A Requirement-Based Approach to Data Modeling and Re-engineering

Alice H. Muntz
Hughes Information Technology Corporation
MS SC/S64/C410
PO Box 92919
Los Angeles, CA 90009
ahmuntz@hac2arpa.hac.com

Christian T. Ramiller
Hughes Information Technology Corporation
MS SC/S64/C410
PO Box 92919
Los Angeles, CA 90009

Abstract

This paper reports on the managerial experience, technical approach, and lessons learned from re-engineering eight departmental large-scale information systems. The driving strategic objective of each project was to migrate these systems into a set of enterprise-wide systems, which incorporate current and future requirements, drastically reduce operational and maintenance cost, and facilitate common understandings among stakeholders (i.e., policy maker, high-level management, IS developer/maintainer/ end-users). A logical data model , which contains requirements, rules, physical data representation as well as logical data object, clearly documents the baseline data requirements implemented by the legacy system and is crucial to achieve this strategic goal.

Re-engineering products are captured in the dictionaries of a CASE tool (i.e., in the form of a business process decomposition hierarchy, as-is data model, normalized logical data model, and linkages among data objects) and are supplemented with traceability matrices in spreadsheets. The re-engineered data products are used as follows: (1) migration of the legacy databases to relational database management systems, (2) automatically generation of databases and applications for migration from mainframes to client-server, (3) enterprise data standardization, (4) integration of disparate information systems, (5) re-documentation, (6) data quality assessment and assurance, and (7) baseline specifications for future systems.

Proceedings of the 20th VLDB Conference Santiago, Chile, 1994

1. Introduction

Re-engineering is the process of analyzing, upgrading, and integrating enterprise information systems to meet the expanded operating requirements of the present, as well as to prepare a sound base for effectively meeting future needs. In this paper, we report our experience, key technical approach, and lessons learned from applying our framework ([MH94], [AHR94], [AHR93], [HI93a], [HI93b]) to eight large scale legacy information systems, each containing several million lines of code, and thousands of data elements.

Our requirement-based re-engineering approach differs from ([P&B94], [HDA87], [M&M90], [M&S89], [NEK94], [MNBBK94],[HCTJ93]) that we couple our extended entity-relationship modeling methodology and tools with model integration processes to produce a normalized data model. In a re-engineered data model, the physical data objects, pertaining to the physical schemas, link to the relevant logical data objects, external data objects, business requirements, and system rules.

Why Requirement-Based Data Modeling and Re-engineering?

A requirement-based re-engineering approach should be used when the information infrastructure of an enterprise is out-of-control and is not meeting its goals. Typical symptoms are:

- people can't share or integrate data across the enterprise
- data can't be combined from multiple sources
- the information people need is in a system somewhere else, not accessible to them
- the Information System (IS) staff can't support the existing infrastructure because of obsolete platforms or programs designed, built, and supported by one person
- the IS department charges are skyrocketing, but people still can't get the information they need (**what**) to conduct the business; when they want it, **how** they want it, or in the **form** they want it
- the information systems don't support enterprise strategic planning or tactical decision support
- information and application requirements have not been mapped to business functions

IS Development Life Cycle Phase	Information Technology (IT) Task	Use of Requirement-Based Re-Engineering Products
Business Requirements	• Functional Economic Analysis • Business Process Improvement • Enterprise Data Architecture • Enterprise Data Model • Cross-functional Model Integration • Enterprise Model Integration • Data Element Standardization • Enterprise Data Warehouse • Enterprise Data Repository	• understanding requirements of as-is business processes /data models and how IS supports users and business • inventory of baseline requirements implemented in the current IS • composition of the logical data dimension and the physical data dimension of enterprise data architecture and enterprise data model • integration of tactical and operational data models among business functions • integration of strategic, tactical, and operational data models for all business functions • uniform name for the same data object to facilitate reuse • same name, meaning, and usage for the sharable data instances and objects
Information system Requirements	• Enterprise IT Consolidation Planning • Enterprise IS Security	• identification of duplicating or similar databases for the same business functions • understanding requirements of as-is business processes /data models and how IS supports users and business
Software/Database Requirements Preliminary Design Detail Design	• Forward Engineering • Object Engineering • Data Quality Engineering	• elimination of obsolete requirements, modification of changed requirements, and addition of new requirements • data objects formulated from data usage, processing rules, attributes, and entities of logical data model • data quality requirements represented in the re-engineered logical data model
Implementation	• Database Generation • Screen Generation • Report Generation	• databases and tables automatically generated by CASE tools using re-engineered data models • screen/report design and implementation via CASE tools using re-engineered data models
Unit Test Integration	• Data Migration	• linkage between the legacy data elements to the as-is normalized data model used by the data extraction program • linkage between the to-be data element to the as-is normalized data model used to store the extracted legacy data to the to-be database tables
Operational and maintenance	• Data Quality Assurance • Data Administration • Training • Maintenance	• data quality requirements specified in the re-engineered data model • data requirements specified in the re-engineered data model • understanding of requirements and how the IS implements the requirements

Table 1. Use of Requirement-Based Re-Engineering Products

- modifying one application causes errors, aborts, and / or erroneous information in a different application

A requirement-based data modeling and re-engineering approach should be used to re-document or map business requirements, functional requirements, and data requirements to architecture, design, and implementation. As depicted in Table 1, re-engineered data models enable identification of obsolete portions of applications, outstanding unfulfilled requirements, applications that require changes or consolidation based on new functional requirements to meet current and future business needs. The re-engineered design will provide a basis for developing a plan to migrate reusable applications to the new application architecture and technology infrastructure.

As a part of our re-engineering approach, we resolve data conflicts (e.g., synonyms, homonyms) during data modeling and model integration. In section two, we describe our approach to identify data conflicts and classify data conflicts. Depending on the scope of the re-engineering effort, various levels of integration will occur. We developed an integration taxonomy to guide our integration process for data modeling and re-engineering legacy information systems. Sections 3 describes this integration taxonomy. Section 4 describes our requirement-based data modeling approach. Section 5 uses an example to illustrate our model integration

approach. Finally, in the last section, we summarize lessons learned from these re-engineering experiences.

2. Five Data Conflicts Types

When systems are to be re-engineered or consolidated, functional and data conflicts, and data duplication need special attention, to determine the impact of selecting one system's implementation over another. Data conflicts fall into one of 5 general types.

Synonyms are data element names which are different but which are used to describe the same thing in the various systems. The "same thing" refers to (a) the same critical attribute of a real-world thing, or (b) the same real-world thing. An example is shown in Figure 1. Here the term ACTIVITY within the pay system is used to identify what is called a UNIT or ORGANIZATION within the personnel system. This conflict was uncovered by noting the similarity in associations as well as in definition.

