initiatives.
- Accelerate the evolution of a taxonomy of domains and architectures and its reduction into reengineering practice. Such a taxonomy is considered essential for successful evolution of a software technology base and for effective identification and promotion of dual use software technology. Domain models and domain-specific architectures allow for more effective reengineering beyond code-level program transformation.
- Identify and analyze desirable and undesirable system properties, their root causes, and their effects. These include properties of legacy systems as well as future systems. A catalog of such system properties is the basis for a systematic (engineering) approach to effective system understanding of legacy systems and their evolution strategies. An understanding of these properties is an essential ingredient in the quantitative methods associated with the reengineering maturity framework.
- Accelerate the evolution of the design record concept toward practical use. The design record concept provides a focus for capture, representation, and visualization of system information and knowledge. The SEI can become a critical link between innovative technology research as directly sponsored by ARPA and its reduction into state-of-the-practice technology. The results of this task directly contribute to the evolution of a codified body of knowledge, an essential element of any discipline.

The proposed activities will not only accelerate the advancement of reengineering practice, but also contribute to the advancement of megaprogramming.

The purpose of this paper is to put the proposed activities in context. This is accomplished by

- defining reengineering and its context (Chapter 1),
- discussing a reengineering framework from a problem-solving perspective (Chapter 2),
- summarizing the state of practice in reengineering (Chapter 3),
- outlining opportunities for improvement of reengineering (Chapter 4), and
- summarizing the activities proposed by the SEI (Chapter 5).

2 Introduction

In the last few years, the world has realized that the number of large systems being built from scratch is rapidly diminishing while the number of legacy systems in use is very high. New system capabilities are created by combining existing systems. At the same time, the context in which these systems have been built has changed. Changes range from changes in the application environment in which these systems operate (e.g., new sensors) to changes in hardware and software technologies (e.g., dramatic increases in processor speed and memory, high-level languages, improved methods). Some of the technologies used when these systems were built can hinder the system's ability to evolve to meet ever-changing demands in a cost-effective way. As a result of these problems, a number of technology solutions have sprung up under a variety of labels, including reengineering, reuse, recycling, modernization, renovation, reconstitution, reverse engineering, design recovery, redocumentation, respecification, redesign, restructuring, and retargeting. For a summary of software reengineering technology, the reader is referred to [Arnold 93].

2.1 Definition

In this paper we are building on Chikofsky's work on a taxonomy [Chikofsky 90], the results of the First Software Reengineering Workshop of the Joint Logistics Commanders Joint Policy Coordinating Group on Computer Resources Management [Santa Barbara 92], as well as insights from ARPA sponsored work including STARS, DSSA and from European efforts sponsored under the auspices of ESPRIT, Eureka (Eureka Software Factory), and the Institute for Systems and Software Technology of the Frauenhofer Gesellschaft [ISST 92].

Definitions for reengineering found in the literature include:

- the examination and alteration of an existing system to reconstitute it into a new form and the subsequent implementation of the new form;
- the process of adapting an existing system to changes in its environment or technology without changing its overall functionality;
- modification and possible further development of an existing system;
- improvement of a system through reverse engineering (and restructuring) followed by forward engineering.

Figure 2-1 illustrates a taxonomy of terms related to reengineering by Chikofsky. In this commonly-accepted taxonomy, software system abstractions are represented in terms of life-cycle phases. Shown are requirements, design, and implementation. The traditional process of developing a system by creating these abstractions is referred to as *forward engineering. Reverse engineering* is the process of analyzing an existing system; identifying system components, abstractions, and interrelationships; and creating the respective representations. Redocumentation and design recovery are two forms of reverse engineering. Redocumentation refers to the creation and revision of representations at the same level of abstraction, while design recovery refers to the utilization of external information including domain knowledge in addition to observations of the existing system to identify meaningful higher levels of abstraction. The third process con.ponent of reengineering is restructuring. *Restructuring* is the transformation of represen. .uions at the same level of abstraction while preserving the system's external behavior. . *ieengineering* is an engineering process to reconstitute an existing system into a new form through a combination of reverse engineering, restructuring, and forward engineeri^ng.

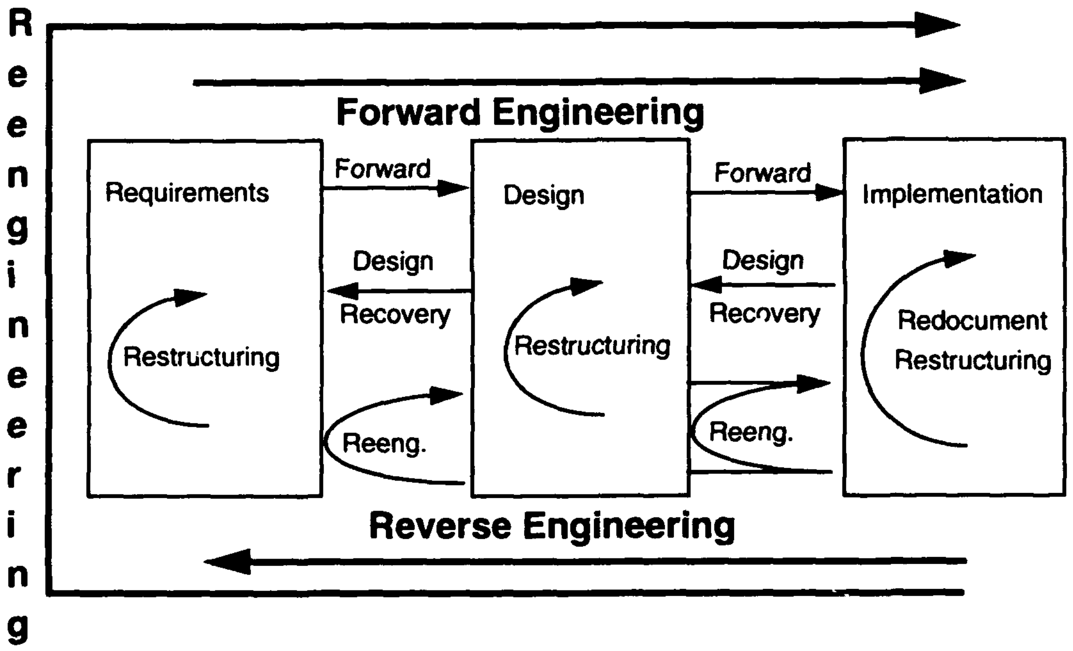

Figure 2-1: Common View of Reengineering

Reengineering relates closely to *maintenance*, which is generally viewed as cons.:ting of corrective, perfective, preventive, and adaptive maintenance. According to ANSI/IEEE Std 729-1983, software maintenance is the "modification of a software product after delivery to correct faults, to improve ;erformance or other attributes, or to adapt the product to a changed environment." In this paper we use the term *system evolution* to include software maintenance.

For the purposes of this paper, we take an encompassing view of *reengineering* as *addressing the engineering problem of (improving) cost-effective evolution of large software intensive systems, both existing and future, through appropriate application of effective best-practice engineering methods and tools.* Evolution of many existing systems is considered as not being cost-effective and cannot keep pace with changes in the application (domain) environment and changes in the computing environment and software engineering technology. The term *legacy system* has been attached to systems with such characteristics. Changes in the application environment (the external environment the application system operates in) as well as in the implementation environment (the hardware/software platform) have to be assumed as a given and have to be accommodated (*engineering for change*). This need for engineering for change applies to both existing systems and new (or future) systems.

2.2 Context

The focus of this paper is on technical aspects of reengineering. However, economic, management, and acquisition aspects play as important a role in the successful improvement of the capability to reengineer legacy systems.

