The Realization that Business Rules *Are* Metadata

By capturing their business rules as metadata, enterprises consistently deliver effective strategic information -- on demand, to the right people, in the right format. They also control the only constant in enterprise management -- Change.

This paper describes methods and best practices for discovering, documenting, and using business rules as metadata.

Table of Contents

Introduction

What is a business rule? According to the Business Rules Group, there are two basic definitions of a business rule.

"From the I/S perspective, a business rule pertains to the facts of the system that are recorded as data and to the constraints on changes to the values of those facts. The business perspective of business rules involves the behavior of people in the business (human activity) system. Because these perspectives are distinct, we have a definition of 'business rule' for each of these perspectives."

> Business Perspective: A directive, intended to influence or guide business behavior, in support of business policy that has been formulated in response to an opportunity, threat, strength, or weakness.
>
> I/S Perspective: A statement that defines or constrains some aspect of the business. It is intended to assert business structure, or to control or influence the behavior of the business.

Business rules are used to capture and implement precise business logic in processes, procedures, and systems (manual or automated). They also provide the basis for expert systems.

Enterprises that take a model-based, architected approach to software component development can use business rules to refine the models and create better designs. An enterprise that properly documents its business rules can also manage change better than one that ignores its rules.

Business rules may be any of the following:

- Definitions of business terms
- Data integrity constraints
- Mathematical and functional derivations
- Logical inferences
- Processing sequences
- Relationships among facts about the business

All business rules are examples of enterprise metadata. They can be defined as metadata, modeled as metadata, and, most importantly, they can be implemented as metadata for an enterprise's operational and strategic information management systems.

Implementing business rules as metadata is the most rigorous and, at the same time, most flexible approach to business rule implementation. This is in contrast to other implementation approaches.

- ❑ Process-driven approaches can be rigorous, but they are by no means flexible. In any process-centric approach, implementing the business rules is fairly straightforward, but, because they are typically implemented in code, changing them can be difficult and labor intensive.

- Procedure-driven approaches, characterized by manuals and checklists, are certainly flexible, but they are not rigorous. Procedures can be changed very easily. However, procedures are only as rigorous as their users choose them to be. People can and will ignore procedures.

Advantages of Managing Business Rules as Metadata

There are several advantages that an enterprise can realize from using business rules as metadata:

Allows maximum flexibility

By documenting and implementing business rules as metadata elements, an enterprise can easily make changes to rules as policies, guidelines, strategies, and environments change. Software component code does not need to be changed. Only the content of the "business rule tables" is changed.

Reduces system maintenance

Not having to change software component code every time a business rule changes will obviously reduce system maintenance. Instead of having to re-design, re-code, re-compile, re-integrate, and re-implement components, business rule changes are a simple process of changing entities in a logical model, automatically generating changed tables in a database design, and then generating the DDL or XML necessary to change or alter implemented tables.

Simplifies system design, development and implementation

Developing software components is a lot simpler when rule-based processing logic is limited to evaluating the content of one or more database tables.

Rules can change without affecting implementation

When business rules are modeled and implemented as metadata, changes in the rules have little or no impact on installed software components. Changes also have limited impact on component design, development, and implementation. Thus, systems built from these components can truly reflect the most current business requirements -- and the requirements can change even during the final stages of development.

Ensures that systems fully support business needs

Because metadata reflects business information requirements, any system designed to process the metadata will consequently meet those requirements. And it will continue to meet the requirements even as they change over time.

MIS personnel don't need to learn the intricacies of the business

Perhaps the greatest benefit of implementing business rules as metadata is that only business analysts and information architects need to know and understand the business. Designers and developers can concentrate on improving their technical expertise and using the latest tools to create the systems that implement the rules, but they don't need to have an in-depth understanding of either the rules or the metadata that represent them.

Business Rule Techniques and Best Practices

Documenting and implementing business rules as metadata is fairly simple. However, any approach, no matter how excellent or useful, can still be performed poorly. This is particularly true when the approach seems to be simple. To ensure that business rules are captured correctly and implemented effectively, requires using a few techniques and best practices.

In order to implement business rules well, you need to document the rules well. There are three basic characteristics of good business rules:

Explicit expression
Any statement of business rules needs an explicit expression, either graphically or as a formal (logic-based) language.

Declarative nature
A business rule is declarative, not procedural. It describes a desirable, possible state that is either required or prohibited.

