Factors in Reuse and Reengineering of Legacy Software

David Eichmann

Repository Based Software Engineering Program
Research Institute for Computing and Information Systems
University of Houston – Clear Lake
Houston, TX 77058
email: eichmann@ricis.cl.uh.edu

Abstract: The reengineering and reuse of large legacy software systems can be an expensive, error-prone endeavor. This paper relates experience in a collaborative research project supporting a NASA contractor team attempting just such a project. Risks both potential and realized are discussed, as well as how such a collaboration can be used to drive an academic research agenda. Organizational and social factors proved to be major aspects of the eventual project outcome.

1. Introduction

Organizations are increasingly recognizing their software portfolios as assets to be utilized (Wegner 1984), rather than as obsolescent artifacts that should be discarded at the first opportunity. However, this shift in perception is not without a price. Recovering the corporate memory, domain knowledge, business rules, etc. from this software legacy is not straightforward, and existing tools are only partially up to the task. Furthermore, organizational and social issues can play as much a part of the success or failure of a reengineering project as do the technical issues.

Interest in software representation as a major factor in the software development cycle has steadily been gaining in importance (DARPA 1997). This leads to an evolving perspective of a spectrum of representation expressiveness in software development environments and tools, as shown in Figure1.

This paper surveys some of the key issues in reengineering legacy software systems, presents a hybrid academic-government-industry program designed to mitigate risk entailed in the adoption of reuse and reengineering technology, and then examines an example project involving a portion of the flight design software for the NASA space shuttle program.

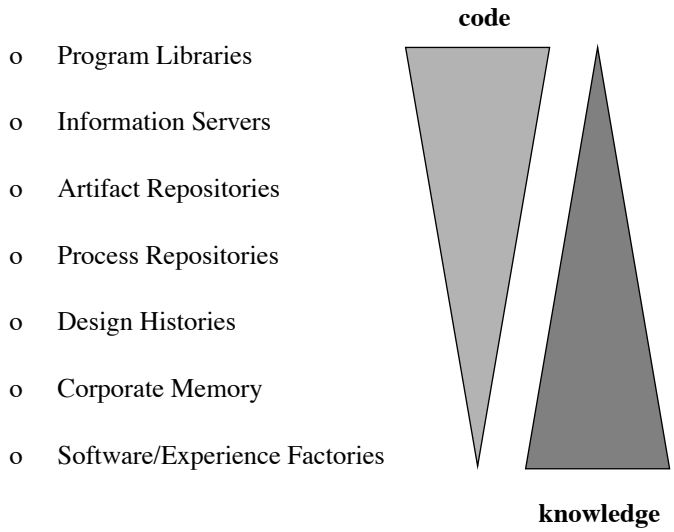

Figure 1: A Spectrum of Representation

2. Intra- vs. Inter-Organizational Reuse

Recent proposals concerning the design of repositories focus on reuse within a given organization. For example, the work by Basili et. al. addresses the distinction between a product organization and a component factory, but presumes that both are in fact just different groups within the same organization (Caldiera 1991, Basili 1992). This approach is similar to that of the Japanese software factory projects (Cusumano 1991, Matsumoto 1989).

A broader context of an industry-based effort in attaining effective reuse and reengineering is more reasonable when considering the nature of multi-contractor large scale software projects. A mature software industry must support both reuse within an organization, minimizing effort across successive projects, and reuse across organizations, minimizing effort across concurrent (and possibly competing) projects. Our focus is to define a framework in which

- an organization can encourage the development of a customer base;
- customers can make informed decisions concerning prospective acquisitions, without the necessity of actually acquiring them;
- a developer organization can protect its capital investment; and
- a consumer organization can equally maintain its strategic position within the market by not releasing information concerning its development plans.

The software industry need not rely upon standards efforts to attain this, rather consumer organizations can be free to choose from a variety of approaches solving any given problem.

3. What is Reengineering?

Chikofsky and Cross define reengineering as “the examination or alteration of a subject system to reconstitute it in a new form and subsequent implementation of that form”

(Chikofsky 1990). This definition clearly is focused on the typical interpretation of the term, the alteration of a software artifact. Arnold, on the other hand, defines reengineering as "any activity that (1) improves one's understanding of software, or (2) prepares or improves the software itself, usually for increased maintainability, reusability, or evolvability" (Arnold 1993). Here we see a broadening of the scope of the term to include issues of comprehension. This interpretation is particularly salient in organizations that have lost the key individuals in which the knowledge regarding the system of interest resides. Finally, GUIDE defines it as "the process of modifying the internal mechanism of a system or program or the data structures of a system" (GUIDE 1989).

There are a number of potential benefits derived from reengineering software. Reengineering can help reduce an organization's evolution risk. It is not uncommon for the only source of information regarding why a software system does what it does to be the software system itself. Reengineering hence can help an organization recoup its investment in software and retain its corporate memory.