The cost of incremental change to a legacy system needs to be reduced. Criteria for deciding on the need for reengineering range from heuristics such as age of code and excessive maintenance personnel training cost (as found in a 1983 NIST document) to parameterized cost models (see [ISST 92, Santa Barbara 92]). Improvement in this cost is anticipated by investing more than the minimal amount into reflecting the requested change. The additional investment would go into improving the way the system has been engineered with the result of smaller incremental cost in the future. If several legacy systems have to be reengineered, their similarities can be captured in a common reusable architecture, treating them as a family of systems rather than isolated point solutions. The cost models for reengineering, together with better understanding of the effectiveness of different engineering techniques, will allow software engineers to make reasonable engineering tradeoffs as they choose a particular evolutionary reengineering strategy for a legacy system.

Engineering effectiveness is influenced by how well an organization is able to manage its engineering process and improve its engineering capability. SEI has provided leadership for government and industry to improve these organizational software process capabilities through work on the Capability Maturity Model (CMM) and its use as an assessment and improvement tool. In the context of this paper we assume that the reader understands the relevance of such capabilities for an organization's ability to systematically, efficiently, and effectively reengineer legacy systems.

Successful improvement of legacy systems through reengineering also requires attention to improvements in the acquisition process and to legal concerns. The Joint Logistics Commanders Joint Policy Coordinating Group on Computer Resources Management is holding a workshop series to address acquisition issues at the policy level. For further discussion of these and other inhibitors to successful transition of improved software engineering practice see the work done on transition models by SEI and others [Przybylinski 91; Leonard-Barton 88].

3 A Reengineering Framework

In this paper we have cast reengineering as an engineering problem. Problem solving involves an *understanding* of the problem, i.e., a clear understanding of the root causes in terms of its existing state, an understanding of the desired state, and a path (plan) to evolve from the current state to the desired state. Figure 3-1 illustrates this. The current state reflects properties of the existing system and the process by which the system is engineered (developed and maintained). A subset of those properties is undesirable, reflecting the problem to be solved. System understanding reflects the process of creating and maintaining an understanding of a system (through analysis, elicitation, and capture). System evolution represents the engineering activity of migrating the existing system to the desired state. Based on an understanding of the current and desired system state and available (re)engineering technology, an analysis making engineering tradeoffs by considering technical, management, and economic risks and constraints results in a (re)engineering plan. During the execution of this plan (i.e., the actual evolution of the system through engineering activity), the plans may be reassessed taking into consideration changes in the context (e.g., technical changes such as promising new technologies or economic changes such as budget reductions or increases).

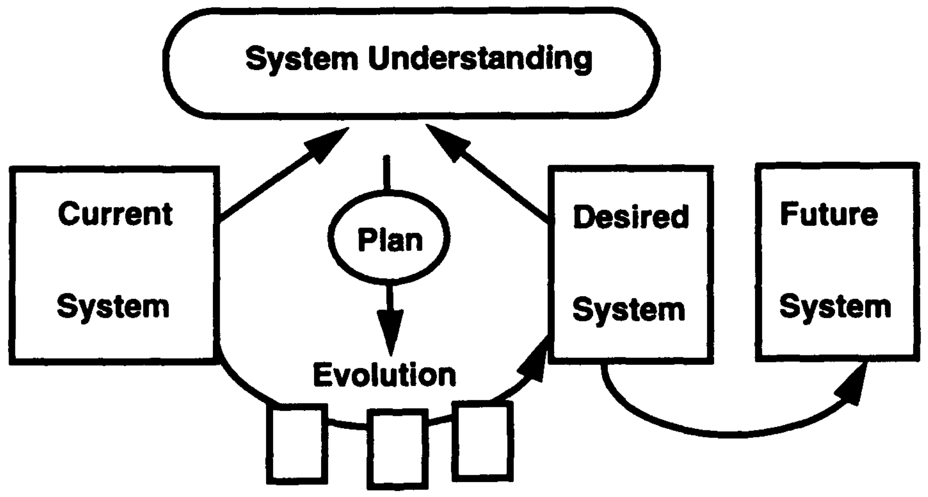

Figure 3-1: Reengineering Problem Solving

3.1 The Current System State

The root causes for the lack of cost-effective evolution fall into two categories: management of the engineering process and the engineering process itself. Management of the engineering process is addressed by SEI's work on CMM and will not be elaborated here. The second category represents technology root causes, i.e., the engineering process, methods, and tools. It will be the focus of further discussion.

The technology root causes manifest themselves in a number of ways. Some examples are:

- Data structures not cleanly implemented. Assumptions that a specific element of shared memory (e.g., Fortran COMMON) is used as the communication mechanism.
- System representations such as architectural and design descriptions reflecting the application domain and the implementation approach may never have been created or documented; the documentation (and sometimes even the source code) is out of date.
- Assumptions about the application environment have been hardcoded in the implementation. Examples include assuming a point solution including fixed number and types of real-world objects.
- The computing environment evolved through several generations. For example, early hardware platforms were memory-limited, resulting in a number of sometimes (in today's view) convoluted implementation "tricks," such as overlay, instruction reuse, and cryptic user interaction. No operating system support was assumed. Today's computing environments typically consist of COTS standard operating systems, DBMS, window systems, and networking support, and are geared toward a high degree of interactiveness and "user-friendliness."
- The implementation technology has evolved from machine code with absolute addressing; to symbolic assembler, high-level algorithmic languages (COBOL, FORTRAN, ALGOL); to languages supporting data abstraction, modularity, information hiding, concurrency support, data modeling capabilities, etc. Design and implementation methods have been coming and going, each leaving its trademark in the code of legacy systems. This code may or may not accommodate the changes demanded from systems today.

Legacy systems also have a number of properties that are worth preserving. Examples include:

- Legacy systems are deployed and have undergone the scrutiny of real users with respect to their functionality meeting their real needs.
- Nonfunctional properties such as performance and accuracy have been fine-tuned.

- Corrective maintenance has resulted in "hardened" code and a wealth of test and validation capabilities.
- System history exists in the form of original designers, current and past maintainers, as well as bug report and change order records.

In many cases some of the root causes and their implications may be understood by some experts, but are not documented and available to the majority of software engineers. Information about systems is quite limited, usually to the source code and/or executable, an operations manual, and people maintaining the system.

3.2 The Desired System State

The desired system state is a combination of properties of the existing system to be maintained, properties expected of a system as part of state-of-the-art software engineering practice and implementation technology, and properties that have their roots in changing environments and are reflected in the system history, but may not have been explicitly expressed by the system user. Examples of maintained properties are functionality, performance, and accuracy. Examples of properties resulting from best practice software engineering and implementation technology include portability, modularity, structure, readability, testability, data independence, documented system understanding, openness (open system), interoperability, and seamless integration. Properties that address continuous change and provide flexibility include localization of information regarding certain different types of change in both the application domain and the implementation, introduction of virtual machine abstractions, and parameterization (dynamic as well as generation technology), COTS, and reuse of components. Properties that encourage reuse of existing engineering know-how include the existence of domain models, domain-independent software architectural principles, domain-specific architectures, and adaptable components.

The desired system state may be known to system users, system maintainers, original system builders, and best software engineering practice experts. The customer (user) may not necessarily be aware of all the potentially desired properties and may only be willing and able to invest in some. Some desired properties can be provided with proven technology, while others depend on emerging technology whose maturity for practical application has not been demonstrated.

3.3 System Understanding

The current state of an existing system and its desired state represent an understanding of the system. This understanding is based on artifacts of the existing system; knowledge and experience with the system as it may exist in users', maintainers', and original builders' heads; and documented system history in the form of bug reports and change records. Figure 3-2 illustrates the sources of information for system understanding. The artifacts are source code, manuals, and the executing system. The knowledge and experience with the system include understanding of engineering decisions, rationale, and possible or considered alternatives, as well as undocumented history and (typically nonfunctional) properties such as performance, robustness, work-arounds, etc. History provides insight into robustness of system components, types, and frequency of changes in the environment (and implementation).

