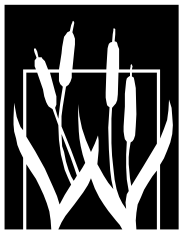

Whitemarsh
Information Systems Corporation

Section of
Achieving Data Standardization

Whitemarsh Information Systems Corporation
2008 Althea Lane
Bowie, Maryland 20716
Tele: 301-249-1142
Email: mmgorman@wiscorp.com
Web: www.wiscorp.com

1

Why Data Standardization

To achieve world-wide enterprise database, data standardization is essential. Data standardization embraces:

- *a standard data architecture,*
- *an approach to accommodate diverse data semantics,*
- *enforcement of standard cultured and language independent value sets,*
- *a semantics and value set enforcement environment that is unavoidable and work enhancing*

Without effectively defined and systemically implemented data standards database objects[1], the foundation stones of world-wide enterprise database cannot succeed.

Data is executed policy. Data semantics are the technical representations of policy specifications. Unless data is represented through homogeneous semantics policy adherence cannot be measured or evaluated.

Policy executions, that is data, are the medium of business communication. Figure 1 illustrates this point. This figure shows subject areas as the basis for one or more business policies. If the subject area were human resources there would be policies regarding the application for employment, the information that must be provided on the first day of hiring, and the like. In turn, these policies result in the technical specifications of data that must be provided. These specifications ultimately map onto actual data that exists. The subject area also maps onto database schemas, for example, the database schema for employees. In this schema, there would be a number of tables, including for example, those that would contain critical information from the application for employment, the first day of hire benefit selection documents, tax withholding documents, dependent information, department assigned, and the like. Again, these tables map onto the actual rows of data that must reside in the database. In short, data is the proof that the policy was executed.

---

[1] Database objects are a type of object that are unique to ANSI/SQL. A database object consists of four parts: data structure, database object processes (object transformation methods), database object information systems, and database object states. Because database objects are uniquely SQL based, their database object classes can be ported between SQL compliant DBMSs. Database object classes are stored in schema information tables and the database objects are stored in SQL databases.

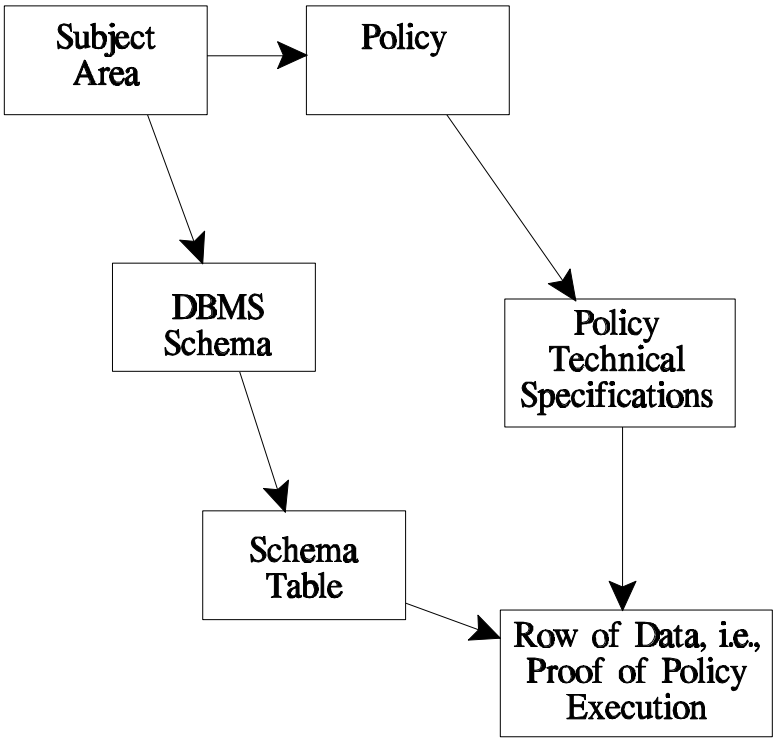

Figure 1. Levels of Abstraction Required for Data Standardization

Figure 1 indicates not only hierarchies of instances, but also contains multiple levels of abstraction. For example:

- Policy is represented through policy technical specifications
- Policy technical specifications are represented through rows of data
- Schema tables are represented through rows of data

Traditionally, databases contains two levels of abstraction: the schema, and the rows of data. The schema represents *types* and the rows of data represent *instances*. When ever there are multiple levels of abstraction there is, by definition, types and instances. Metadata repositories, built on a DBMS foundation not only contain both a schema and rows of data, they may contain data across multiple levels of abstraction. Thus, repositories may contain multiple sets of interleaved metadata (type) and data (instances).