Reengineering can make software easier to change and improve its reusability. Incorporation of new design and implementation techniques can create a more modular and composable system, accommodating not only future modifications more effectively (via techniques such as information hiding) but also the recomposition of a system into a new configuration for a new application (via techniques such as refactoring (Opdyke 1992)). Hence reengineering is a catalyst for automating software maintenance, providing a cusp through which an organization can take a fresh direction in its development and support of the organization's software portfolio, as well as acting as a potential agent for enabling reuse within the organization.

A. Reengineering Economics

The decision to reengineer is usually fiscal, rather than technical. Systems become increasingly expensive to maintain as they age and 'bit rot' sets in – the increasing occurrence of errors induced through miscomprehension of increasingly contorted code. However, the criteria from which arises the decision to redevelop or reengineer a system are diverse and extremely fuzzy. Sneed (Sneed 1991) lays out a simple framework for judging the derived benefit.

Management is typically first concerned with the cost of reengineering relative to the cost of redevelopment and the cost of maintenance. The choice between redevelopment and reengineering is principally one involving the difference in value attached to the old and new system, balanced by the risks associated with the anticipated costs of (in)action:

Reengineering Benefit =
[Old_Value – (Reeng_Cost * Reeng_Risk)]
– [New_Value – (Dev_Cost * Dev_Risk)]

The life-expectancy of the existing system relative to the time required to reengineer it and the time required to redevelop it should appear as weights in the risk terms. Note that it is quite possible to derive negative benefit if the cost or risk of reengineering the system is sufficiently high compared to the costs of just starting over from scratch.

The choice between continued maintenance and reengineering is structured similarly:

Reengineering Benefit =
[Old_Maint_Cost – Reeng_Maint_Cost]
+ [Old_Value – (Reeng_Cost * Reeng_Risk)]
– [New_Value – (Dev_Cost * Dev_Risk)]

factoring in an additional term addressing the expected difference in maintaining the old versus reengineered system. The *added* value of reengineering a new system must be balanced against the *current* value of the present system. This is effectively the risk of reengineering relative to the risk of a new development and the risk of doing nothing.

B. Reengineering Risks

In order to factor risk into an economic decision model such as the one above, it is first necessary to understand just where risks lie, and the form that they might take. Arnold elaborates the following framework for categorizing risk in the reengineering of software (Arnold 1993):

- Process risks
  - Extremely high manual reengineering costs
  - Cost benefits not realized in required time frame
  - Cannot economically justify the reengineering effort
  - Reengineering effort drifts
  - Lack of management commitment to ongoing reengineering solution
- Personnel risks
  - Programmers inhibiting the start of reengineering
  - Programmers performing less effectively to make an unpopular reengineering project look less effective
- Application risks
  - Reengineering with no local application experts available
  - Existing business knowledge embedded in source code is lost
  - Reengineered system does not perform adequately
- Technology risks
  - Recovered information is not useful or used
  - Reverse engineering to representations that cannot be shared
  - Reengineering technology inadequate to accomplish reengineering goals
- Tool risks
  - Dependence on tools that do not perform as advertised
  - Not using installed tools
- Strategy risks
  - Premature commitment to a reengineering solution for an entire system
  - Failure to have a long-term vision with interim goals
  - Lack of global view: code, data, process reengineering
  - No plan for using reengineering tools

This list compresses into three core concerns: the organizational will to follow a reengineering project through to completion, the skill and commitment of the technical staff involved, and the existence of the information necessary to accomplish the task. Each of these concerns will arise later in the paper as we discuss our example project.

4. The Repository Based Software Engineering Project

The Repository Based Software Engineering (RBSE) project is a research and development program whose mission is to provide a technology transfer mechanism to improve NASA's software engineering capability (Eichmann 1995). RBSE is sponsored by the NASA Technology Utilization Division and is administered by NASA's Johnson Space Center and the Research Institute for Computing and Information Systems (RICIS), a part of the University of Houston – Clear Lake. The purpose of RBSE is the support and adoption of software reuse through repository-based software engineering in targeted sectors of industry, government, and academia. The project consists of two principal activities, a repository initiative and reuse engineering initiative, as shown in Figure 2.

A. The Repository Initiative

The repository initiative has focused upon the creation, demonstration and deployment of tools supporting reuse efforts. The Multimedia Oriented Repository Environment (MORE) (described below) is the major result of this aspect of the project. MountainNet, Inc., a small technology firm, operates the Electronic Library Services and Applications (ELSA) repository as the demonstration portion of the repository initiative. The ELSA Repository, an instance of MORE, contains a comprehensive collection of information about all aspects of software engineering and the software development life cycle, as well as an extensive collection of public domain software with related documentation provided to support software development efforts. In addition to software and related documentation contained in the reuse library, ELSA contains information related to conferences, tools and environments, publications, and references.

B. The Reuse Engineering Initiative

The reuse engineering initiative focuses on the support of specific software projects as they transition to a reuse oriented development and maintenance paradigm. Figure 2 shows our primary pilot, a collaboration with the Reusable Object Software Engineering (ROSE) project. There were three organizations involved in ROSE, the Rockwell Space Operations Company (RSOC) (including UNISYS as a major subcontractor), the Software Technology Branch of the Information Systems Directorate of NASA (NASA/STB) and the RBSE research group.