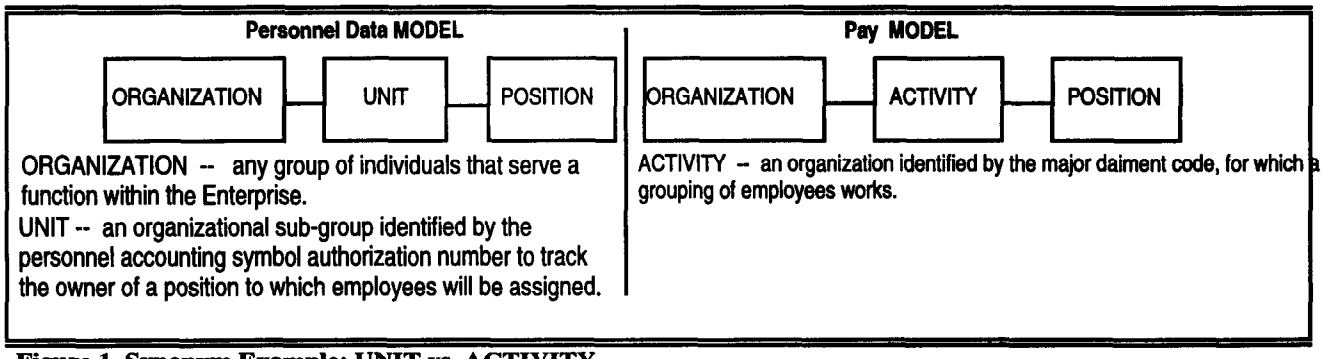

Figure 1. Synonym Example: UNIT vs. ACTIVITY .

Homonyms are data element names that are the same or almost the same but which are used to represent things which are different with respect to usage or other characteristics. An example of a homonym is shown in Figure 2. Here the term AGENCY is used to identify the owner of a POSITION in the personnel system. Meanwhile, in the payroll system the term AGENCY (with a synonym -- SERVICING AGENCY-- in the payroll system) is related not to a POSITION but to an EMPLOYEE.

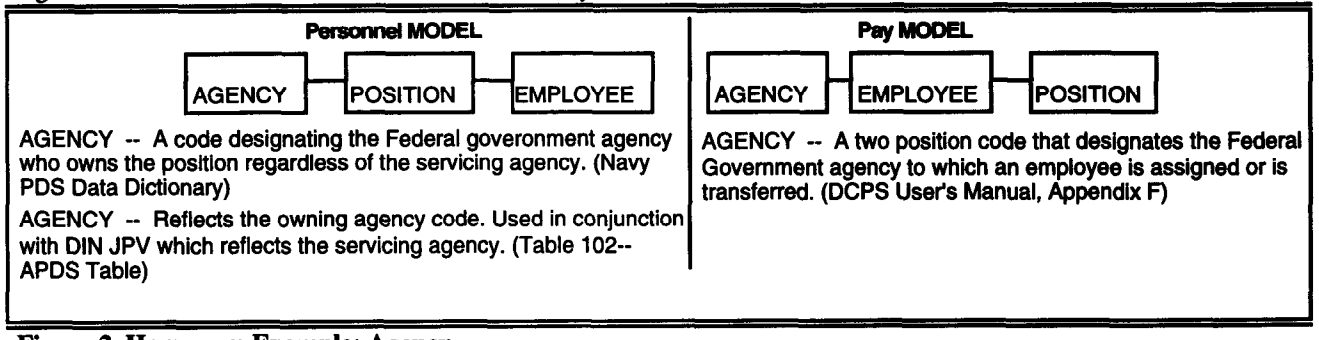

Figure 2. Homonym Example: Agency

Structural anomalies which are tracked as **Attribute/Entity** conflicts are an occurrence of a data element treated logically as an Entity in one system (i.e., many characteristics about it are tracked and stored) while in another system it is treated logically as an Attribute (one value which is used as a characteristic to describe same entity).

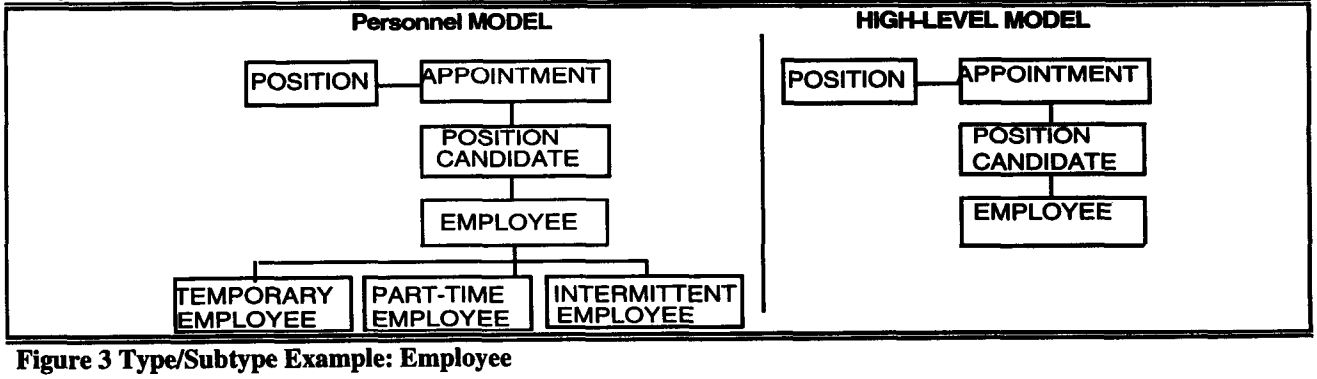

Figure 3 Type/Subtype Example: Employee

Type/Subtype conflicts show up where one system keeps only basic characteristics, or attributes, about an entity, and thus the varying types are simply multiple subtypes of that entity, while another system keeps a more complete set of characteristics about real-world things, thus encompassing all the variations within one type. As a result, these real-world things can be categorized into multiple types, each described by its own set of distinguishing characteristics. This conflict can usually be resolved by making the multiple type entities of the second system subtypes and placing common characteristics into the single entity from the first. An example of this type of problem is shown in Figure 3. Here the high-level model is more abstract and has only an EMPLOYEE entity, while in the personnel system, separate characteristics are tracked for TEMPORARY EMPLOYEES, INTERMITTENT EMPLOYEES, and PART TIME EMPLOYEES. Further investigation may even find additional employee types. This conflict actually helps to highlight a normalization need, such that common attributes across the multiple types will not be duplicated in the consolidated implementation.