To effectively represent the data required to store and control data architectures across the enterprise, the repository must capture data from across multiple levels of abstraction, as shown in the database represented by the DBMS tables in Figure 1. In database, there is no restriction on the quantity of levels of types and instances. If one pair is allowed, then all pairs, including interleaving pairs are allowed (type then instance, which in turn is type then instance, which in turn ....).

Because data is so critical, it is essential that it be commonly represented and understood. Database is a mechanism of common data definition, common data loading and update, and common reporting. But what of databases that cross different cultures, languages, and finance systems? If all data is defined to specifically reflect cultures, languages, and finance systems it becomes virtually impossible to accomplish effective business communications.

Data, the medium of effective business communication, must both reflect localized culture, language, and finance systems yet able to be amalgamated into broader contexts for country, region, and world wide decision making. The alternatives are simple: one single data standard for all reasons, or a set of mapping rules and mechanisms that allows data to be recast from local culture, language, and finance systems to the broader country, region, and world contexts. While the former is certainly simpler and less costly, it has never been successful. Thus, we are only left with the later.

Database does not solve this problem, rather, it exposes it. Prior to database objects, data definition and data value incompatibilities were hidden by technology mismatches. Because of the database language SQL, these technology inconsistencies have been removed. Exposed for all to see are the data semantics mismatches. These must be solved prior to meaningful data sharing.

1.1 Data Semantics

Data semantic problems exist when the same concept is known under two different names, is described differently, or is different in some way that causes the semantic sameness to be hidden. For example, the EMPLOYEE_ID of 525-90-2896 as an ASCII string of 11 characters is really a Social Security Number of 9 integers. While the concept is the same, their names, data types and its lengths are different. The common concept is hidden by the differences.

Another example of semantic irregularities is found whenever concepts hide intrinsic rules and/or common sense. For example, the default number size for the Oracle DBMS is Decimal 12.2. For the column ANNUAL_OVERTIME_HOURS, this value would allow for 999,999,999.99 hours. Considering there are traditionally only 2080 work hours per year, the amount of accruable annual overtime seems excessive.

A third example of problems associated with data semantics are the different data structure representations for the same concept. For example, in one inventory data file there were many sets of fields all dealing with inventory valuation. Each set of fields consists of 12 related fields, one for each calendar month. The field names were only five characters long and had to detail the type of data, the calendar month, and whether the data was for the current, prior, or next year or period.

In another inventory file there were only four fields: year, calendar month, item number, and inventory quantity. The first inventory data structure contained 39 different fields to represent a very specialized version of inventory valuation data while the alternative data structure only had four fields representing a very generalized version.

Data semantics irregularities are most commonly evidenced through differences in:

- data names

- data types
- data lengths
- data structures

1.2 Process Semantics

Process semantics are the rules that govern data transformation. That is, the allowed or disallowed sets of values, interrelationships of values from different fields, or the transformations of allowed values from one valid state to the next. For example, death-date must be equal to or greater than the birth-date, or that an employee cannot have a discharge record without first having an employment record. Process semantics are enforced either from a database or by computer programs that access files or databases. If the data operations are to and from standard access files then the only possible place all these process semantics is computer programs. To prevent multiple programs from executing conflicting process semantics, the process semantics should be placed within the highest level of common usage. For database applications the highest level of common access because of multiple programming languages and/or multiple users from different sites is within the database schema.

Over the years, starting in the early 1970s, through the present day, the quantity of process semantics able to be stored within and welded to the schema has been dramatically increasing. This trend has heightened the ability to have high integrity and quality database information systems. The derived value is so great that to define, store, and manipulate data through any other mechanism than a semantically rich DBMS should be considered a significant error. In addition to increased integrity and value, the cost savings accrued to database and information system design, implementation, and life cycle maintenance is dramatic. The savings are at least 25% over the standard access traditional approach, and in some systems environments 75% savings have been achieved.

1.3 Specified, Implemented and Operational Versions

Databases and their supporting information systems exist in three forms: specification, implementation, and operation. The specification of a database and an information system is typically independent from its implementation form. This specification form is also called "logical." The term "logical" however, is often used by different persons for different reasons.