The work by Basili et. al. on experience factories addresses the distinction between a product organization and a component factory, but presumes that both are in fact just different groups within the same (potentially virtual) organization (Caldiera 1991, Basili 1992). Our organizational model is similar to that of the experience factory with one key distinction – the experience factory is intra-organizationally focused; we are addressing inter-organizational issues as a critical aspect of development team support. This inter-organizational focus is derived from the multi-contractor climate that NASA/JSC software is developed within. The intention is to address the adoption of a corporate memory paradigm for NASA as the 'corporation.' NASA projects are long-lived – and frequently outlive a given contractor. Information loss during contract transitions can be severe, particularly with regard to design and evolution rationale. Serving as project memory and providing a channel for initiatives in

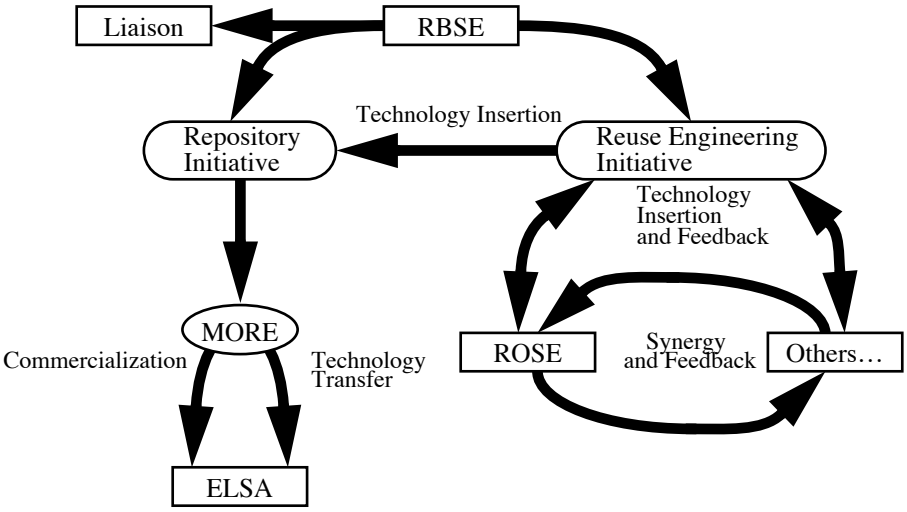

Figure 2: The Program Architecture

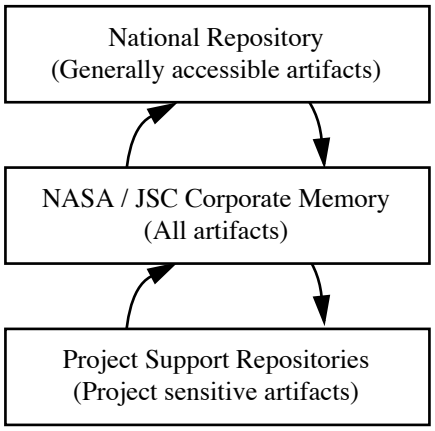

Figure 3: Repository Capability Layering

standards and practices (e.g., efforts such as the Experience Factory initiative in the NASA/ GSFC Software Engineering Lab) have been key drivers for our work with ROSE.

C. RBSE Technology Transfer

The reuse engineering initiative is intended to address the broader context of an industry-based effort in attaining effective reuse and in the adoption of new technologies. The software industry need not rely upon standards efforts to attain this, rather consumer organizations can be free to choose from a variety of approaches solving any given problem. This leads to Figure 3, three perspectives of a single repository. We envision RBSE as an infrastructure for the spread of software engineering through the NASA community – civil ser-

vants, contractors, researchers – and out to the country as a whole. Both artifacts and processes are a critical aspect of that infrastructure. Our initial collaborations have involved support of discrete development teams. We're moving towards a corporate memory paradigm and our pilot collaboration with development teams (e.g., ROSE) allows us to:

- interact with development teams as a source of information and technology transfer and as monitors of activities, capturing key technology for transfer to other projects;
- represent the information gathered;
- match project expectation and requirements; and
- capture actual return-on-investment (part of the project sensitive nature of the ROSE experience base includes costs models).

5. The Reusable Object Software Engineering (ROSE) Project

The ROSE project was a collaborative effort to reengineer a large legacy system – the Flight Analysis and Design System (FADS) for the Space Shuttle program. Quoting from the ROSE project description:

> "ROSE is being developed to provide an economical and effective approach to reengineering and maintaining [Shuttle] Flight Design and Dynamics (FDD) systems. The software that supports FDD operations is the result of an evolutionary process that began with the planning of the mission of a reusable orbiter and has continued to this day. While the software was developed under a variety of methodologies and languages, it has a couple of similar characteristics; it is difficult to use and expensive to maintain. The Flight Design and Analysis System (FADS) development was originally targeted to address the software in addition to upgrading the hardware platform. For a variety of reasons most of the software was rehosted to the new platform with minimal change. Therefore, the problems associated with operations and maintenance of the software persist.
>
> The ROSE project is intended to bring the existing FDD software in to a consistent architecture and design framework. To accomplish this, the existing functionality is being reengineered into a modern architecture and design based on Object Oriented Software Engineering (OOSE) techniques and implemented in a language that supports those techniques., and accomplished through a repeatable defined process. The information associated with the architecture, design and process is stored in a repository that is available to and maintained by sustaining engineering.
>
> The ROSE project first performed a pilot project to train a team, verify the return on investment, refine the process and methods, and establish the tools to automate them. The success of the Pilot determined the extent and feasibility of proceeding with the main project."