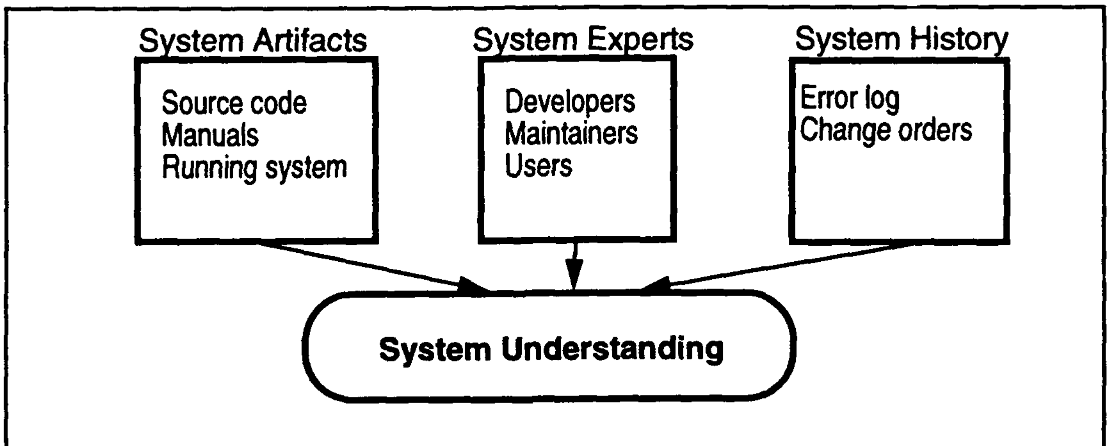

Figure 3-2: Creating System Understanding

Capture, representation, currency, and accessibility of this system understanding is a big challenge. Figure 3-3 illustrates a framework for representation of such system understanding. A central component of system understanding is the system design records that document system representations at different levels of abstraction. This is complemented with rationale for design decisions, the software engineering process and methods used, and the evolution history. Let us first elaborate on models of (software) systems.

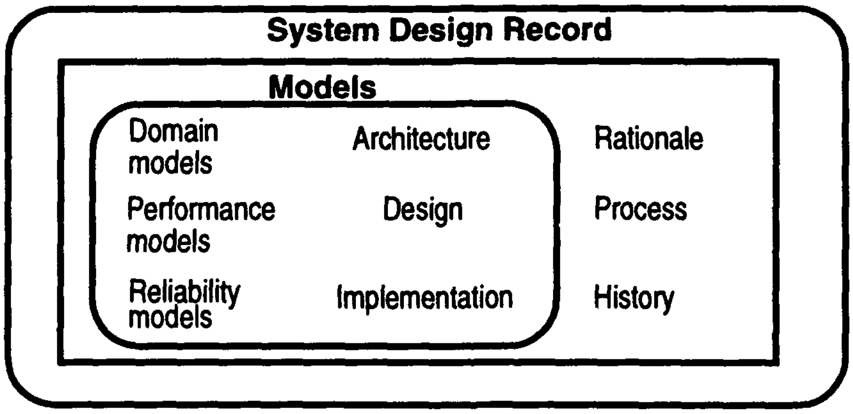

Figure 3-3: Representing System Understanding

These representations are *models* of the system. Models reflect views of the system focusing on certain aspects with different degrees of detail. The purpose of a model is to present a view that is understandable, i.e., not too complex. This is accomplished by the model capturing those abstractions that are relevant from a particular perspective. Some models focus on architectural issues while other models focus on data representation, behavioral, reliability and performance aspects of a system. Examples of models are domain models, domain-specific architectures, real-time timing models such as rate monotonic analysis (RMA), performance models based on queuing theory, etc.

Models have different degrees of formality and may have the ability to be executed. The models may reflect designs (i.e., the notation they are expressed in needs to be transformed into executable implementations), or they may be executable and capture all the desired user functionality and can act as prototype implementations, which can be made more robust or efficient through reimplementation (i.e., transformation into a modeling notation that more appropriately satisfies the need).

As more than one system is considered, models can show their similarities and differences. Systems can be grouped into families. Some models focus on information about the application domain (domain models) while others focus on the implementation architecture. Domain models and domain-independent architectural modeling principles are combined to create domain-specific architectures. Those architectures are populated with components and adapted to the particular application needs. The result is a technology base of models that can be (re)used for a number of systems, leveraging existing engineering know-how. Domain analysis and architectural analysis contribute to the population of this technology base, while application engineering can get adapted to utilizing these models (see Figure 3-4). Furthermore, the technology base can be expanded by the emergence of new modeling concepts, e.g., safety modeling.

While some models represent the executing system itself, other models reflect constraints the system must satisfy. Those are models used to validate desired system behavior. Examples of such models are assertions validated in design reviews or verification, or translated into test suites and test data validating the behavior of the running system. When reengineering a legacy system, such test and validation models exist and have stood the test of time. They can be leveraged for verification and validation of the desired system. Depending on the particular migration path to the desired system, alternatives to full regression testing may be considered. One example is validation of functional equivalence at a certain level of abstraction through comparison of event traces [Britcher 90].

Engineering decisions, rationale, and alternatives complement these models. They may be captured through elicitation processes such as IBIS [MicroComputer Corporation (MCC)]. The models together with the engineering knowledge are known in other engineering disciplines as experience modules.

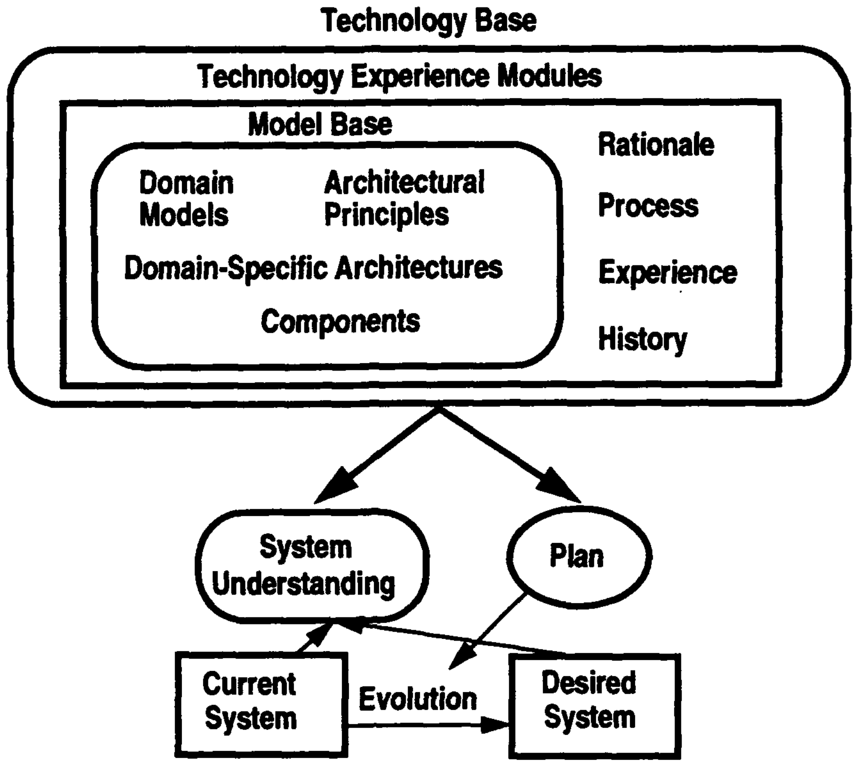

Figure 3-4: Engineering Technology Base

In this idealized view, the amount of engineering information available to the engineer grows tremendously, resulting in information overload. In order to cope with this situation an intelligent intermediary (intelligent engineering assistant or engineering associate) will become essential to the successful utilization of the system understanding. Technologies that are potential contributors to this notion of intelligent assistant include case-based reasoning and intelligent tutoring.