The last conflict type is **Stored vs. Calculated Redundancy**. An example of this kind of conflict is the DOB (date of birth) sent from the personnel system to the payroll system, and the EMPLOYEE AGE CATEGORY stored within the payroll system after calculating the age from the date of birth. This conflict was at first only suspected, but was later validated when the actual usage of 'DOB' was noted by pay system experts--that is, that the date of birth is important to them only in determining the appropriate age category for life insurance premium calculation.

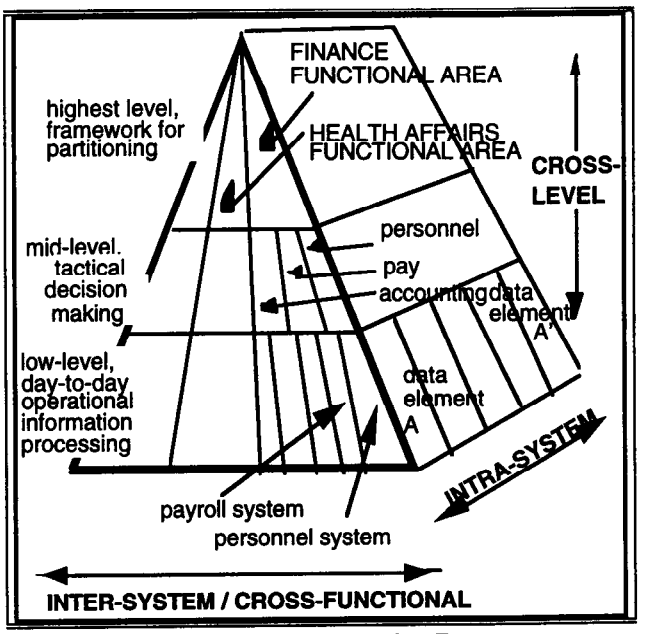

Figure 4. Four Different Integration Processes.

3. Integration Taxonomy

The process of integrating information can be categorized into a taxonomy which spans four different integration processes, as shown in figure 4.

Intra-system integration are undertaken to resolve data conflicts that exist during the re-engineering of each system. Within each system, several physical data elements may logically reference the same information. When a logical model is generated, like elements are combined as one logical object. For example, in the payroll system there are two similar objects: AGENCY and SERVICING AGENCY. They represent the same data, and the same data values. The difference is not in their content but in their usage. Within a logical model, both elements would be combined, and the latter would be indicated by associating the resulting object AGENCY with the real-world object served--in this case, the personnel office responsible for pay transactions of the employees.

Inter-system integration is undertaken to resolve data conflicts that exist among system being consolidated.

Cross-functional integration is required when information models from two or more functional areas are combined, and where the same information is maintained in both areas. *Inter-system* integration may also occur concurrent with *cross-functional* integration if the systems support more than one functional area. *Cross-functional* integration occurs within our integration task, for example, by logically integrating information that supports personnel functions with information that supports pay functions. The overlap occurs in the process of calculating pay, where information about an employee's salary, entitlements and deductions must be obtained from personnel to calculate civilian employees' pay, and information about time, attendance, deductions, and actual pay must be sent back to personnel to keep their records up-to-date.

Cross-level integration pertains to the level of conceptual detail being captured within the information model. At the lowest level, details that are used for day-to-day operations are included within the models. At the mid level, details are only included if relevant to the decision-making that will be made about how to accomplish broad functional goals. At the highest level, only abstract concepts are captured, to provide a framework for decomposition of functional areas into specific systems or organizations which provide those functions. Thus, *cross-level* integration occurs when folding one or more lower level models into one of the mid or higher level models, or simply mapping the more detailed concepts of the lower level model to a higher level concept through aggregation. Finally, our integration task also includes *cross-level* integration between a mid-level model which, for example, reflects current policy for calculation of pay and the lower level models.