Coherent representation
A single, coherent model for all the kinds of business rules is desirable.

The techniques and best practices that follow allow business rules with these characteristics to be discovered, documented, and implemented effectively.

If you can model it, you can implement it

The basis for good business rules is a single, coherent model for all the rules, now matter what their source or where they are implemented. There are several reasons for using such a model:

- Having a formal repository for all rules promotes rigorous and thorough analysis of rules and their requirements.
- A single model, partitioned into subsets that reflect a portion of the enterprise, can reduce complexity of rules and simplify their collection.
- Using a common model and common types of notation enhances communication among business analysts, between analysts and information owners and users, and between analysts and designers/developers.
- Most importantly, a model of rules and requirements will assist in overcoming the tendency in many enterprises to take a "ready…fire…aim" approach to software component development.

Using a model for all software component development efforts, not just business rule management, has the following benefits:

Incorporates best practices

Modeling something before you build it is a best practice of any effective engineering methodology, and works especially well for software engineering. Other best practices borrowed or adopted from other engineering disciplines are useful only if a model-based approach to software component development is used.

Meets the unique needs of the enterprise

All business requirements are documented in the model and linked to the database tables and software components that implement them.

Promotes data sharing

One of the characteristics of a logical enterprise model is that data elements (and business rules) are defined only once no matter how often they are used or in how many systems they are implemented. This promotes sharing data.

Eliminates redundant analysis

If a business rule or requirement has already been modeled for a prior project or application, there is no need to completely re-analyze it in order to use it in a new application.

Encourages reuse -- not reinvention

If a previously modeled business rule or other logical model element is useful in a new project, it stands to reason that any design or implemented software component should be equally useful and should therefore be reused.

Allows effective component development AND management

By encouraging reuse and sharing, a model-based approach to software engineering also provides a means to manage component development. Logical business objects are modeled that represent related clusters of information. These logical data objects are combined as needed for specific functions or applications. The data objects, including metadata, are then combined with logical control flow models that represent object behavior and are used as the basis for multiple physical component designs. The designs can then be implemented on any platform using any application development tools.

This means that logical models that accurately reflect business rules and requirements can be used to effectively manage software component development -- quality requirements become quality models; quality models become quality designs; quality designs become quality components that make up quality information systems (see Figure 1).

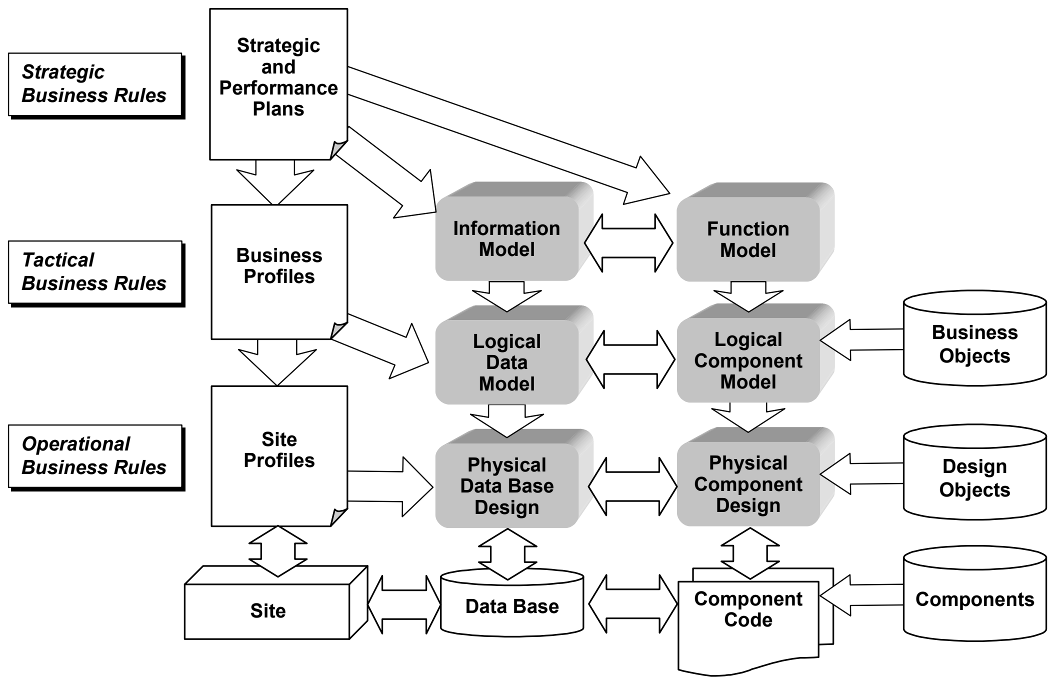

Figure 1. Model-Based Component Development

Business Rules in Business Models

Business rules take several forms when documented in enterprise business models. There is one best representation that is appropriate for each particular type of business rule. For example, strategic business rules are best modeled as simple business statements, while operational business rules are best modeled as static data instances and derivations. The following are the model artifacts that can be used to represent business rules.