3.4 Evolutionary Migration Path

The understanding of the system, both the current and the desired system state, is the technical basis for determining the particular reengineering strategy to be chosen. It requires analysis, considering alternatives, and making engineering tradeoffs. Such a technical engineering analysis consists of two major components: choosing the degree of legacy leverage, i.e., what can be taken over and what has to be newly created; and choosing the approach for migrating over to the desired system, i.e., how to introduce the changes into the system. The reengineering case study by Britcher [Britcher 90] nicely illustrates that no single approach is appropriate, but engineering tradeoffs need to be considered.

Legacy leverage refers to the ability to utilize (recycle) as much as possible of the existing system in the process of evolving to the desired system. Both the existing and the desired system can be described in terms of a collection of models. For the legacy system, code exists. Other models may have to be derived from the code or other information sources. Certain abstraction may not exist in the legacy system or may reflect undesirable properties. The goal is to eliminate undesirable properties while at the same time introduce desirable properties. Choices have to be made as to which legacy system models to ignore, which ones to transform, and which ones to leave intact. This is illustrated in Figure 3-5. The choices are driven by our understanding of the legacy and desired system properties as well as their reflection in the different models. In concrete terms this means that in some cases, undesirable properties of legacy systems can be eliminated by massaging the code or transforming the data representation, while in other cases a new architecture or data model has to be developed and only a few system components can be translated into the new implementation language.

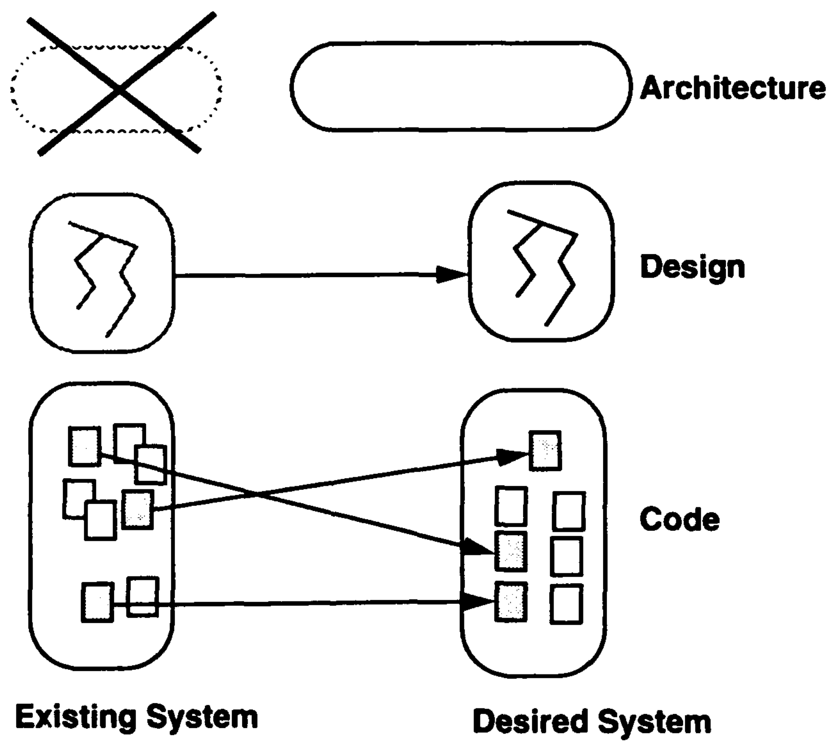

Figure 3-5: System Evolution

The change can be introduced in a number of ways. The following are three classic approaches, but hybrid approaches are possible:

- **Big Bang Approach**: The desired system may be built separately from the legacy system, although parts of the legacy system may have been recycled. Once completed the new system is put into operation while the old system is shut down.

- **Phase-out Approach**, also known as **Incremental Development**: The architecture of the desired system may be created and a skeleton implementation developed. A mapping between the data representation of the legacy and the desired system, implemented as a two-way transformation filter allows the skeleton desired system to run as a shadow of the "live" legacy system, while parts of the desired system implementation are completed and incrementally added to the skeleton. This approach incrementally phases out pieces of the legacy system.
- **Phase-in Approach**, also referred to as **Evolutionary Development**: The legacy system code may be restructured to introduce modularity and partitioning. Desired system properties are incrementally introduced into the existing system resulting in an incremental evolution of both the architecture and the system components.

Validation of the desired system can utilize existing testing capabilities. Validation can be decomposed into validating that the desired system still provides equivalent functionality and detection of bugs in the reimplementation.

The choice of the particular reengineering strategy is affected by the risks the alternative approaches. Risks to consider are:

- Perceived and actual undesirable and desirable system properties
- Ability to eliminate or reduce undesirable system properties
- Maturity of technology inserted into the system
- Introduction of new technology to system maintainers (reengineers)
- Impact of introduction of the reengineered system
- Impact of system changes on performance and robustness
- Cost and time of reengineering

In summary, reengineering is an engineering activity that involves system understanding and evolution through application of appropriate engineering practices. The framework outlined here does not promote particular techniques but accommodates emerging technologies as they mature.

3.5 An Engineering Framework

The framework presented above in the context of reengineering can be used as an engineering framework for software intensive systems. A full discussion of this point is beyond the scope of this paper. The following characterization of different software engineering processes and paradigms serves to quickly illustrate the validity of this claim:

- **New system development**: The system to be improved in the application environment may be a system performing without computer support. This legacy system has desirable properties to be maintained and undesirable properties to be overcome. For example, for many information systems the data model of the legacy system, though not documented, may be directly applicable to the desired system. In traditional life-cycle terminology this is referred to as the requirements phase. Software recycle is applicable only if parts of the legacy system are computer-based. For the introduction of the new system the same migration alternatives may be considered as discussed in the context of reengineering.

- **Reengineering of future systems**: Change in the application environment and the implementation environment are givens. When a new system is being defined, customers often focus on the functionality needed to address their particular problem at that time. Many of the types of changes that will occur over the lifetime of the system and their implications on desirable system properties are not considered during requirement definition. Reengineering of future systems implies that engineering for change and up-to-date maintenance of a system understanding (system design records) occurs from the outset. Engineering for change requires an understanding of commonly-accepted changes as well as an anticipation of paradigm shifts due to new technology and localization of assumptions about certain environment constants.
- **Open systems**: The open systems concept has gained momentum over the last few years, as reflected in organizations such as X-Open, Open Systems Foundation (OSF), and the User Alliance for Open Systems. This concept permits interoperability, allows rapid technology insertion and upgrade, encourages alternative solutions to be applicable, and provides one solution applicable in a number of systems. Characteristics of open systems are modularity and standard interfaces. These are desirable properties of both legacy and future systems as they reduce system cost.
- **Reuse**: Reuse is an engineering activity that focuses on the recognition of commonalities of systems within and across domains. It consists of the creation of models with different abstractions (ranging from code components to domain models) and their use during the engineering of an application. Thus, the focus is on the growth and utilization of the technology base.
- **Evolutionary development**: Evolutionary development focuses on designing the architecture of a system in such a way that the capabilities offered to the user can grow incrementally. New capabilities may be introduced through prototyping of new system components (possibly utilizing different implementation technology). Such prototypes interoperate with the operational system and may get hardened through incremental reengineering.
- **Megaprogramming**: Megaprogramming focuses on recognition of system commonalities at high levels of abstraction (e.g., architecture) and creation of system instances through parameterized automatic composition or generation.
- **Model-Based Software Engineering (MBSE)**: The objective of MBSE is to improve the effectiveness and efficiency of producing software intensive systems through better utilization of engineering experience and system understanding. MBSE focuses on the use of engineering product models as the primary means for improving the construction and maintenance of software.