The result of an evolutionary process that began with the initial planning of a reusable orbiter, FADS is composed of 2.5 MSLOC of mostly FORTRAN that has undergone several 'minimum impact' changes in a variety of methodologies and languages and several programs with overlapping or redundant functionality. Not surprisingly, FADS is expensive and difficult to maintain. A mid-project expansion of scope to include the Mission Operations Center (MOC) Trajectory System took the total legacy system size for ROSE to 4 MSLOC.

A reduction in the sustaining engineering costs was major focus of the project and the project was sold to NASA management specifically on expected return on investment. The strategy chosen involved

- decomposing the system into ‘delivery increments’ for phased deployment;
- adoption of a dual-lifecycle distinguishing domain engineering and applications engineering activities; and
- risk mitigation through
  - a preliminary pilot to demonstrate proof-of-concept,
  - a secondary pilot to baseline project estimates, and
  - an evolutionary spiral approach to each delivery increment.

A. The Realities of Reengineering

The time line appearing below is derived from the NASA technical monitor’s monthly status reports to upper management through the duration of the ROSE project.

- 1/1/93 – Pilot start
- 6/7/93 – 30% behind schedule and 30% under budget
  - procurement the big, unanticipated, and recurring risk
- 7/30/93 – 21% behind schedule and 8% over budget
  - platform availability and learning curve
- 8/31/93 – Pilot review, ‘go’ given for secondary pilot
- 4/5/94 – Pilot review, ‘go’ for project
  - 88% reduction in application specific code
  - 86% reduction in application complexity (# of logic paths)
- 9/9/94 – NASA merges ROSE with MOC rehost effort
  - LORAL added to mix, team size doubles
- 3/6/95 – DI-1 (Navigation) delivered
- 6/95 – massive replanning of Trajectory-Off-the-MOC (TOTM)
- 7/31/95 – team retargeted at TOTM
  - Original goals pruned back to Orbit/Navigation
  - DI-2 (DOPS/LANDOPTS application) still on track
- 12/95 – conflicts between architectural interfaces/approaches arise
- 2/96 – massive replanning
- 3/12/96 – ROSE-TOTM cancelled

B. Risks Realized, Pure ROSE

The project time line clearly carries evidence of the effect of many of the issues discussed in Section 3.B. Repeated staffing increments coupled with decreasing training budgets resulted in an ‘oldbie/newbie’ mentality, where oldbies (team members involved in the original pilot and the associated intensive training) maintained a distinct sub-group identity.

The administrative entanglements of government contracting work resulting in procurement playing a major factor in pilot project schedules, even though the procurement was for readily accessible workstation technology. Once pay-back started, NASA couldn’t resist solving a different problem with the team, before they finished the current task.

Finally, much of our perception regarding process proved to be somewhat idealistic. Process models frequently were too prescriptive and too slow to adapt – encouraging workarounds and divergence. Resistance to process adherence proved to be difficult to overcome once it took root in increasingly harried team members.

C. Risks Realized, ROSE–TOTM

The transition from pure ROSE to ROSE-TOTM proved to be the watershed for the project. The competitive environment within the JSC contractor community presented significant challenges to project activities. Too many contractors with a serious stake in the outcome (particularly with respect to future work) resulted in turf issues frequently cascading into technical decisions, particularly in the case where ROSE team lacked the domain knowledge regarding the real-time applications, and were dependent upon the MOC team for that knowledge. The situation was further exacerbated by NASA opening negotiations for a Consolidated Shuttle Operations Contract (CSOC) during the later phases of the project. Cancellation appears to have been driven as much by the CSOC deadline (eventually awarded to USA) as by technical issues. There was little enthusiasm for having a major project underway during a contract change-over.

The three core concerns discussed at the end of Section 3 are clearly identifiable in the realities of the ROSE project. The aggregate organization of NASA and its contractors succumbed to the temptation of fixing one problem by reassigning a team developed and trained for another problem, and thereby created an organizational climate where project goals were not universally adopted or agreed upon. Budgetary limitations resulted in an uneven skills base and contention for resources. Finally, key system knowledge was eventually seen as a competitive advantage, rather than as a shared resource.

While it appears that ROSE failed to achieve its goals, it did in fact demonstrate that it was possible to achieve a significant reduction in system size and complexity, even on a very large scale, through a careful mix of people and technology. What must come next is a proper respect for the social as well as technical aspects of software projects (Kling 1980).