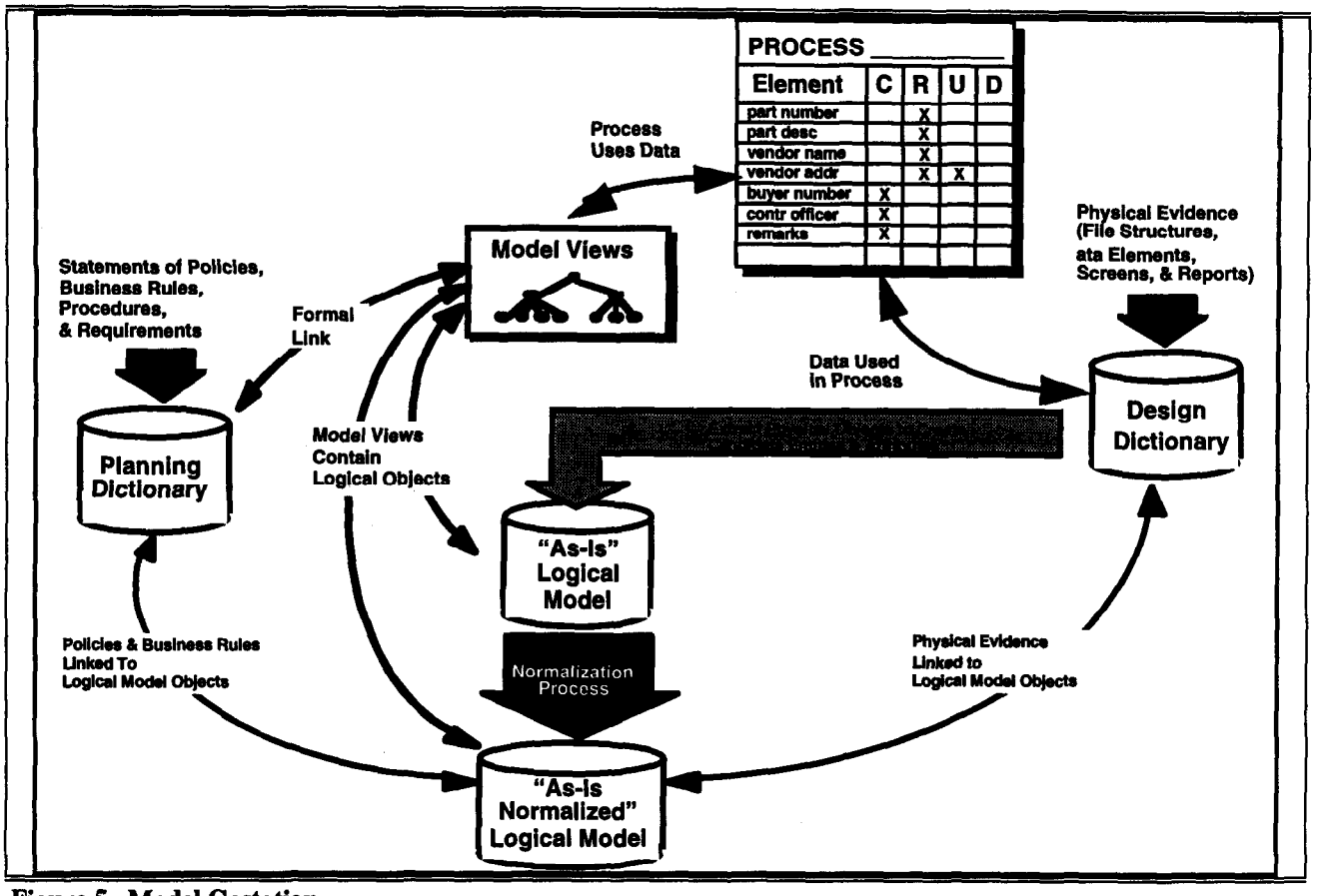

Figure 5. Model Gestation

4. Data Modeling and Re-Engineering Approach

With our re-engineering approach, we couple our extended entity-relationship modeling approach with the model integration processes.

As shown in Figure 5, our extended entity-relationship modeling approach partitions data models into the following components: (1) model view, (2) plan dictionary, (3) physical dictionary, (4) data dictionary, and (5) linkages (i.e., links between plan dictionary objects and model view objects, links between plan dictionary objects and data dictionary objects, links between data dictionary objects and design dictionary objects).

Business processes and data are inexorably linked. In our projects we have concentrated on the data requirements side, but we never forget about the underlying processes. Data does not appear suddenly out of the blue. Data is created by and is used by processes. Likewise, there are no business processes which don't use data. Failure to recognize these fundamental truths leads to confusion and artificial or theoretical results, not practical models and solutions.

As show in Figure 5, we first construct an "as-is logical data model" to represent the requirements, the logical structures and semantics, and the physical implementation of databases, screens, and reports. We then normalize the "as-is logical data model" to an "as-is normalized data model", up to the fifth normal form when possible, by using the up-to-date business strategies, business rules, and functional dependency information. The usage of each data object (i.e., entity, attribute, and association) are also documented by specifying the "how" (i.e., Create, Read, Update, and Delete), the "why" (i.e., links to the relevant policies, business rules, business procedures, practices, system implementation rules), the "when" (i.e., the "triggering business processes" and the "to be triggered business processes"), and the "where" (i.e., the "business process view" where the entity resides.) are also documented. During the normalization process, synonyms and homonyms in the physical data structures are identified and resolved. The resolution of each such conflict is documented with linkages to/from the logical attributes and the conflicting data items.

The As-Is Model

The *as-is data model*, once created, serves as a baseline representing the current configuration of the system. The process of creating the *as-is data model* is called *reverse engineering*. This model contains the existing definitions of processes, data structures, and data elements. Typically, our experience is that a large number of homonyms cannot be identified by analyzing application code, data dictionary, or schema definition. Multiple external definitions result from differing usage of the data

element between organizations within the enterprise using the application/database or "phased" usage of a data element during various stages of the process supported by the application where the meaning of the element changes. For example, illustrated in Figure 6, "Manufacturers cage part number indicator code" is used in four different ways by four organizations (i.e., DCSC, DESC, DGSC, and DISC.) During focus sessions, we discovered this homonym and obtained clear definitions, purposes, and usage of data elements from policy experts, end users, system implementers, and system maintainers.

Model views are created to partition the model into smaller pieces for analysis and to create meaningful groupings of business rules and data. Generally, these are process- or function- based although we can also create "data" views based on clusters of related data.

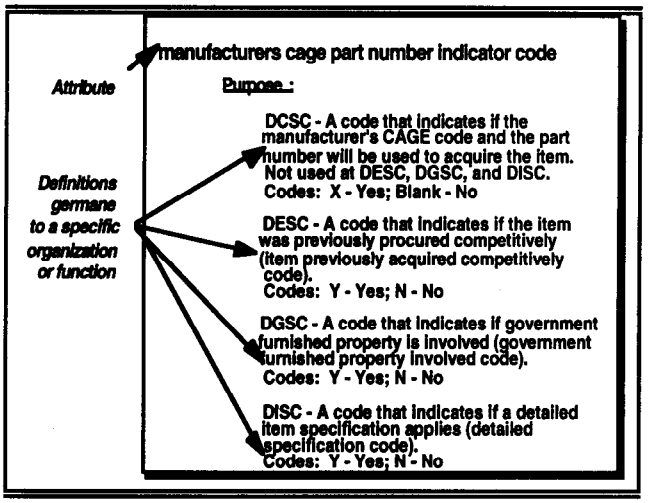

Figure 6. Homonym Detected from End User Usage.

As shown in Figure 7, functional and business rules, developed from policies and regulations, drive system rules while system rules describe processing logic and data relationships to be implemented by the application code. To facilitate the requirements-driven approach, business rules and policies are generated from external regulations and laws. Common practices and procedures within the business enterprise are documented in the *plan dictionary* of the data model as shown in Figure 6. Additional rules are also captured in the *plan dictionary* through analysis of the application code (these are the system rules - that implement the business rules). Rules are then linked to the *model views* to which they apply as illustrated in Figure 5.

Physical data is correlated with business processes (model views) via the create, read, update, delete (CRUD) matrix in spreadsheets. This matrix is also used to capture the operations performed on the data for a specific process, whether the data is optional or mandatory, and is system-supplied or user-entered.

After the physical components of the model are defined (in the *design dictionary*) and validated, are "extracted" from the design dictionary, they are placed in the logical *data dictionary* as candidate objects and used to populate the *model views*. Preliminary linkages from physical to logical are established as part of this extraction process to establish the pedigree of the candidate logical objects. The as-is model is then frozen for future reference and is ready for use as a baseline for the development of the as-is normalized model.

The A-Is Normalized Model

After as-is data model is derived, we begin the forward engineering process which includes normalization (through fifth normal form) of the candidate objects. Normalization reduces complex data structures to their basic, most natural form by removing redundancies, eliminating data conflicts, and assigning attributes to entities based on the essential meaning of data throughout the organization. The resulting data architecture is based on how data is related to other data, not on how the data is used by the applications or on who uses the data. This provides a more stable and flexible structure. This logical data architecture is used to drive the creation of physical data structures free of creation, update, and deletion anomalies that result in inconsistent data values and other data integrity problems.