4 State of the Practice

In many cases an organization with legacy system problems has a low level of maturity, i.e., has an ad hoc management process as well as an ad hoc engineering process and is in search of the silver bullet in the form of the right reengineering tool. Much of the focus of reengineering technology has been on tools. Availability of tools has been market driven. A majority of tools attempt to satisfy the needs of the MIS community. This community has a vast number of legacy systems and the dominating implementation language is COBOL.

Summaries of reverse and reengineering markets and technologies are documented in reports such as [R-EvHa 90]. Categorizations of reengineering tools can be found in [IEEE 90] and [IEEENews], as well as organizations categorizing tools such as Air Force Software Technology Support Center (AF STSC).

Reverse engineering (redocumentation) tools range from code formatters and cross-reference generators to data flow and call graph generators. Analysis tools examine data structures, identify unreachable code, and check for compatibility with language variants, e.g., compatibility with different C compilers. Other types of analysis tools focus on providing statistical measures of "goodness" of code, e.g., McCabe metrics.

Restructuring tools are available for code translation, mostly for translation between different variants of COBOL, but also for translation between languages. Another form of restructuring supported by tools is data restructuring between representations, e.g., from a hierarchical to a relational representation.

In a 1983 document NIST has provided criteria for deciding on the need for reengineering. Such criteria range from age of code (greater than 7 years) to programs being emulated and training cost of maintenance personnel being too high. More elaborate decision aids involve economic cost models. The draft DoD Reengineering Economics Handbook is one example of such tools.

The SEI has developed a curriculum module on maintenance/reengineering as part of the Master of Software Engineering (MSE) curriculum development. This curriculum module reflects current practice and is being used in MSE programs at universities as well as in continuing education programs. Furthermore, there are a number of commercial offerings of reengineering courses and seminars, as well as practitioner conferences on the topic.

A number of other technologies contribute to the practice of reengineering, but are sometimes not discussed in that context. Examples include:

- Rate Monotonic Analysis (RMA) as a real-time performance analysis technique. The SEI has been instrumental in turning the underlying theory into practical application.

- Use of structural modeling and resulting models in the flight simulation community. The SEI has been instrumental in moving that community from Fortran and 60s programming techniques to Ada and modern software engineering practices.
- Component reuse through component libraries. Libraries for generic components (e.g., stacks and queues), as well as for special application domains (e.g., numerical algebra algorithms or missile guidance software (CAMP)) exist. In some areas generation technology (e.g., parsers) toolkit technology (DBMS and UIMS), and higher level languages (e.g., 4GL, SQL) have evolved and been put into practice.
- Quality of the system is improved through engineering process approaches such as the cleanroom method and design or code inspections. Other key process areas (KPA) as outlined in the CMM such as configuration management (CM) can also greatly impact an organization's ability to keep legacy systems up-to-date.

The SEI has projects in several other technical areas actively engaged in impacting practice (e.g., Ada adoption, domain analysis and reuse, software architectures, process definition, CM system concepts, environment integration and adoption, fault tolerance, Ada/SQL, risk assessment and mitigation). These complement the SEI activities surrounding the CMM, organizational change, transition management, and software engineering education.

5 Opportunities for Improvement of Reengineering

Improvement of reengineering practice has a number of facets. As pointed out earlier, this paper focuses on the technical aspects of reengineering and its advancement. The technical aspect can be subdivided into advancement of state-of-the-art of reengineering technology and advancement of reengineering practice. The first focuses on innovative research resulting in the creation of new technological solution with potential for practical use. The second focuses on reduction of promising technologies into engineering practices by creating awareness and understanding, by reducing risk through trial use, and by improving practitioners through improvement and training programs. As part of its transition role the SEI focuses on advancing best practice (including assessment of advances of state-of-the-art) and on facilitating the advancement of common practice.

Advancement in reengineering practice is measurably accomplished through improvement in the software process as outlined in the CMM, i.e., the management of software engineering, whose key practice areas are spread across the five CMM levels, and a placeholder for software development processes at level 3. In short, the CMM represents a Process Maturity Scale (PMS). Actual engineering of software is driven by a second scale that reflects increasing sophistication of software technology and its methodical (systematic) application. In this paper we refer to this scale as the Software Engineering Technology Maturity Scale (TMS). Figure 5-1 illustrates this scale in five units

1	2	3	4	5
Ad hoc	**Repeatable**	**Defined**	**Evaluating**	**Optimizing**
Informal	**Common artifacts**	**Domain spec. architectures Generators**	**Formal Quantitative**	**Cause/effect Technology tradeoff**

Figure 5-1: Software Engineering Technology Maturity

Progression through the units of this scale reflects:

- increasing levels of abstraction,
- increasing degree of formalism with theoretical foundation,
- increasing automation, and
- increasing optimization.

Success in improving the effectiveness and efficiency of systems engineering is accomplished through a combination of the process maturity measure and technology maturity measure.

The foundation for a technology maturity scale consists of conceptual frameworks and taxonomies and accompanying quantitative methods for analyzing technology areas with respect to their applicability to a particular software engineering problem, e.g., reengineering or timing problems in a hard real-time system.

The remainder of this section discusses advancement of the state-of-the-art (innovative research), recognition of technology trends through analytical research, growth of the software engineering technology base through engineering research, practitioner buy-in through community consensus building, and self-sustaining transitions by leveraging established infrastructures. The discussion generates a list of opportunities for intellectual work to advance the state of software engineering and reengineering.

5.1 Advancement of State of the Art

The state of the art is driven by the research agendas of funding agencies such as ARPA, National Science Foundation (NSF), ESPRIT, Eureka, MITI, etc. BAA 93-11 of ARPA is just one example of increased research attention on fusion and application of basic computer science technology in a systematic manner, and system understanding at the architectural level, in particular. Areas of interest to the funding agency and the discussion in this paper include:

- **High assurance systems**: Hybrid approaches of formal methods integrated with architecture, interface specification, and composition supported by state-of-the-art Software Engineering Environments (SEEs)
- **Advanced environments**: Advanced environment architectures and framework prototypes, integration support including metalanguage composition and process programming approaches
- **Component-based software**: Component construction and adaptation as well as architecture-based composition and assembly
- **Domain-specific software**: Domain analyses to capture domain-specific architectures, pilot application engineering based on DSSA. Contributions to precise representations of systems including architecture, module topology, temporal dynamics, and hard real-time constraints
- **Software understanding and reengineering**: Software asset representation (SAR) and accessibility, design records, reuse, reverse engineering, validation, and architecture representation

This trend is not new. A number of research programs and efforts have been underway for several years including ARPA's DSSA program, Prototech program, Rome Labs Knowledge-Based Software Engineering (KBSE) program, as well as non-US efforts such as the ProSpecTra, MACS, ReDo, Docket projects under the ESPRIT program as well as the Eureka Software Factory (ESF) program.

5.2 Advancement of Software Engineering Practice

As a federally funded research and development center (FFRDC), the SEI is considered a neutral and objective party. Its mission causes it to focus on advancement of practice. Its leadership role and its limited size requires the SEI to focus on high-leverage opportunities and, along with other parties, to transition software engineering technology into practice (i.e., to focus on advancement of best practice and facilitate advancement of common practice). We will proceed by first examining four perspectives that contribute to the advancement of software engineering practice and then discuss SEI's role and possible contributions in light of its characteristics (in particular the leadership role). These perspectives are:

- analytical research in the form of technology-trend analysis,
- engineering research in the form of maturation of state-of-the-art technology into engineering use,
- community consensus building, and
- self-sustaining transition infrastructure.