6. ROSE / RBSE Support Specifics

The RBSE reuse engineering group was directly involved with the ROSE team from its formation, participating both in its initial planning, the presentation of the project to Rockwell and NASA management, and its day-to-day activities. ROSE began as a six-month, twenty-person initial pilot to demonstrate proof-of-concept for the approach. The sixth month review resulted in the expansion of the team to forty people and extension of the pilot to a full year. This year long activity generated enough information to assess probable success of the full project, and in the year end review, the team was given the entire flight analysis and design system as its scope, with a set of phased delivery increments to successively replace major FADS subsystems. In particular, RBSE personnel participated in:

- process and methodology definition;
- reverse engineering;
- forward engineering;
- experience capture and repository representation; and
- data management.

The remainder of this sub-section discusses each of these areas in more detail, with particular emphasis placed upon the roles played by RBSE personnel.

A. Process and Methodology

The ROSE personnel came mainly from what are referred to locally as 'disciplines' – predominantly trained as engineers in the fields of aerospace, aeronautics, etc. – with little formal training in large-scale software development. The RBSE team has done a great deal of mentoring concerning general principles of the development methodologies, life cycle models and process definition as the ROSE team as moved through what can only be termed a dramatic capability improvement. Ad hoc approaches to legacy system maintenance have been replaced with a spiral development model focusing on risk assessment and mitigation, definition and execution of an explicit life cycle model (a variant of a model designed by one of the members of the RBSE team) for each iteration of the spiral, and the use of domain/ application engineering as an architectural context.

B. Reverse Engineering

ROSE employed a mix of internal techniques and commercial tools in their reverse engineering of the legacy FORTRAN code. RBSE's involvement with this aspect of the project was in the area of process model definition; ensuring that the expertise of the developers of the techniques was documented in a manner sufficient for the rest of the ROSE team (and subsequent teams) to use the techniques productively and repeatably.

The challenges encountered in absorbing 2.5 million (and eventually 4 million) lines of legacy FORTRAN have provided very useful data concerning scalability of these techniques and tools. The use of non-standard approaches to modularity in the existing FORTRAN has been major impediment to effective use of many of the tool suites.

C. Forward Engineering

As mentioned above, the RBSE team was substantially involved with ROSE process definition, and the majority of effort relating to this was in the forward rather than the reverse portions of this reengineering project. The selection of domain engineering / application engineering as the paradigm to be employed in the creation of the new system was driven strongly by a large amount of redundancy both within FADS itself and with other closely related systems within NASA/JSC.

The additional effort expended in generating a generic solution rather than a point solution for FADS is already perceived as beneficial, and was key in the decision to expand the project scope to include the Mission Operations Center Trajectory System. RBSE's role through this was one of conceptual definition and mentoring the ROSE team in how domain-oriented paradigms could be blended with the commercial OO tools and techniques that were available.

D. Data Management

A large percentage of the systems related to the ROSE domain of shuttle flight analysis and design exclusively employ sequential files for data management. A data management study was commissioned as part of the initial pilot to consider how database management systems might be incorporated into the project. This study resulted in the definition of a set

of persistent class interfaces that have become the data foundation of the overall system architecture. This 'API' supports plug-compatibility of sequential file interfaces, relational databases and object databases, both improving system portability and mitigating risk related to immature technology (in the case of the object databases).

E. Experience Capture and Repository Representation

The points mentioned in the preceding subsections clearly involve significant capture of the experiences of the ROSE team as they made the (sometimes painful) migration from a traditional organization and paradigm to a self-empowered, domain-centric, process-oriented perspective. Much of this transformation was readily accepted by ROSE team members – the switch from FORTRAN to C++ and the use of CASE environments in particular. Other aspects were more problematic. The concept of an executable process model was accepted in principle, but the constraints placed upon developers by available enactment tools and the requirements for documenting modifications occurring in iterations of the spiral resulted in a fair amount of (frequently warranted) dissension.

The buy-in on the concept of capturing artifacts as created was much higher in comparison. ROSE operated an instance of MORE and developed the glue to automatically migrate meta-data out of their configuration management system and into the repository system as artifacts were maintained. This type of tight integration of tools is frequently seen as beneficial to the acceptance of metrics programs (Selby 1991), and a similar effect may have occurred here.

A substantially more important aspect of the repository integration, however, is likely to be the nature of information capture and retention during all aspects of the project, not just during its final phases. The ROSE team used the World Wide Web and (more recently MORE) to keep its numerous members abreast of developments across the various subteams. This directly supports our goal of corporate memory with respect to software development, and promises to be one of the more interesting areas in which to continue our work.