Business statements

A business statement is a simple declaration. Business statements should be written in business language. They should be concise and clear. They should represent a single concept or idea. They should state business requirements, not system requirements. The kinds of strategic business rules that can be modeled using business statements are illustrated in Figure 2.

Data entities and attributes

A relational data model, by its very nature, represents business rules. For example, by definition, an "entity" represents an information element that has importance to the enterprise. It can represent a person, place, thing, or event of significance. An "attribute" is used in a logical data model to represent a data item or characteristic of an entity. For example, if the entity is EMPLOYEE, then the attributes might be *employee name*, *employee number*, etc. Some attributes

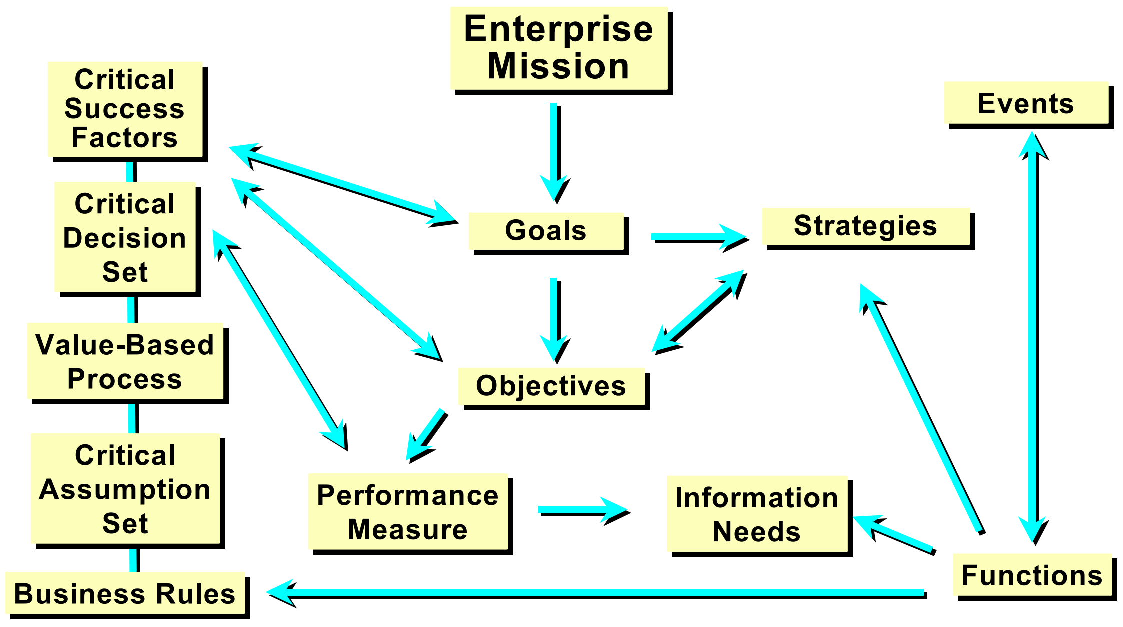

Figure 2. Strategic Business Rules

are used to uniquely identify an instance of the data element (e.g., an employee number uniquely identifies only one employee).

Entity associations

An association between two data elements in a logical data model (or physical database design) also illustrates a business rule. The existence of the association is a bi-directional rule. "Every EMPLOYEE has a JOB." "Every JOB is filled by an EMPLOYEE."

These rules can be further refined by describing the degree of the association. "Every EMPLOYEE has only <u>one</u> JOB." "Every JOB is filled by <u>more than one</u> EMPLOYEE."

The nature of the association can also be described as mandatory, optional, or becoming mandatory. "Every EMPLOYEE <u>must</u> have one and only one JOB." "Every JOB <u>must</u> filled by at least one EMPLOYEE." "Every PAYCHECK <u>must</u> be payable to only one EMPLOYEE." "Each EMPLOYEE <u>will eventually</u> have a PAYCHECK."