To the database designer, entities and attributes are the "logical" product of their efforts. When the DBMS's DBA staff produce the DBMS schema, their product is commonly called "physical." But to the database management system specialist, the DBMS schema is "logical" and the actual completely tuned database that is running on the computer is called "physical."

The terms "logical" and "physical" are therefore the names applied to two sides of a transformation. The "in" side is "logical" and the "out" side is "physical." Because of this common confusion, the term "specified" applies to all forms of the database and information

system specification in a form that is independent of computer hardware, DBMS, or operating system.

The term “implemented” applies to all the forms of the database and information system specification that is dependent or bound to the computer hardware, DBMS and operating system. The implemented forms should occur only after the specified forms.

Finally, the term “operational” refers to these same components that are actually operating on the computers through the DBMS and the operating systems.

A database and information system’s specification may be transformed many different times for different combinations of computer hardware, DBMS and operating systems. Thus, there is a one-to-many relationship between the specified and implemented forms. In addition, there may be implemented databases that are compositions of different parts of specified databases. This makes the relationship many to many. This commonly occurs in data warehouse environments.

It is critical to preserve both the specified and implemented forms of data and process semantics, and to also preserve the mappings between these two forms. When this is done, database and information system specifications merely have to be re-transformed whenever there is a need to significantly change the computer hardware, DBMS or operating system. The analogy is essentially the same between the implemented and operational forms of a database and information system.

In a recent resystemization effort a senior executive complained that since the fundamental business model of the corporation had not changed in 40 years that it was a complete waste of time and resources for the data processing department to rediscover and respecify the fundamental business model every 5-10 years just because the technology of business systems’ implementation was changed. The executive made his point even more forcefully by not only *rejecting* his cost allocation from the effort but also he charged data processing for *his* time.

1.4 Life Cycle Costs

There is fundamental formula that governs all data processing system development and evolution cost. Figure 2 illustrates that formula. It is divided to two cost lines, 1st implementation version, and follow-on evolution and maintenance versions.

The ratio between the cost of analysis and design products and the product costs for the remaining first implementation version effort (binding, implementation, testing, training, documentation, and production commencement) is about 1 to 4 for a total cost of 5 units. If the analysis and design costs are 6 staff months ($90,000 at $15,000 per staff month), the implementation costs are another 24 staff months ($360,000) for a total cost of $450,000.

The life cycle costs for the follow-on evolution and maintenance versions is about 5 times the cost of the first implementation version. That means that the total cost is about 30. In this

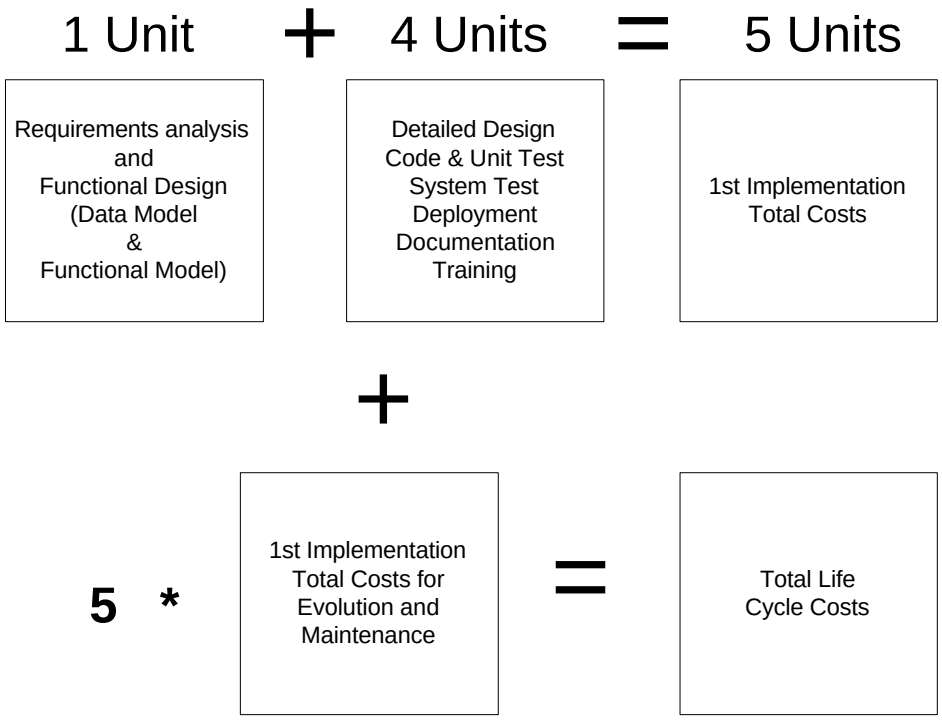

Figure 2. Fundamental Formula for Estimatig Data Processing System Life Cycle Costs

example, the evolution and maintenance costs would be $2.25 million, and the total cost would then be $2.7 million.