7. RBSE / ROSE Lessons Learned

We have learned a great many things in our collaboration with the ROSE team:

- Even large organizations require support in the *how* of software reuse as much as the *what*.
- Experience capture needs to be *injective* and *proactive*, with RBSE collaboration occurring in the development group's environment (we were a part of their team, and involved in all team functions).
- Maintaining that presence requires 2 – 3 people to cover the various meetings for a development group of 20 – 50 and to assess and characterize artifacts deriving from that contact. This places some very non-traditional demands upon faculty and research assistants, who are frequently used to setting a more flexible work schedule.
- Supporting teams transitioning to environments dramatically different from that which they are accustomed to requires multiple modes of interaction:
  - Experience/Technology Transfer – Pedagogy (the basics of teaching the concepts is obvious, but the mentoring of ROSE team members in the direct transition of application of those concepts to extremely complex problem domains was also critical);

– Repository Services – Technical Support (reverse engineering / reengineering, process (as product), domain engineering and analysis, formal artifact certification); and
– Tool Crafting (architecture recovery, component recovery, process modeling and enactment). Teams buy in to tools far easier than they do processes, and when things get hot, there's a great deal of 'idealogical backsliding.' The tool crafting activity provides a means of keeping communications channels open.

Each of these modes depends significantly upon the other two for successful utilization. A balanced technology transfer group must be able to train, operate and create within the environment that the mentored team employs.

We are at a point in our collaborative work where we are examining how we might scale our activities to interact with multiple development groups. One of the major challenges that we face in doing this is how scarce staff expertise (frequently any given category of activity is covered by a single individual) can either be time-shared across collaborations or 'cloned' by the hiring and training of additional personnel.

8. Research and Development Areas

The lessons that we've learned with the ROSE team have also directly impacted our research activities. Pure academic research has frequently been the target of criticism from practitioners for its over-emphasis on theory without regard for practical application of that theory. We believe that practical requirements can actually invigorate research, rather than hold it back. This has been particularly true for our work in repository technology and process modeling. We expect the same to hold true with our work in domain modeling, slicing, and measurement as our prototype tools in these areas mature to the point of handling ROSE-sized problems. Our major research spin-offs included:

- **MORE: the Multimedia-Oriented Repository Environment** – MORE is a meta-data based repository system based completely within the World Wide Web. Building MORE, shown in Figure 4, involved a complete redesign of our previous repository to support the following goals (Eichmann 1994a, Eichmann 1994b):
  – an adaptable group definition mechanism for managing access to proprietary sub-collections of assets;
  – the World Wide Web as our sole user interface (including all administrative access);
  – integration of other Internet resources into our user interface through the use of URLs in repository data.

  The malleability of our meta-data scheme has been a key aspect of its acceptance by a number of industrial and government projects.
- **Reengineering Process Modeling** – This involves capture and modeling of the ROSE reengineering process (Perera 1993). We initially attempted to employ some of the commercial process modeling and enactment tools in our modeling of the ROSE process. This quickly became intractable for rather fundamental reasons – our tools were very good at modeling a process, reasonable good at enacting that process, and virtually unusable at *evolving* that process. When nothing is static, most particularly the process, this situation is untenable. We are currently designing a process modeling tool that handles versions of processes as readily as revision control systems handle versions of source files. Our prototype will also address the typical distribution of responsibility among a variety of contractors, none of which share any more information than is necessary for project purposes. This competitive atmosphere leads to strong requirements for information hiding in the

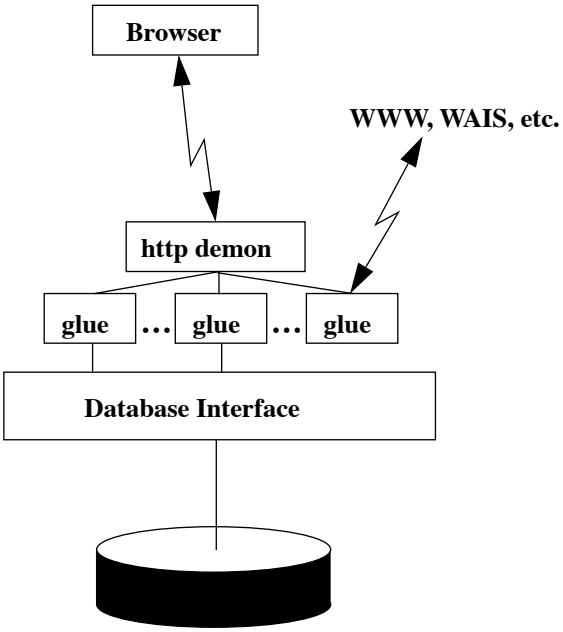

Figure 4: The MORE Architecture

entire development environment, but most particularly in the case of process, where the raw data relating to effort expended and staff expertise lurks.

- **Domain Modeling and Analysis** – Capturing and modeling the common architectural features and domain models has only recently become a key aspect of mature engineering of families of systems. Our work to date on ROSE has been primarily in the capture of as-is models of the existing domains (McKay 1994). We are naturally interested then, given the long life cycles of human-rated systems in our environment, on how these domain models themselves are evolved and validated.
- **ION (the Intermediate Object Notation)** – Bridging the gap between the OO design notations (Booch, Rumbaugh, etc.) and OO programming languages has proven to be of substantial effort when dealing with very large scale systems. ION attempts to provide an intermediate point in the transformation from conceptual design to concrete implementation – where design decisions have been made, but the gory details of operation are still too obscuring for the developer (Atkinson 1993, Gunawardena 1995, Mishra 1993). ION also promises to be of use at the boundary between domain architectures and the components and subsystems used in instantiating those architectures.
- **Interface Slicing** – Our work on interfaces slicing centers around using program slicing techniques to tune complex assets for client requirements (Beck 1993a, Beck 1993b). The issue here is the computational complexity and micro (i.e., statement level) focus of more traditional slicing techniques. Interface slicing operates at the macro (i.e., module specification) level, substantially improving the complexity of dependency analyses. Our goal with this work is to be able to make efficient use of highly flexible generic artifacts, where only the required functionality is propagated into the new system.
- **Reusability Metrics** – With no commonly accepted measure of an artifacts reusability at