These examples of business rules may seem quite simple and obvious. But if you change a single word the meaning can change significantly. Also, they may not be so obvious to a programmer developing systems that implement these rules.

Domain values

The author has extended the basic theoretical relational modeling approach to facilitate capturing business rules. We document hierarchical and structured instances of an entity within the entity itself. Meta-data is actually "hard coded" in the entities as static values; e.g., the entity CUSTOMER TYPE could contain the following values which represent business rules about the specific types of customers for this enterprise:

customer type	customer type description	customer type component
Internal	**Subsidiary or Business Unit**	**$custsys\custyp_i**
Federal	**US Government**	**$custsys\custyp_f**
State	**State or Local Government**	**$custsys\custyp_s**
Commercial	**Non-Government**	**$custsys\custyp_c**
Partner	**Strategic Partner or Affiliate**	**$custsys\custyp_p**

A static entity containing metadata instances is used to create a static table design with the same instances, which is then used to generate the Data Definition Language (DDL) that will create a database table that actually contain the metadata. Not only does this allow us to have databases that are guaranteed to accurately reflect business requirements and rules, we can design and develop software components that hardly ever change. All decision elements, domain instances, and even called component names can be stored in tables, not code. Only the content of the "rule" and metadata tables change -- not code.

Derivations

In strategic information models -- particularly those that represent data warehouses, decision support systems or executive information systems -- it is important to be able to document how measures and metrics are derived from other data elements. These derivations take the form of algebraic functions coupled with SQL-like statements; e.g., Total Positions Filled = COUNT (JOB ASSIGNMENT [where JOB ASSIGNMENT.job id = JOB.job id [where job assignment start date <= TODAY & (job assignment end date >= TODAY | job assignment end date = NULL)]])

Rule expressions

A rule expression is used to capture the logic component of business rules. Rule expressions evaluate to either true or false depending on the existence of combinations of data element occurrences and other rule logic. Rule expressions use algorithmic functions combined with extensive use of qualifying expressions to precisely depict the desirable status of complex interrelationships. Rule expressions provide a rigorous mechanism for modeling business rules. There are two types of rules:

- Simple Rules

  Used to enforce or preclude the existence of specific patterns of relational logic. Can be represented by single values that indicate "optional," "mandatory," and "not allowed."

- Complex Rules

  Used to control the amplitude of occurrences of a specific pattern of business logic. Complex rules control combinations more complex than simple "Yes/No", "On/Off." They amplify the degree and/or nature of an association between two or more data elements. Complex rules are usually represented by two values that indicate a range of possibilities (0 = none, 1 = one, X = a fixed integer, N = no fixed maximum).

For example:

Min	Max	Rule
0	0	Not allowed
0	1	Optional, may have one
0	X	Optional, may have up to X
0	N	Optional, may have more than one
1	0	Not meaningful (illogical)
1	1	Mandatory, one and only one
1	X	Mandatory, may have up to X
X	N	Mandatory, must have X or more

Rule entities

Some modeling tools provide the ability to model special rule entities in logical data models. These are entities that contain rule metadata as static values. A rule entity is usually an intersecting (associative) entity between two other static entities. Their purpose is to control population of one or more other entities (and ultimately the database tables generated from the models). Rule entities may be used for either simple or complex rules.

- Figure 3 illustrates a rule entity containing simple rules: *Planned Maintenance* can never be either *Urgent* or an *Emergency*. An *Emergency* or *Storm Damage* can never be placed *On Hold*. *Storm Damage* can never be *Proposed*.

  (PROJECT STATUS RULE controls the creation of occurrences of PROJECT.)

- Figure 4 illustrates a rule entity containing complex business rules: Only one person can be *Responsible* for a particular product. Products don't have to have *Owners*, *Managers*, *Customers*, or *Contacts* assigned to them. There can be any number of *Owners* or *Customers* for a product. There can only be one *Manager*, but there may be up to three *Contacts*.

  (PRODUCT PERSON REASON controls creation of occurrences of the intersecting entity PRODUCT PERSON.)

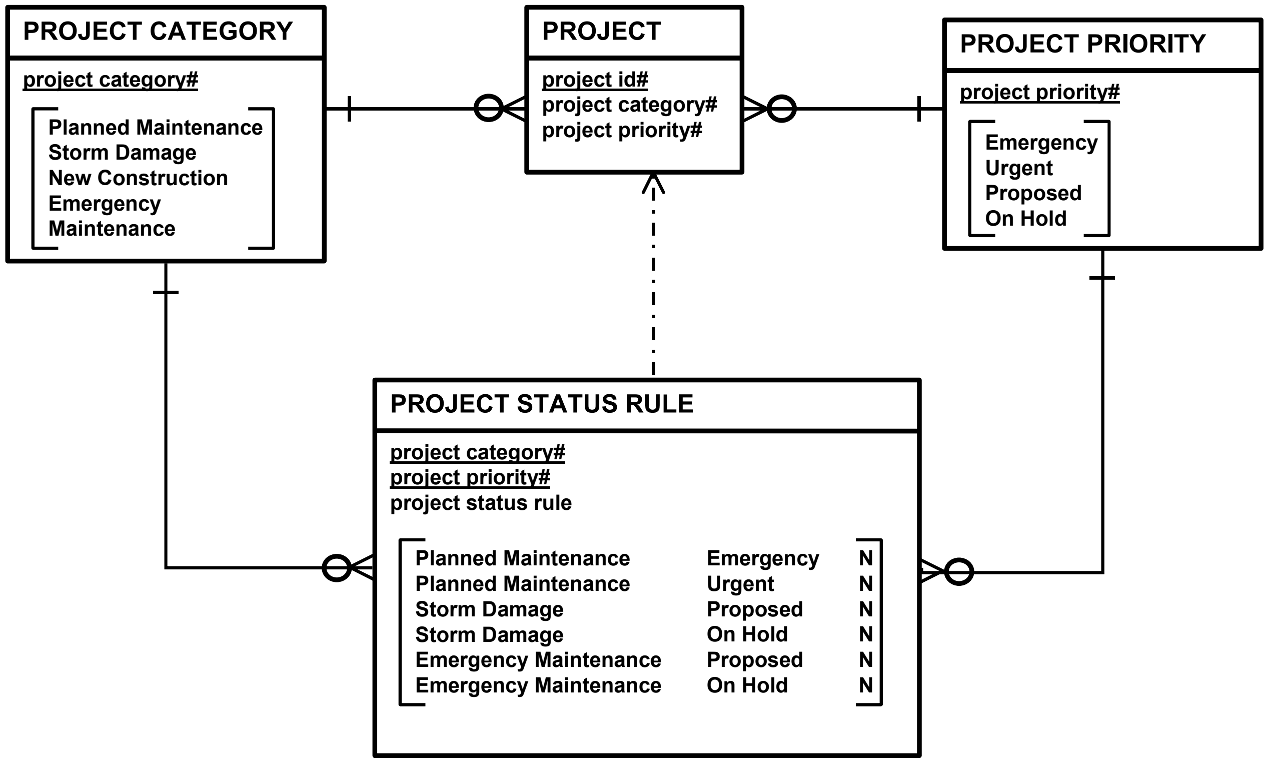

Figure 3: Simple Meta Data Rule

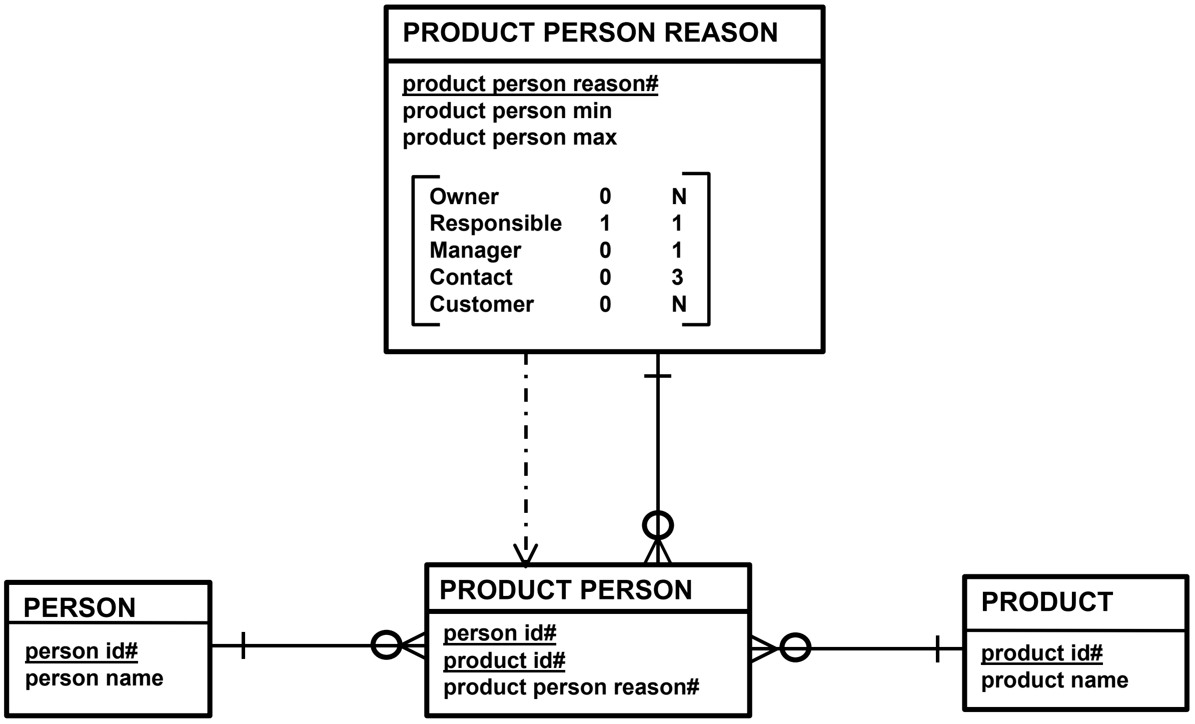

Figure 4: Complex Meta Data Rule

❑

Business Rule Linkages

All rule-based entities should be linked to the management statement(s) that documents the business rule being modeled. This linkage supports and facilitates tracing of requirements, particularly when the linkages are extended to include physical database and component designs as well as implemented components and tables.

Business Rule Life Cycle

The author recommends a controlled iteration of specific tasks to discover, document, model, and implement business rules as metadata. The business rule life cycle is illustrated in Figure 5.

The author calls the business rule life cycle *"e-Engineering"* because it applies equally to all Enterprises -- large and small, public and private, for profit and non profit. It is Effective, Efficient, and is Essential for E-business and E-commerce. For more detail see the author's paper entitled "e-Engineering – The Unified Methodology."

The life cycle involves a multi-phased approach that coordinates strategic, operational, and organizational demands. Our business rule life cycle is a unified methodology that incorporates the best practices of business and information management. This methodology involves all the activities that organizations perform to improve productivity, gain and maintain competitive advantage, optimize resources, deliver quality products and services, and meet internal and external customer expectations and demand.

The following is a typical cycle:

1. Describe the enterprise mission in a brief statement of purpose: what the enterprise does, how, and for whom.
2. Make assumptions and gather data about external factors: government policies, rates of inflation, markets, and demographic changes.
3. Assess enterprise strengths and weaknesses.
4. Establish goals, objectives, and measures linked to the enterprise mission.
5. Develop strategic and operational plans to meet the goals and objectives, utilize strengths, overcome weaknesses, counter threats, and address opportunities.
6. Design/re-design and integrate cross-functional processes to meet goals and objectives.
7. Implement information systems that support enterprise processes and assist in decision-making.
8. Evaluate performance to ensure that goals and objectives are being met.
9. Reevaluate and change goals, objectives, processes, and measures as necessary.

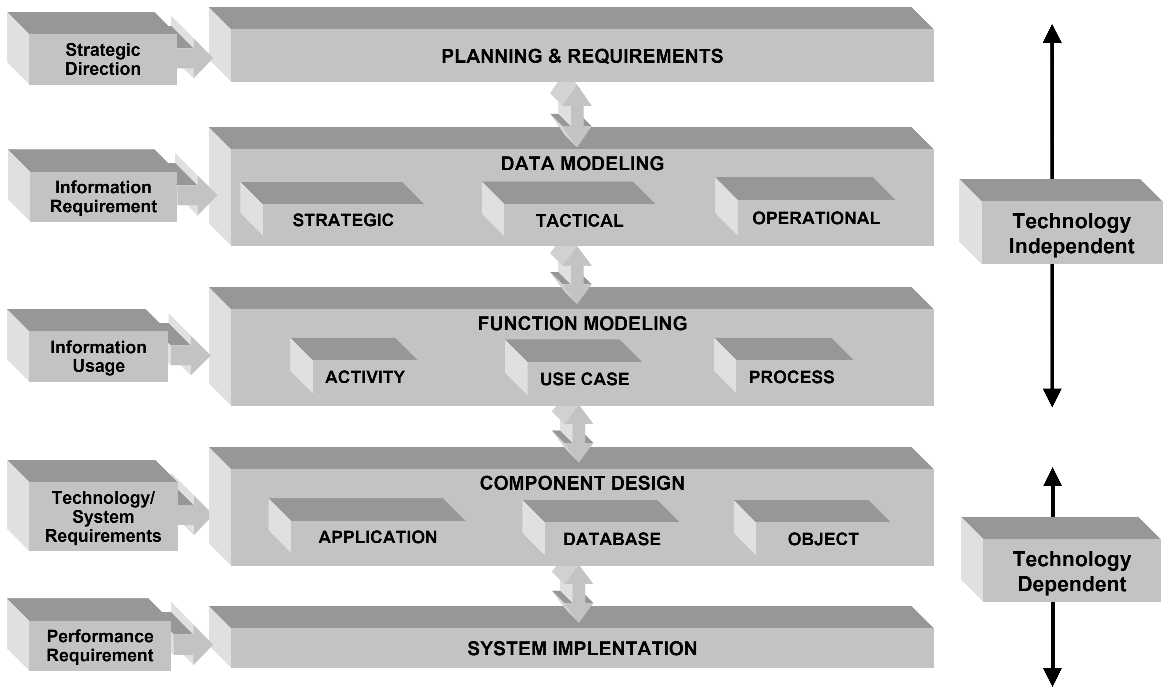

Figure 5. Business Rule Life Cycle

During each of these steps, business rules are discovered, documented, and modeled. First, business requirements and business rules are defined in real-world terms: goals, strategies, objectives, tasks, critical success factors, performance metrics, etc.

As business requirements are being defined, an Enterprise Architecture model is developed using logical models that are both unambiguous to developers and understandable to business experts. The Enterprise Architecture is made up of logical data, process and activity models that represent the information and business rules necessary to support the defined business requirements.

In the logical models, the definition of an entity, accompanied by its available processes (behaviors), forms an encapsulated unit that describes the functionality (logical component model).

Model components and linkages -- requirements to logical model elements to physical design objects -- provide the best structure for component management and make change management both easier and faster.

The logical data and process models resulting from this approach are technology independent and reflect the business goals and requirements of an enterprise. They can be subsequently revised, amended, and refined independently of the implementation environment of any physical systems.

The extra time spent getting requirements right, and designing logical objects, pays significant dividends in reducing the time spent designing and building software components. It also ensures that the organization can consistently develop quality information systems.

This approach for discovering, documenting, modeling and implementing business rules, coupled with model-based quality software component development, reflects the principles of the Software Productivity Consortium and can help organizations achieve progressively better levels of the Software Engineering Institute's (SEI) Capability Maturity Model (CMM™). The CMM is today's most widely used benchmark for software process improvement.

Summary

Business rules ARE metadata so it makes sense to define and deploy business rules as metadata. This approach captures rules as logical model elements and implements them as database tables, triggers, and object actions. The key to this is fully understanding the rules, documenting them consistently, and building conceptual information models and logical data models that accurately reflect the rules. This makes the rules *visible.* The rules can be modeled, including specific values for metadata. These migrate automatically to any database design generated from the models and, ultimately, are automatically inserted into the database tables when they are created or altered from the design.

This means that a business rule can be documented once in a logical model and still be part of multiple system designs and implemented databases. This one-logical, many-physical representation of business rules as metadata also allows a significant change in software component design. Components can be designed to access database tables for rules and do not have to include complex decision tables and rule-based processing logic.

For more information about this topic please contact:

Visible Systems Corporation
711 Atlantic Avenue
Boston, MA
contact@visiblesystemscorp.com

Alan Perkins has been a Systems Analyst on the White House staff, Director of the US Army Data Processing School in Germany, Vice President of R&D for a virtual corporation, Vice President of Consulting for Visible Systems Corporation, and General Manager of a high-tech consulting firm. He has provided information and enterprise management consulting to numerous companies, associations and government agencies.

Mr. Perkins specializes in Enterprise Architecture Engineering. He helps clients quickly engineer enterprise architectures that are actionable and adaptable. His approach results in architectures that enable and facilitate enterprise initiatives such as Corporate Portals, Enterprise Data Warehouses, Enterprise Application Integration, Software Component Engineering, etc.