These perspectives address the different phases of technology maturation: awareness, understanding, trial use, and institutionalization. SEI's role and potential contributions have to be viewed in the context of its leadership role.

5.2.1 Technology Trends

Advances in the state of the art result in the creation of new technologies. These emerging technologies may demonstrate the feasibility of a new concept, but their practicality has not been demonstrated. Many hopes are placed in new technology, unfortunately often with little systematic analysis.

Analytical research is a critical element to advancing best practice. Analytical research requires conceptual frameworks to be created as analysis tools. The SEI has demonstrated the value of such frameworks in a number of areas such as CM, CASE environments, model-based software engineering, RMA, process concepts, etc. Technology assessments are performed in the context of such a framework. Technology assessments do not require all products to be surveyed. They involve in-depth analysis of examples of technology to probe for major advances. Such probing is accomplished through hands-on pilot studies in technology testbeds and through case studies of technology application by others. Promising technology may be identified. The identified technology can then be matured into a state that permits its application in an engineering-like fashion (see below).

The conceptual framework and technology probes result in the recognition of possible technology trends. These trends can be mapped out in terms of technology roadmaps. These roadmaps relate different technologies to each other and identify potential for technology fusion, i.e., leveraged advances in practice through a combination of technological solutions.

The technology roadmaps also provide feedback for innovative research, analytical research, or engineering research in particular areas, as viewed from the perspective of improving best practice. Examples of such feedback are the work in Ada 9X, a revision of the Ada language standard to incorporate the latest insights in real-time support (engineering research) and the need for a unified formal configuration management model (innovative research).

In specific terms, there is a need for conceptual frameworks and roadmaps at various levels. The following is a progression of conceptual and analytical frameworks that culminate in a reengineering roadmap for improvement of reengineering practice. Such a roadmap will have been validated through pilots and trial use lessons learned and experiences.

- a taxonomy of:
  - domains as the foundation for a base of domain models
  - architectures as the foundation for a base of architectures
  - desirable and undesirable system properties for existing and future systems including their root causes and implications
  - quantitative models and methods ranging from domain specifics to performance and reliability
- a record of system understanding, i.e., capture and representation of system information and experience
- a reengineering technology maturity framework providing a basis for reengineering improvement strategies

With its leadership role, the SEI is ideally positioned to take a very active role in establishing such conceptual frameworks in cooperation with the research, industrial, and government communities. Due to its ability to take objective and neutral views the SEI is able to evolve technology roadmaps that reflect technology advances throughout the world and remain untainted by the politics of companies, industries, and countries. The SEI has demonstrated its ability to provide technical leadership by teaming up with other experts and fostering community consensus (e.g., CMM and ISO 9000, software metrics, National Institute of Standards and Technology/European Computer Manufacturers Association (NIST/ECMA) reference model, Next Generation Computing Resources Project Support Environment Standards Working Group (NGCR PSESWG) reference model, North American PCTE Initiative (NAPI)).

5.2.2 Engineering of Technology

Technology identified as promising has to be matured for engineering through systematic application in real-world pilot projects. Part of this maturing involves creation of prototype implementation of technology that scales up as well as adapts the engineering process to effectively incorporate the technology. Lessons learned from the pilots identify benefits and limitations of the technology. The experience gained from such pilot applications results in engineering expertise which, if captured and made available, allows other engineers to make engineering tradeoffs without having to duplicate the learning experience.

This leads to a number of activities that are essential for increasing our corporate engineering knowledge. They include:

- populating the model base, i.e., creation of domain models, domain-independent architectural patterns, and domain-specific architectures in various domains;
- complementing the models with expertise, i.e., rationale, history, context and assumptions and their consequences (system properties);
- adapting engineering processes to be model-based, and refining of engineering views such as reuse, reengineering, open systems, etc.; and
- populating the reengineering maturity framework with key software engineering technologies and validating of their effectiveness.

Such engineering knowledge becomes the foundation of software engineering as an engineering discipline. It can be captured in a software engineering handbook. Its content is driven by the conceptual frameworks outlined above. It presents the engineer with applicable technologies, methods, and tools, strategies for choosing among them considering engineering tradeoffs, and for adapting the engineering process. Its structure allows new technologies, methods, and tools to be incorporated, thus, allowing it to reflect the state of best practice. It is supported by a compendium of handbooks detailing engineering activities at the tactical level from the perspective of different engineering roles (e.g., reengineering economics handbook, Ada adoption handbook, reuse handbook, RMA handbook, structural modeling guidebook, CASE integration guide).

SEI's need to leverage its limited resources requires cooperation with a number of other parties. Programs such as STARS, DSSA, PRISM, CARDS, etc., and other advanced technology demonstrations (ATD) in both government and commercial industry provide fertile ground for gaining engineering experience. The SEI has a number of ongoing efforts that build toward software engineering as a discipline, ranging from domain analysis, software architectures, elicitation of system understanding, and engineering of CASE environments, to real-time concerns, fault tolerance, distribution, and simulation. The character of SEI projects has changed from technology innovation to technology analysis and engineering. A number of other organizations have made similar shifts: as our understanding of software engineering matures as do software engineering technologies. As a result, a number of frameworks, maturity models, and handbooks are emerging (e.g., the reengineering economics handbook [Santa Barbara 92], reuse capability model [SPC/VCOE 92], CMM-RealTime [SEISymposium 92]).

The SEI has an opportunity to take the lead in evolving a strategic framework for software engineering as a discipline while leveraging ongoing work. Such a framework would result in more effective coordination and leveraging of ongoing technology analysis and engineering work.

5.2.3 Community Consensus Building

Adoption of technology is facilitated by creating acceptance and by an appropriate technology maturation infrastructure.

In the past, a number of technologies have become accepted because a major player dominated the market. In that case, conceptual frameworks were of benefit to the technology user because they provided an understanding of the technology at hand. In recent years, the scenario has changed in that a number of products offer a particular technology and technologies have to interoperate. The result is an increased need for technology providers to cooperate and agree on technology standards rather than create technology niches. This is evidenced in the increasing number of standardization activities initiated by industry through customer pressure as well as through government and standards bodies. In this context conceptual frameworks and their analytical use become consensus building tools, as they provide objective insight into alternative technology solutions.

In several areas the SEI has recognized this shift and has taken on a technical leadership role in selectively chosen community forums to foster agreement on a common view through an agreed-upon conceptual framework. The benefit of such an approach is three-fold. First, the evolution of the framework benefits from the contributions and critical review by technical experts participating in those forums. Second, the forum and its participants become the champions of the framework and assist in the technology transition to their community. Third, the SEI leverages existing organizational infrastructure of the appropriate forum while focusing on technical leadership. Some organizations have the charter of establishing and publishing standards (IEEE) or recommending standards for acquisition (NGCR, NIST), while others complement the standard with guides, validation suites, and certification processes (ISO, AJPO, ECMA). Some organizations, e.g., NIST, have technology testbeds and outreach programs in place for practitioners to gain insight into operational best-practice technology of particular interest to smaller companies with limited research and development budgets. These standardization and outreach activities establish an infrastructure for maturation and reduced resistance for acceptance of technology.

On occasion, the SEI or one of the forum has recognized similarities in the problems being addressed by two communities that seem to operate relatively independently. The technology analysis research function can play an effective role in bridging these communities. Recognition and understanding of similarities and differences in the technical problems and pursued solutions allows these communities to leverage from the experiences in their domain. This is becoming especially relevant as dual-use of technology is becoming an important factor.