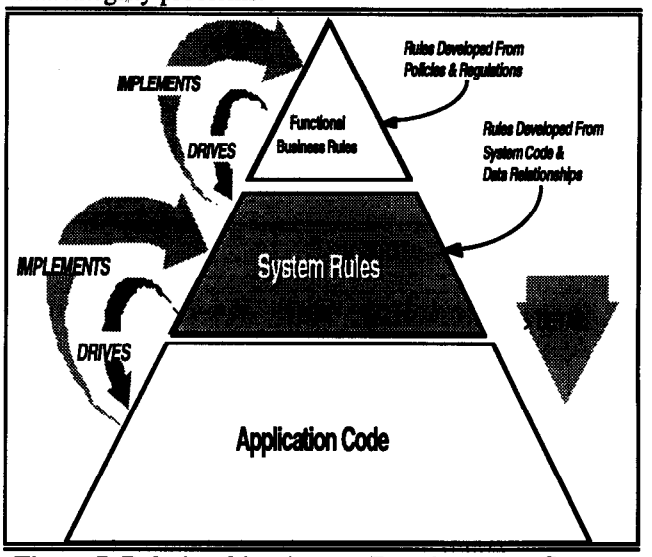

Figure 7. Relationships Among Requirements, System Rules, and Application Code

As a result of normalization, new objects are created and refined relationships established that differ from the original as-is condition. Therefore additional rules (requirements) must be written to support the normalization assumptions/rationale. New business rules are recorded as modeling issues are resolved.

Standard naming conventions (per enterprise standard) are applied to logical objects. In many instances new standard logical names based on the essential meaning of the data will prove more transportable across the enterprise than the legacy physical data names.

A full set of trace links between logical objects and rules (planning statements) can now be completed. Trace links between logical and physical objects must be refined to account for logical objects eliminated to condense

synonyms and additional logical objects created to resolve homonyms. In some cases, new objects (entities/attributes) are created which are not currently part of the application/database but are logically part of the business enterprise supported by the application being reverse-engineered.

The stage has now been set to achieve a model-driven development environment as shown in Figure 8.

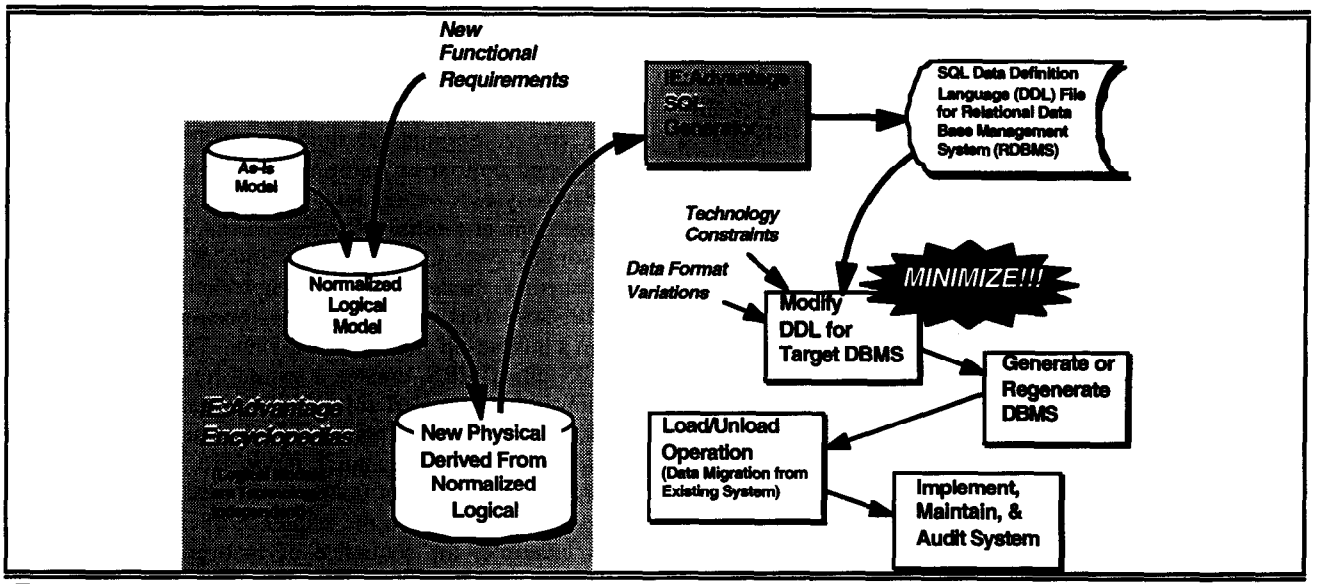

Figure 8. New Physical Schema Derived from the Re-Engineered Data Model

Forward Engineering and Model-Driven Development

Upon completion, the normalized model can be archived as a baseline for configuration management purposes, and a new model is generated from the normalized model to meet two possible scenarios. Scenario 1 is to upgrade the system implementation for current needs by migrating the existing data base to a relational data base by creating new physical components from the *as-is normalized data model.*

Scenario 2 is to expand system functionality. We would first modify the *as-is normalized data model* into the *to-be normalized data model* by capturing new business requirements. In addition, we integrate strategic, tactical, and operational level data models with the re-engineered "as-is normalized data model". We first use the strategic level enterprise data and process models to isolate the relevant concepts and identify relationships between tactical data and process models. Next, we use the relevant tactical data and process models to identify grouping of business processes and possible entities and associations. Then, we identify identical data objects, synonyms, and homonyms in the related operational data models. Finally, we resolve conflicts and produce an integrated logical data model with a traceability matrix. Then, migrating the existing data base to a relational data base by creating new physical components from the *to-be normalized data model*.

5. Our Model Integration Approach

In this section, we will use a payroll system, a personnel system, the pay calculation application and related data elements from the two systems as a small example to illustrate integration complexity and our approach to integration. In this example, we have *cross-level* integration between a mid-level model which reflects current policy for calculation of pay and the lower level models; we have *cross-functional* integration between the personnel system representing personnel functions, and the payroll system representing pay functions; we have *intra-system* integration within each migration system model as it relates to the calculation of pay that is part of the re-engineering effort that prepares it for integration; and finally we have *inter-system* integration since we need to resolve data conflict between the two re-engineered data models representing the data requirements of the two systems.

Integration Scope

The scope of this task is the information exchanged between the two systems for the pay calculation. As shown in Figure 9, there are three subsets of data that should be distinguished.

The most critical items are those shown in the intersection, i.e., elements that are shared by the two systems and are directly related to pay. Examples of this type of data are an Employee-Id and the employee's associated Pay Grade. The second subset includes elements that are not shared by the systems but which are directly related to pay. Examples of this kind of data are such things as Time and Attendance data or Allowance Categories which are kept only within the payroll system. The third subset includes elements that are not shared, but

are common to both systems, and are either directly or indirectly related to pay calculation. The most obvious examples of this kind of data are reference tables, such as locality adjustment percentages which affect pay, or various benefit plan codes. The less obvious are elements such as salary limitation accumulators which are officially kept only within personnel, but are actually maintained within both pay and personnel. Stewardship (i.e., organizational authority and responsibility for a specific set of data elements) may be the most difficult to determine for this type of data.

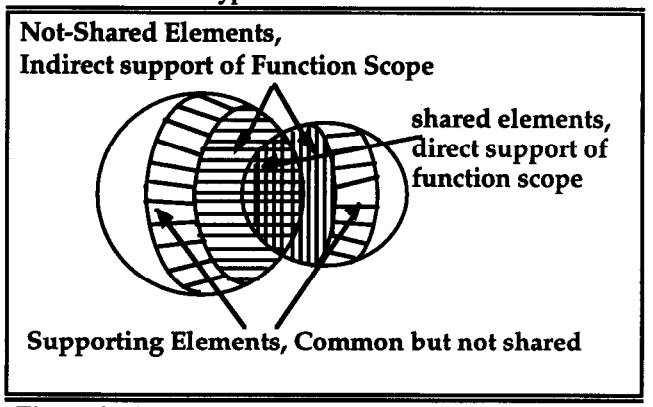

Figure 9 Scope of Data Elements to be integrated.

Bias Priority Approach

We use a bias priority approach discussed in [BLN86]. Integration tasks encountered in our projects involve all four integration processes (i.e., intra-system, inter-system, cross-functional, cross-level integration.) Therefore, the approach being taken on this task is to merge each of the re-engineered data models into one medium-level logical model.

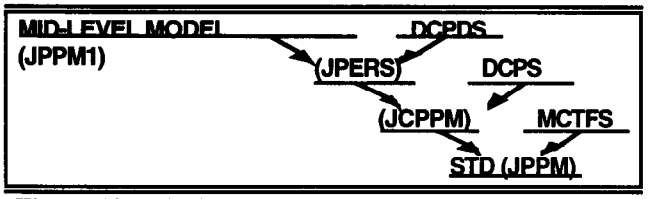

Figure 10. Priority Sequence Biases Resulting Standard Elements.

In Figure 10, DCPDS is a re-engineered data model for a personnel system, DCPS is a re-engineered data model for a payroll system, and MCTFS is a re-engineered data model for a combined pay and personnel system. In the process, conflicts of naming, structural representation, and semantics are uncovered. In addition, stewardship may be unclear. The goal of the task is to isolate each of these conflicts and document the alternatives with the data-related impact of each. The stewardship of the conflicting elements may also be discovered; if so, this is noted to help in establishing resolution authority.

The initial version of the mid-level data model is developed top-down, using a variety of policy-level input sources. In this specific example, policy documents included DoD Directives, the Federal Personnel Manual and supplements, and several high-level models that have been developed, including the DoD Enterprise Model. A similar mid-level data model in a non-government organization can also be developed using similar documents such as Federal Law, State Law, corporate directives, a corporate personnel manual. The mid-level model resulting from this policy review has been called the Joint Personnel and Pay Model, JPPM.

The two re-engineered models, DCPDS and DCPS, developed bottom-up using existing data structures and in-depth system expert interviews as input sources, are merged one-at-a-time. The first merge, DCPDS (i.e., the personnel system data model) into JPPM, creates an intermediate model which is being called Joint Personnel, or JPERS. This model integrates personnel information from a bottom-up and top-down view. The second merge, DCPS into JPERS, creates a second intermediate model which is being called Joint Civilian Personnel and Pay Model, or JCPPM. This model uses the integrated top-down/bottom-up personnel model to help guide the integration of bottom-up pay information. The *bias priority* is inherent in the sequence of integration. In this example, the sequence of incorporating DCPDS, DCPS, and MCTFS is driven by the time that a re-engineered data model is available. We did not intentionally impose preference on any of the systems, but the timing of model availability turns out determine the bias priority.

As shown in Figure 11, the mid-level model (JPPM1) has the highest priority; if there is a conflict between it and DCPDS, the mid-level model representation will take precedence. JPERS will therefore be biased towards this mid-level view. Then , JCPPM will be biased towards JPERS (i.e., the integrated model from the mid-level model and DCPDS) over DCPS. Finally, the Standard (the JPPM model) will have the bias toward JCPPM (i.e., the integrated model from the mid-level model, DCPDS, and DCPS) over the MCTFS . As the integration process moves to the final stage, MCTFS will have least impact on standardized terms as compared to DCPDS and DCPS when conflicts arise.

The *bias priority* supported by this approach is especially important when integrating a large number of existing systems over time. The greatest advantage is in managing the complexity that arises when consulting the number of system and functional experts who may not previously be aware that their data is also kept in other systems, perhaps used as well as named quite differently. Special data uses may also have to be compromised to accommodate the standards, and mediation without a pre-determined priority could be very costly in time, money, and delays due to stalemate. The most effective sequence is established by weighing the following factors: (1) the breadth of use of the data elements in each system to ensure the most complete understanding possible early-on, (2) the dollar costs of changes to each system or the

time and cost of mapping the system's data to the standards resulting from earlier integration, to limit the ultimate costs, and (3) the number of employees or organizations supported by each and the relative importance of the functions supported by the data being integrated, to consider the risk of change. More detailed advantages of this approach over others are discussed in [BLN86].

Since one of the strategic objectives is to produce sharable data among enterprise organizations performing various business functions (such as personnel, payroll, procurement, health care benefit, inventory management), after the integration process we use a naming standardization procedure [DoD94] which ensures reusability of data objects. For example, we re-use the standard name if the data object is equivalent to a standard data element in the enterprise data repository; this is done by transforming the name of an attribute into the format "Modifier Name.Generic Data Element" such that the transformed attribute and its associated data model can be submitted as an enterprise standard.

Lessons Learned

Establishment of Re-Engineering Strategic Objectives

Getting formal commitment and authorization from the system stakeholders (especially administrative and technical management) is a crucial factor to the success of reverse engineering projects. One problem we experienced was the lack of timely access to key system and functional personnel because they were simultaneously required in other activities. In addition, the costs of properly re-engineering systems and the value of the re-engineering products produced are consistently under-estimated by management (this is true is the commercial environment as well). It has been challenging to convince management that re-engineering is a substantially broader and more complex task than just "restructuring the code" by pumping the code through a CASE tool. Restructuring the code is useful to improve the system maintainability, but it does not support the following critical tasks: (1) facilitation of data migration, (2) evolution of a departmental system to satisfy enterprise-wide system requirements, (3) data sharing among functional areas (e.g., Health Care, Personnel, Pay), and (4) incorporation of new or changed data requirements resulting from business process improvement activities. Nor does code restructuring consider deletion of obsolete functions or requirements that don't need to be included in the future systems or provide for identification of incomplete coverage of business requirements by existing programs.

Establishment of Use of Re-Engineering Products

Throughout the initial phase of our projects, we have been continuously asked:

- What reverse engineering products are produced?
- How can these products be used?
- How is reverse engineering related to other system development activities?
- When will the re-engineering products be ready?
- Why do they need re-engineering in order to obtain standard data elements?

The matrix in Table 1 helps to answer these questions.

Data Standardization and Integration Misconceptions

A dangerous misconception is that data standardization is simply assigning related data elements with the same name in each system. If implemented blindly, this will cause future unforeseen and disastrous results, the causes of which are difficult to isolate and correct. To correctly use information from data elements, the business rules, policies, and the functional dependencies among elements must be identified and represented in a data model for each system. Before standardization can be achieved, model integration is an essential step to identify and resolve synonyms and homonyms based on the rules, policies and functional dependencies represented in the models. Without the integration step, incorrect information continues to propagate throughout the enterprise.

The Need for a Re-Engineering Cost Model

Costs of re-engineering efforts are difficult to estimate. Costs models are important in order to clearly enumerate the contributing factors/tasks/resources required for a project (based on the environment (technology, administrative and operational) and the size of the system in question. One vendor we know estimates a flat $1.00/line of code in the system. We were unable to even estimate the lines of code in the medical portion of the project because of the unstructured nature of the MUMPS programming language code. One measure of prowess among MUMPS programmers is how complex a program can be written with a single line of code. Perhaps this accounts for the various estimates in the number of lines of MUMPS code in one of a hospital information system ranging between 1.3 million to 2.5 million depending on whom you ask.

Re-Engineering Tools and Tool Usage

The tools available are geared to specific target portions of the IS life cycle. Many tools on the market today exhibit varying degrees of tunnel-vision (covering only the portion the vendor is interested in and not providing adequate links to other tools or common formats. Business rules and other information captured in logical or conceptual models (or phases) may not be passed easily

to forward engineering tools of choice, and as a result it is not easy to implement the structures, rules, and business requirements with full traceability.

Selection of the proper tool set across the entire life cycle requires a comprehensive strategic vision and a re-engineering process model (across the entire IS life cycle) aligned with the objectives of the customer organization. Tool selections should align closely with the customer organization's goals, priorities and resource constraints.

The current crop of CASE tools focuses on associating physical data structures and variables to segments of code. This is useful in identifying how physical schemas are created, updated, read, and deleted, but it is not sufficient to construct the logical and conceptual external views of data requirements. Even if the functional and technical experts help the information engineers understand the existing physical systems, the CASE tools do not provide, for instance, the facility to link to the physical evidence (e.g., functional specification, software, data dictionary, focus session results) recorded in support of the functional requirements. More importantly, using such tools in a purely mechanical fashion (i.e., without help of the technical and functional experts) will be more damaging, producing inadequate, inaccurate, or incomplete results. Recovering a normalized logical data model together with the associated business rules, policies, and physical data structures is hard, and requires significant amount of human effort, but this extra effort is required in the long run. CASE tools may augment the analysis to provide initial understanding of physical implementation of the system, but CASE tools cannot replace human effort for recovering the conceptual and logical data requirements.

Task Timing: Availability of Re-Engineered Models

If the legacy systems being integrated do not have existing and up-to-date models of the processes and information structures, the integration task should be delayed until they are close to readiness. While it is important for the integrator to attend and perhaps participate in some of the re-engineering modeling sessions held in order to become familiarized with the terms used in discussion, this participation can occur towards the later stages of model review, as preparation for integration. The models should be at a relatively stable state, and complete with traceability before integration begins.

Task Reviews

Deliverable Products Reviewer

It is important to find a reviewer of the deliverables, especially of the conflict cross-references, the conflict resolution document, and the traceability of the model, who will concentrate on coverage and format, not necessarily on term resolutions. The question that must be answered by that reviewer is "Will those who are required to resolve have enough information to make a decision?". If not, the task output can not be relied on. However, there is a need to steer clear of *too much* information, as well, in order to keep the documents readable. Some representations are good for working documents, some are good for technical input, some are good for formal review, and some can be made more simple for validation. It is important to adapt to the readers' needs, and to have a customer reviewer available to help guide the content.

Track Progress Using Tangible Metrics

It is possible to establish tangible metrics that can be used to gauge the progress of the integration task for status reporting and scheduling. For our integration project, the metrics were based on the intersection diagram shown in Figure 1. Each legacy system being integrated has a total number of objects (entities and attributes) within it. This number applies to each full circle in the diagram. The isolation of overlap areas in the systems as they are prepared for integration will uncover the number (and percent) of objects that fall within each band on the diagram (i.e. a--shared and direct functional relevance, b--shared and indirect functional relevance, and c--common). Once these numbers have been identified, progress can be monitored by reporting the number of objects in categories a, b and c that have been merged; the number of objects in each category that show up in a conflict; the number of objects in each category that have resolutions; the number of objects that have had traceability noted upwards, the number noted downwards; and so on.

Integration Task Process/Methodology

Draft Top-Down Model is Key to Familiarizing with Terms

In some of the first systems we worked on, the mid-level model which reflects policy (to compare to implementations) was not planned. Its development on the integration task was done as an exercise of the tools and procedures, while awaiting the re-engineered models of the personnel and payroll systems. It was found that this exercise greatly contributed to familiarity with terminology as used by each of the systems' experts. This familiarity helped guide the initial focus sessions, to help in quickly isolating terms that were being used differently in each system's context.

Establish Pair-Wise Priority Prior to Scheduling Focus Sessions

The sequence of integration will affect the bias of terminology usage that ends up in the ultimate consolidated standard. If the focus sessions do not

proceed in this order, however, or if attendance by the integrator does not occur in this sequence, or if the sessions do not cover interchange elements right from the beginning, terminology bias will occur incorrectly. In the case of our project, payroll system information was available significantly earlier than personnel system information. While the bias has for the most part been overcome through rigor in integration, it is possible that information provided by the payroll system contacts might have been questioned more thoroughly from a personnel usage standpoint than it was, simply by being introduced after personnel functions and data were more familiar. One suggested resolution is to be sure to schedule interview sessions according to the priority sequence. An additional suggestion is to be sure to obtain on-going validation of the terms from each system being integrated even after its integration has been completed. For this prototype effort, it was important to take the payroll system input back to the personnel system experts to validate the feedback obtained, and to reach a common understanding.