Any methods and techniques that reduce the first implementation ratio of 1 to 4 to for example, 1 to 2 produces dramatic overall cost savings. If reduced to 1 to 2, and if the life cycle maintenance is still 1 to 5, then the overall cost is only 15. That's a 50% savings in life cycle costs.

A significant contributor to the reduction in the first implementation ratio of 1 to 4 ratio is CASE, repositories and code generators. If the first implementation is created through computer-aided systems engineering (CASE), and if the specifications are used and reused via metadata repositories from one implementation cycle to the next, and if the resultant database information system is largely generated because a significant quantity of the process logic is welded to the DBMS schema and/or accomplished through sophisticated report writers, then reducing the ratio of 1 to 4 to a ratio of 1 to 1 or even less is very possible.

The negative consequence from not having a quality systems environment (CASE, repository and code generators) is non-linear rising staff costs. The costs rise non-linearly because of rising sophistication, requirements, and complexity. If however, a quality systems environment is in place, then the costs per systems can either remain the same or be reduced. That is because the work-enhancing tools from CASE, repository and code generators compensate or more than compensate for rising sophistication, requirements, and complexity.

Without a quality systems environment, the only way to maintain or reduce costs in the face of rising sophistication, requirements and complexity is to cut corners in the development and maintenance. Traditionally, the first items cut are reviews, testing, training, and documentation. While there might be the appearance of a short term savings, the long term increased costs are dramatic because systems will neither be maintainable or evolvable. They will only be replaceable.

For example, if a system cost $2 million in 1985 (that is, about 30 staff years (at about $60,000 per staff year for both direct and indirect costs) which computes a cost of $70,000 for analysis and design (life cycle cost divided by 30), that same system will now cost about $5.4 million. If however, because of CASE, repository and code generators, the ratio can be brought to 1 to 2, then the cost to develop the same system today will be about $2.7 million. The cost model to justify the funds for a quality systems environment is not just its cost but also its savings, and whether its costs exceeds its savings.

To determine whether the current systems environment is already tuned to make maximum use of CASE, repository and code generators, the following questions must be answered:

- Are all the existing systems documentation within automated metadata repositories or are able to be placed within automated metadata repositories such that maintenance impact analysis can be quickly and easy?

- Are all the business concepts uniformly defined within a single metadata repository and are mapped to implementations of these same concepts so that whenever any concept changes the impact of that change can be quickly assessed?

- To the maximum extent possible, are all the data and process semantics implemented through clauses in DBMS schemas rather than in computer programs?

- Are all database and information systems metadata preserved in their three forms, specification, implementation, and operations along with the mappings between the of metadata?

The extent to which these four questions are affirmatively answered is the extent to which the information systems organization can either maintain the existing systems development and maintenance costs or reduce these costs.

At the center of optimizing the data processing systems development and evolution and maintenance environment is effective data standardization. Data standardization embraces both semantics, value sets, and ultimately the seemingly infinite quantity of data. Achieved, the semantics of standardized data, metadata, can be used multiple times. That minimizes the analysis and design component of the 1st implementation version. This in turn, caused reductions in the detailed design through training components of the 1st version (assuming CASE, repositories, and code generators). When the 1st implementation version is significantly reduced then the costs of the evolution and maintenance versions also dramatically decrease. Simply put, data standardization, the catch phrase for regularizing semantics, value sets and ultimately data streams, is key.

Finally, the key to success with CASE, repositories and code generators is data standardization. There are no long term savings with CASE, repositories and code generators without data standardization

1.5 Contents of this Book

Chapter 2 presents the most common reasons for data standardization failure. Chapter 3 provides case studies of two different database application classes to illustrate the different types of data standards problems. Chapter 4 provides an overview of the parts of an effective data standardization effort. Chapter 5 presents the most critical component of the data standardization effort, data semantics. Chapter 6 presents the second half of the data standardization effort, establishing standard value sets. Chapter 7 provides six distinct types of cases that require special data standardization attention. Chapter 8 provides the work breakdown structure for a data standardization effort. Chapter 9 provides a summary of the entire effort and some final advice.