hand, we found ourselves in need of creating one. Rather than tackle the sometimes arcane world of polynomial construction, we instead decided on a somewhat novel approach – the use of neural networks in the construction of measures of the reusability of an asset (Boetticher 1993a, Boetticher 1993b). Preliminary results have been quite promising, and a ROSE team member has also expressed interest in becoming involved with this in the formation of a maintainability measure.

9. Future Work

Four distinct research perspectives have arisen out of the work described here. The first involves the creation of a ROSE archive. We have a full dump of almost all of the ROSE artifacts – documents (meeting minutes through baselined designs), CASE artifacts, program sources, etc. We are currently constructing a first cut classification of the material in MORE for NASA evaluation. Once approved, the archive will be a open source of reengineering source material and a shared value-added testbed.

The second involves the extension of MORE from a simple repository system to an open environment for information management and retrieval, based upon agents and ontological reasoning. We are currently using the Web as our target domain.

The third developed out of our observation of ROSE team members attempt to use existing tools on FADS with only limited success. Large software systems generate far too much information for current tool technology to readily manipulate. We are investigating cataloging of ROSE-style artifacts for inclusion into MORE using program comprehension and slicing techniques. Hybridization of the two techniques should mitigate some of the computational complexity of recognition algorithms.

Finally, the domain engineering approach used in ROSE has interesting connections into the area of design patterns. Design patterns are emerging as a technique for capturing and transferring design expressions - recurring 'statements' used by a design group. Much of the ROSE domain modeling operates at two levels, at the level of architecture – which is well suited to domain engineering techniques, and at the level of expression – better suited to pattern language techniques. We are preparing to employ patterns as a means of providing a design rationale for MORE as a means of testing the boundaries between domain architecture and pattern language (Irving 1996, Weisskopf 1996). Given the guaranteed nature of research assistant turn-over (due to graduation), this should prove an excellent test of 'organizational memory.' Patterns also have interesting connections to the cliches that drive program recognition. Having a catalog of patterns employed by designers in a given domain offers a means of automating cliche definition. Building a model of pattern-cliche-code semantics would allow us to retarget our Web agent architecture to understanding legacy systems.

10. Conclusions

NASA/JSC's software portfolio contains a small but extremely high profile collection of very large (multi-MSLOC) systems, many of which are human-rated. These systems are long-lived, evolving and they frequently interlock. The cost of maintaining these systems as they age has led to projects such as ROSE in order to reduce effort and improve quality.

The RBSE team created a strongly synergistic relationship with the ROSE team, and we

feel that this approach is one that can be employed by other software engineering research groups in grounding their efforts in practical realities while generating a multitude of challenging research questions.

MORE is an example of the way in which concepts can win over the team as a whole: the notion of Web-integrated repositories for documents and artifacts is apparently vastly popular. The ROSE WWW server became the nexus of process documentation, repository access and teaching material availability. The rapid adoption of MORE is indicative of the appeal of the simple, easy to use and distributed environment that the ROSE team chose to use, often at the detriment of costlier, harder to use – although equally functional – tools. In this respect, ROSE team members found themselves using a toolset comprised of a great many free tools at the same time as investing in top-notch commercial products.

It is sometimes difficult to get buy-in – researchers and practitioners have different viewpoints on the subject. Useful techniques and concepts can fail to make the transition simply because researchers cannot articulate their usefulness in terms that practitioners can relate to. Our exposure to the industrial environment has helped improve the relevance of our activities by forcing us to ask "and what would ROSE benefit from this?" However, the ROSE team organization being quite tight, it is was often difficult to passively observe goings on. A physical presence at meetings and inclusion on internal mailing-lists was required in order to witness the activities performed. The close-knit teams make the team boundary opaque to outside observer.

Acknowledgments

A great many people contributed in numerous ways to the issues related in this paper. Apologies for not listing you all. UHCL – E. T. Dickerson, Charles McKay, Colin Atkinson, Mike Weisskopf, Sharon White, Carl Irving and numerous other research assistants. I-Net – Terry MacGregor, Dann Danley, Jim Helm. NASA/JSC/STB – Ernie Fridge, Bob Savely, Charles Pitman, Michel Izygon, Alan Plumb. NASA/JSC/MOD – Cheryl Andrews. RSOC – Bill Cottrell, Jerry Gerstman, Dick Osburn, Dave Forrest, Terry Hodgson and the rest of the ROSE crew.