Use of Legacy System Documentation

Legacy system logical data models are required, but, in our case, models were not available at first, so legacy system documentation was researched as sample input for conflict isolation. Since this documentation often has more in-depth definitions than are found in model data dictionaries, conflicts were sometimes found that would not have been discovered simply by reviewing the input models. On the other hand, the models contain more semantics in the form of data dependencies and association text, as well as in the structural representation of the element (i.e. entity vs. attribute, primary vs. subtype entity). The models also represent current actual practice, something not easily gleaned from documentation. The recommendation is to try to use both as information sources to help in investigating, analyzing, and uncovering candidate conflicts.

Along the same lines, the legacy system model representation should take precedence whenever contradictions occur between them and the system documentation, since the models attempt to capture actual as-is implementation as it has evolved over time, while documentation may not be up to date with actual practice.

Integration Task Output

Conflict Groups

The conflicts found between legacy systems were originally going to be documented only in cross-reference or matrix form. These show one element name from one system with one element name of the other system, noting the potential conflict. To serve the purposes of conflict resolution, however, the elements need to be logically grouped to include all of the several conflicts that involve those terms. The reason is that conflict resolutions impact each other when like terms are involved.

For example, if there was a synonym conflict between ACTIVITY in one system and UNIT in another, it may be agreed that UNIT should be selected; if there is also a synonym conflict between UNIT and ORGANIZATION, and on its own the term ORGANIZATION is selected, does that now mean that what used to be called ACTIVITY should now also be changed to ORGANIZATION? Or would the existence of both conflicts as a group lead the integrator to suggest that all of ACTIVITY, ORGANIZATION, and UNIT be called UNIT? Or, would the integrator find that there was a homonym use of UNIT going on? The grouping of both conflicts together (or multiple conflicts if necessary) provides the highest level of semantics from which the integrator can make recommended resolutions.

Stewardship Conflict Resolution

When identifying stewardship of the exchanged data elements, there may be a subset which is claimed by both (or multiple) system experts. This issue cannot be resolved by the integrator during integration. It should be noted within the Conflict Resolution Document, and the preliminary recommended resolutions can be used as input in the integrator's decision. The recommendations will require further review before actual implementation; this multiple claiming should be addressed at that time. The integrator can raise a flag within the document to make reviewers aware of these problems.

Use of Cross-Reference List for Customer Input

In our integration task, cross-reference lists were used heavily to retrieve input from systems' functional area experts. Be very careful to include only information that is relevant to the given system when requesting feedback, if possible, to keep the expert focused on input you need rather than differences found in the other system(s). Along the same lines, be sure to provide separate lists for each system, with their terms as the first, identifying column. This is not only an issue of pride-of-ownership but also a matter of providing information that will be most recognizable to them; the more recognizable, the better and faster their review can be.

Conclusion

Requirements-based re-engineering is not for solving minor information technology problems or providing minor improvements to legacy information systems. Re-engineering is a non-trivial task best directed toward formulating long-term solutions for the business enterprise.

A requirements-based approach to data modeling and re-engineering requires cooperation and resources from the

policy makers and interpreters, end users and functional experts, as well as the spectrum of IS professionals Therefore, it is of paramount importance to align project objectives, plans, technologies, and tools with the organization's strategic plan in order to obtain buy-in and commitment from system stakeholders ([MH94], [AHR94], [AHR93], [HI93a], [HI93b]).

It is essential to understand that automated tools, while facilitating the process of re-engineering, cannot do the entire job. A suite of tools supporting the life cycle of the re-engineering effort that can exchange data would be ideal. No CASE tool in itself can do the entire job and may even prove to be bottleneck. To ensure that a thorough job of re-engineering is performed human intervention is required for analysis and problem solving. Indeed, a committed team is required to establish the continuity of expertise and purpose needed to successfully implement a model-driven system development and maintenance environment.

Acknowledgments

The work described here is based on the joint efforts of the Data Processing Systems Modernization Program. The authors are indebted to Carlo Zaniolo of UCLA for comments and discussions.

References

[DoD94] DoD Directive 8320.1-M-1: Data Element Standardization Procedures.

[MH94] Muntz and Hobson. Lessons Learned: Re-engineering DoD Legacy Information Systems. *Software Engineering Techniques Workshop on Software Reengineering, Pittsburgh, PA, May 3-5, 1994*

[AHR94] Aiken, Muntz, Richards. DoD Legacy Systems: Reverse Engineering Data Requirements. *Communications of the ACM, May 1994, Volume 37, Number 5*, pages 26-41.

[AHR93] Aiken, Muntz, Richards. A Frame work for Reverse Engineering DoD Legacy Information Systems. *Proceedings of Working Conference on Reverse Engineering, May 21-23, 1993, Baltimore, Maryland*, Page 180-191.

[HI93a] Hughes Information Technology Corporation. Approaches to Cross System Functional Integration: Phase I , Final Report for Data Processing Systems Modernization Program. Prepared for Defense Information Systems Agency, April 5, 1993.

[HI93b] Hughes Information Technology Corporation. Reverse Engineering DoD Legacy Information Systems, Phase I, Final Report for Data Processing Systems Modernization Program. Prepared for Defense Information Systems Agency, April 13, 1993.

[BLN86] Batini, Lenzerini, Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys, Vol. 18, No.4, December 1986*, pages 323-364.

[P&B94] Premerlani, Blaha. An Approach for Reverse Engineering of Relational Databases. *Communications of the ACM, May 1994, Volume 37, Number 5*, pages 42-49.

[HDA87] Hogshead-Davis, Arora. Converting a Relational Database Model into an Entity-Relationship Model. *Proceedings of the Sixth International Conference on Entity-Relationship Approach,* 1987.

[M&M90] Markowitz, Makowsky. Identifying Extended Entity-Relationship Object Structures in Relational Schemas. IEEE Trans. Softw. Eng. 16, 8 (Aug. 1990), 777-790

[M&S89] Markowitz, Shoshani. On the Correctness of Representing Extended Entity-Relationship Structures in the Relational Model. *SIGMOD '89. ACM*, New York, 1989.

[NEK94] Ning, Engberts, Kozaczynski. Automated Support for Legacy Code Understanding. *Communications of the ACM, May 1994, Volume 37, Number 5*, pages 50-57.

[MNBBK94] Markosian, Newcomb, Brand, Burson, Kitzmiller. Using an Enabling Technology to Reengineer Legacy Systems. *Communications of the ACM, May 1994, Volume 37, Number 5*, pages 58-71.

[HCTJ93] Hainaut, Chandelon, Tonneau, Joris. Contribution to a Theory of Database Reverse Engineering. *Proceedings of Working Conference on Reverse Engineering, May 21-23, 1993, Baltimore, Maryland*, page 161-170.