Examples of bridging communities with potential high pay-off include:

- reuse and reengineering;
- parser technology and pattern/message processing;
- configuration management support in CASE, CAD, CAM, CAFE, etc.;

- environment integration and flexible environment architectures in CASE, CAD, CAM, CAFE, etc.; and
- command and control, logistics systems in military and commercial contexts (e.g., Army movement, Federal Express).

5.2.4 Transition Infrastructure

- Current practice is advanced through institutionalization of technology. Institutionalization is facilitated by a solid educational infrastructure for basic and continued education, software engineering process groups, and technology receptor groups.
- In its leadership role, the SEI has focused on creating a curriculum and curriculum modules for a master's degree in Software Engineering (MSE), including a module on maintenance/reengineering. This curriculum has been the basis for a number of university software engineering programs as well as for in-house industry and commercial continuing education programs. In leveraging its limited resources, the SEI benefits from focusing on a process for incorporating advances in best practice into the curriculum, and from cooperating in the development and piloting of courses, leaving the adaptation and delivery to the educators in the community. As conceptual frameworks mature, they can provide additional conceptual structure through incorporation into the curriculum.
- The SEI is taking on an active leadership role in the identification and transition of technology-intensive solutions to education and training. Utilization of on-line presentation material and the software engineering video network (SEVN), including satellite connections, are already in place. Technology work in advanced tutoring based on multi-media technology at the SEI and involvement of the CMU School of Computer Science (SCS) in a multi-media educational technology consortium (with Intel, Sony, Pittsburgh Public Television Corporation QED) offer an opportunity to provide insight into educational technology trends.
- A number of SEI projects are cooperating with transition agents in outside organizations (e.g., DISA, STSC). SEI's focus on advancing best practice allows the SEI to provide strategic technology insight, while the transition agent organization concentrates on operationalizing technology transition at the tactical level.

6 SEI Role in Improving Reengineering Practice

First, we summarize ongoing activities that contribute to the solution of the problem of reengineering. Then, we outline a set of activities that have great potential for advancing our understanding of software engineering and reengineering in particular, for which the SEI is in a position to take the lead and cooperate with CMU-SCS on the research components.

6.1 Ongoing SEI Software Engineering Activities

For the past 5-6 years, the SEI has had the lead in evolving and applying the five-level software process Capability Maturity Model (CMM). Its focus has been on the management of the engineering process. Accompanying this activity, a core competence in process modeling (i.e., the capture of processes in process definitions) has evolved and is providing the technical lead in the STARS Process Asset Library (PAL) effort. A second accompanying effort in process metrics has established a guide to process measures in the context of the CMM.

Over the last 2-3 years, the SEI has evolved a framework and quantitative methods for assessing management and technology risks. This assessment instrument is being piloted in both government and industry and is starting to find its way into industry practices.

The SEI over the last 3-4 years has taken Rate Monotonic Analysis (RMA) theory through engineering research resulting in a quantitative method for software engineers to systematically analyze and design systems to meet hard real-time criteria. This activity has culminated in an RMA handbook and self-sustaining transition of the technology.

The SEI is in the process of evolving conceptual frameworks to address reliability properties of systems through fault tolerance and zero defect approaches.

The SEI has focused on moving practitioners' attention from searching for the best tool or method, i.e., silver bullet, to capturing system information in models of increasing levels of abstraction (e.g., in form of domain specific software architectures). For that purpose, the term Model-Based Software Engineering (MBSE) has been used. Some of the work in the use of models has its roots in SEI's efforts 6 years ago in helping the flight simulator community to move from FORTRAN to Ada through a pattern-based approach. This has resulted in structural modeling as a method for expressing flexible flight simulator (and other application) architectures. This method is currently being captured in a guidebook. Other contributions are SEI's work in domain models, i.e., a framework, process, and method for creation of domain models through utilization of commercially available design notations and methods, the piloting of domain modeling in the user interface and logistics domains, and the evolution of an application engineering process centered around domain models and complemented with structural modeling. This activity closely interacts with DSSA, STARS, and CARDS. Current emphasis of this MBSE view to application engineering is on a reuse perspective. In a related research activity with the CMU School of Computer Science (SCS), software architecture principles and an architectural taxonomy are being explored.

The SEI is investigating use of technology for software engineering information capture, representation, and accessibility. This is being accomplished through fusion of state-of-the-art technologies in elicitation, hyper- and multi-media, information abstraction, and advanced tutoring. This activity has its roots in Advanced Learning Techniques and support for ARPA's High Definition Display (HDD) program. It has strong interactions with the CMU Engineering Design Research Center (EDRC), the Robotics, and Information Technology Center (ITC), and is a partner in the Mediamation Alliance, whose other partners include Pittsburgh Public Television Corporation QED, CMU-SCS, Sony, and Intel. Through the incorporation of advances in ARPA's design record effort, a technology framework in support of system understanding can be evolved (resources permitting).

Computer-aided environments (CASE, CAD, CAM, CAFE, CAPE) are large systems that have to be engineered as flexible, open [illegible] supporting various engineering processes. This is a challenge as CASE tools have [illegible]f-life of 3-5 years, with many releases in that period. The SEI has been instrumental in helping the community understand the complexity of issues involved in assembling and adopting CASE environments. This has been accomplished through conceptual frameworks for CASE adoption and CASE integration. These frameworks are being used by IEEE, NIST, and NGCR to identify and strategically pursue areas of standardization in support of open systems and interoperability in CASE environments. The same frameworks have become the foundation for guidance to practitioners for adopting CASE technology and for an environment technology roadmap. As such, CASE environments technology is becoming a (model-based) software engineering domain in its own right with several commercial players emerging. The framework for the environment technology roadmap has been designed to accommodate advances in process enactment technology; the SEI is actively tracking this technology. The frameworks clearly identify the great overlap between CASE, CAD, CAM, CAFE, and CAPE and the respective groups have shown interest in this work illustrating the benefits of models to recognize commonalities within and across domains. In the long term, process-centered environment technology and technology for managing software engineering information (see above) is envisioned to fuse into a system providing intelligent engineering assistance utilizing codified engineering knowledge normally found in handbooks and in the minds of experts.

The SEI recognizes that software engineering is a team activity that must be supported. Components of a CASE environment (i.e., individual tools) focus on supporting individual engineers. Configuration management (CM) systems provide support in controlling concurrent engineering activities and maintaining a history of software evolution. The SEI took the lead in establishing a conceptual framework for CM services. This framework is used as a guidepost for practitioners to examine commercial products and for researchers to unify CM concepts. As research in cooperative work (a.k.a. groupware, concurrent engineering support) technology is maturing, its applicability in the context of engineering environments will have to be investigated.

In summary, although the SEI does not currently have a focus on reengineering, many of the pieces are already in place.

6.2 Potential SEI Reengineering Activities

This section focuses on reengineering specific activities that are strategically important to the advancement of reengineering practice and are appropriate for the SEI to take on. The activities lead to a reengineering maturity framework and technology roadmap that permits organizations to improve their reengineering capabilities. New insights gained from these activities feed into a best-practice reengineering guide to keep practitioners abreast of improving practice.

The central activity is:

- Develop a reengineering maturity framework. This activity will leverage SEI's activities in engineering of systems, i.e., application engineering with domain models, domain-specific architectures, RMA as a timing model, etc. as well as research activities on reengineering at CMU-SCS. This activity will focus on identifying technologies for reengineering systems, assessing their maturity and effectiveness, and evolving quantitative methods supporting engineering trade-offs to be made by practitioners. Products potentially include a guide to best reengineering practice, a reengineering technology roadmap, and a reengineering improvement strategy by leveraging reuse focused activities.