References

Arnold, R. S. (1993). "A Road Map Guide to Software Reengineering Technology," in *Software Reengineering,* R. S. Arnold (ed.), IEEE Computer Society Press.

Atkinson, C. and Izygon, M. (1993) *An Implementation-Oriented Notation for the Graphical Representation of Object-Oriented Programs,* University of Houston – Clear Lake.

Basili, V. R., Caldiera G., and Cantone, G. (1992) "A Reference Architecture for the Component Factory," ACM Transactions on Software Engineering and Methodology, vol. 1, no. 1, pp. 53-80.

Beck, J. and Eichmann, D. (1993) "Program and Interface Slicing for Reverse Engineering," jointly appearing in *Working Conference on Reverse Engineering* (Baltimore, MD, May 21-23) pp. 54-63 and the *International Conference on Software Engineering* (Baltimore, MD, May 17-21), pp. 509-518.

Beck, J. (1993) *Interface Slicing: A Static Program Analysis Tool for Software Engineering,* Ph.D. dissertation, West Virginia University.

Boetticher, G. and Eichmann, D. (1993) "A Neural Network Paradigm for Characterizing Reusable Software," *Proc. of The First Australian Conference on Software Metrics* (Sydney, Australia, November 18-19) pp. 41-50.

Boetticher, G., Srinivas, K. and Eichmann, D. (1993) "A Neural Net-Based Approach to Software Metrics," *Fifth International Conference on Software Engineering and Knowledge Engineering* (San Francisco, CA, June 16-18). pp. 271-274.

Caldiera, G. and Basili, V. R. (1991) "Identifying and Qualifying Reusable Software Components," IEEE Computer, vol. 24, no. 2, pp. 61-70.

Chikofsky, E. and Cross, J. H. (1990) "Reverse Engineering and Design Recovery: A Taxonomy," IEEE Software, vol. 7, no. 1, pp. 13-17.

Cusumano, M. A. (1991) *Japan's Software Factories,* Oxford University Press, New York, NY.

Defense Advanced Research Projects Agency, Evolutionary Design of Complex Software Program, http://www.ito.darpa.mil/ResearchAreas/EDCS.html.

Eichmann, D., McGregor, T.and Danley, D. (1994) "Integrating Structured Databases Into the Web: The MORE System," *First International Conference on the World Wide Web* (Geneva, Switzerland, May 25-27) pp. 369-378.

Eichmann, D. (1994) "MORE: A Case Study in Reengineering a Database Application for the Web," *Second International World-Wide Web Conference: Mosaic and the Web* (Chicago, IL, October 18-20).

Eichmann, D., (1995) "The Repository Based Software Engineering Program," *Fifth Systems Reengineering Technology Workshop* (Monterey, CA, February 7-9).

GUIDE, (1989) *Application Reengineering,* GUIDE Pub. GPP-208, GUIDE International Corp., Chicago.

Gunawardena, S. (1995) *Mapping Problem Oriented Notations to the Implementation Oriented Notation (ION),* Master's thesis, University of Houston – Clear Lake.

Irving, C. and Eichmann, D. (1996) "Patterns and Design Adaptability," *Third Annual Conference on The Pattern Languages of Programs* (Allerton Park, Illinois, Sept. 4-6).

Kling, R. (1980) "Social Analyses of Computing: Theoretical Perspectives in Recent Empirical Research," Computing Surveys, vol. 12, no. 1, pp. 61-110.

Matsumoto, Y. (1989) "An Overview of Japanese Software Factories," in *Japanese Perspectives in Software Engineering,* Y. Matsumoto and Y. Ohno (eds.), Addison–Wesley Publ., Singapore.

McKay, C. (1994) "Position Paper for the ARPA Sponsored Domain Modeling Workshop," *ARPA Domain Modeling Workshop* (Fairfax, VA, September 8-9).

Mishra, R. R. (1993) *ION – an Implementation Oriented Notation for Graphical Representation of OO Programs,* Master's thesis, University of Houston – Clear Lake.

Opdyke, W. F., (1992) *Refactoring Object-Oriented Frameworks,* Ph.D. thesis, U. of Illinois at Urbana-Champaign.

Perera, D. N. (1993) *A Software Reengineering Process for an Object Oriented Environment,* Master's Thesis, University of Houston – Clear Lake.

Selby, R. W., Porter, A. A., Schmidt, D. C. and Berney, J. (1991) "Metric-Driven Analysis and Feedback Systems for Enabling Empirically Guided Software Development," *Proc. Thirteenth In. Conf. on Software Engineering,* (Austin, TX, May 13-16), pp. 288-298.

Sneed, H. M., (1991) "Economics of Software Re-engineering," Journal of Software Maintenance: Research and Practice, pp. 163-182.

Wegner, P., (1984) "Capital-Intensive Software Technology." IEEE Software, vol. 1, no. 3.

Weisskopf, M., Irving, C. W., McKay, C. W., Atkinson, C. and Eichmann, D. (1996) "Maintenance In a Dual-Lifecycle Software Engineering Process," *Int. Conf. on Software Maintenance, (*Monterey, CA November 4-8).