Key activities that provide the foundation for a reengineering maturity framework are:

- Accelerate the evolution of a taxonomy of domains and architectures. This work builds on existing activities at the SEI and CMU-SCS as well as outside activities. The taxonomy is considered essential for effective identification and promotion of dual use software technology.
- Identify and analyze system properties (desirable and undesirable) as well as their roots and effects. The innovative research component explores the identification and analysis of new system properties. The analytical research component consists of locating and consolidating previously identified properties. A catalog of such system properties is the basis for a systematic (engineering) approach addressing effective system understanding and evolution strategies for legacy systems.
- Accelerate the evolution of the design record concept toward practical use by incorporating pragmatic results regarding system properties. The SEI can provide a link between innovative technologists and the state-of-practice. Building on existing activities, the SEI can be instrumental in technology fusion of several technologies for intelligent capture, representation, and access to engineering information.

These activities do not by themselves accomplish major improvement in reengineering capabilities. They provide the structural and analytical framework to allow a reengineering and reuse infrastructure to evolve to meet practitioners' needs. The infrastructure has to grow in cooperation with other partners. Population of a software engineering technology base with results from DSSA, STARS, ATDs, etc., can be accomplished in a self-sustaining manner once the domain and architecture taxonomy have gained community consensus. These activities can leverage currently evolving infrastructure such as CARDS, STSC, etc. The Virginia Center of Excellence (VCOE) in reuse and technology transfer can be leveraged as a transition infrastructure into aerospace industry. Similarly, other transition agents can be leveraged.

In summary, the above activities will not only accelerate the advancement of reengineering practice, but also contribute to the advancement of megaprogramming and to the improvement of the maturity of software engineering technology.

7 Conclusion

Reengineering has been presented as an engineering problem. As such, reengineering draws from a number of software engineering technologies. Advances in best practice of (re)engineering benefit from innovative research (i.e., creation of new technology solutions); analytical research (i.e., recognition of promising technologies and technology trends); and engineering research (i.e., the maturation of technology into engineering use). Due to its limited resources (currently approximately 50-60 members of the technical staff addressing engineering of software and its education) and its leadership role, the SEI can best contribute to reengineering by advancing best practice through contribution of conceptual frameworks. Evolution of such frameworks for subareas of software engineering has been successfully leveraged through technical leadership in selected community forums. In the context of reengineering, such model outlines a roadmap for improvement in our effectiveness and efficiency to reengineer systems.

8 Acknowledgments

The thoughts expressed in this paper have been evolving within the SEI as well as in the software engineering community. The author attempted to condense them into these pages. I am particularly thankful for Floyd Hollister's inspiration to view reengineering from a problem-solving perspective. The content of this paper was shaped by discussions with and input from Floyd Hollister, Sholom Cohen, John Salasin, Jim Withey, Dennis Smith, Bernd Bruegge from CMU-SCS, Maribeth Carpenter, John Goodenough, and Doug Waugh. I also would like to thank Larry Druffel for his insightful review of an earlier draft of this document.

References

[Arnold 93]	Arnold, R.S., *Software Reengineering*, IEEE Computer Society Press, Los Alamitos, CA, 1993.
[Britcher 90]	Britcher, R.N., "Re-engineering software: A case study," *IBM Systems Journal*, Vol. 29, No. 4, 1990., pp. 551-567.
[Chikofsky 90]	Chikofsky, E.J. & Cross II, J.H., "Reverse Engineering and Design Recovery: A Taxonomy," *IEEE Software*, January 1990, pp. 13-17.
[Przybylinski 91]	Przybylinski, S.R., Fowler, P.J., and Maher Jr., J.H., "Software Technology Transition," a tutorial presented at the 13th International Conference on Software Engineering, Austin, TX, May 12, 1991.
[IEEE 90]	*IEEE Software*, Editor-in-Chief: Ted Lewis, Published by the IEEE Computer Society, Elsevier Science Publishing Co. Inc., New York, NY, May 1990.
[IEEENews]	Software Engineering Technical Committee Newsletter, IEEE Computer Society/TCSE, Editor: Samuel T. Redwine, Jr., Vol. 11, No. 3, January 1993.
[ISST 92]	Witschurke, R., "Wiederverwendung in der Informationsverarbeitung: Re-use, Re-engineering, Reverse Engineering," ISSN 0943-1624, Institut fur Software und Systemtechnik, Universitat Dortmund, Germany, December 1992.
[Leonard-Barton 88]	Leonard-Barton, Dorothy, "Implementation as Mutual Adaptation of Technology and Organization." *Research Policy*, 17 (5), pp. 251-267, October 1988.
[R-EvHa 90]	Rock-Evans, R.; Hales, K.: *Reverse Engineering: Markets, Methods and Tools*. London: Ovum LTD. 1990.
[Santa Barbara 92]	Santa Barbara 1, "Back to the Future Through Reengineering," Joint Logistics Commanders Joint Policy Coordinating Group on Computer Management, Santa Barbara, CA, November 1992.
[SEISymposium 92]	*Capability M...urity Model (CMM) Real-Time Extensions*, Daniel Roy, Software Engineering Symposium, Carnegie Mellon University, Software Engineering Institute, Pittsburgh, PA, September 1992.
[SPC/VCOE 92]	Reuse Adoption Guidebook, SPC-92051-CMC, Version 01.00.03, Produced by the Software Productivity Consortium Services Corporation, under contract to the Virginia Center of Excellence, Herndon, VA, November 1992.

UNLIMITED, UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for Public Release Distribution Unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CMU/SEI-93-SR-5	5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION Software Engineering Institute	6b. OFFICE SYMBOL (if applicable) SEI	7a. NAME OF MONITORING ORGANIZATION SEI Joint Program Office
6c. ADDRESS (city, state, and zip code) Carnegie Mellon University Pittsburgh PA 15213		7b. ADDRESS (city, state, and zip code) HQ ESC/ENS 5 Eglin Street Hanscom AFB, MA 01731-2116
8a. NAME OFFUNDING/SPONSORING ORGANIZATION SEI Joint Program Office	8b. OFFICE SYMBOL (if applicable) ESC/ENS	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F1962890C0003
8c. ADDRESS (city, state, and zip code)) Carnegie Mellon University Pittsburgh PA 15213		10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO	PROJECT NO.	TASK NO	WORK UNIT NO.
63756E	N/A	N/A	N/A

11. TITLE (Include Security Classification)
Reengineering: An Engineering Problem

12. PERSONAL AUTHOR(S)
Peter H. Feiler

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (year, month, day)	15. PAGE COUNT
Final	FROM TO	July 1993	46

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS (continue on reverse of necessary and identify by block number)
software engineering, software-intensive systems, reengineering, reengineering tools,

19. ABSTRACT (continue on reverse if necessary and identify by block number)

This paper discusses a plan that addresses how the Software Engineering Institute (SEI) may assist the Department of Defense (DoD) in reengineering its large software-intensive systems. This plan is based on a view of reengineering as an engineering problem to improve the cost-effective evolution of large software-intensive systems. This view of reengineering, which takes the whole software engineering process into account, fosters a growth path by leveraging promising emerging software engineering technologies. Reengineering also builds on the industry's improvement in its ability to manage the software engineering process, an accomplishment of SEI work in the Capability Maturity Model (CMM) and its key process areas.

(please turn over)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ■ SAME AS RPT □ DTIC USERS ■	21. ABSTRACT SECURITY CLASSIFICATION Unclassified, Unlimited Distribution	
22a. NAME OF RESPONSIBLE INDIVIDUAL Thomas R. Miller, Lt Col, USAF	22b. TELEPHONE NUMBER (include area code) (412) 268-7631	22c. OFFICE SYMBOL ESC/ENS (SEI)

DD FORM 1473, 83 APR EDITION of 1 JAN 73 IS OBSOLETE UNLIMITED, UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS