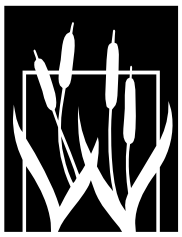

Whitemarsh
Information Systems Corporation

Return on Investment (ROI)

Business Information System Environments

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

Table of Contents

Whitemarsh ROI Savings Summary

While data models are essential for business information system development, business information systems are where "the rubber meets the road." It is through business information systems that enterprise-policies are executed and the data resulting from those policy executions is captured, stored, analyzed and reported.

Like enterprise data models, the traditional processes for business information system creation are both expensive and fraught with error. Whitemarsh, over the years has participated in IT projects where business information system development techniques have been dramatically improved and made cheaper. The ROI derived from all these improvements is 7.7.

Supporting Links	
Link Area	**Link**
Enterprise Data Management Areas	http://www.wiscorp.com/roi_businessinformationsystemenvironmentsenterprisedatamanagemen.html
The Data Administration News Letter Articles	http://www.wiscorp.com/roi_businessinformationsystemenvironmentstdan.html
Short Papers	http://www.wiscorp.com/roi_businessinformationsystemenvironmentsshort_papers.html
Clients	http://www.wiscorp.com/roi_businessinformationsystemenvironmentsclients.html

1.0 Issue

Business Information Systems development is not a easy task. Even less easy is Business Information System maintenance when the business information system isn't conceived and constructed with maintenance in mind.

Even if business information systems are easy to accomplish and maintain, developing one is a strategic activity of an enterprise. That is because business information systems are a combination of both data and process, which are at times conceived, planned, developed, and maintained with different goals, methods, and strategies. At times, data and process are at loggerheads with each other, wherein each undervalues the other.

There have been times when data modelers have declared that the data needs of a business information system developer do not exist. There have been other times when a business information system developer proclaims that data modelers are just not needed. Both assertions are wrong, and if allowed to persist will cost the enterprise way more in badly designed

databases and badly engineered business information systems than would have occurred if there had been a helpful interrelationship.

Data Models, the schematic representation of stored data is the expression of an enterprise's persistently retained policy execution. Thus policy, as represented through data, can only be anecdotal if it is not formally defined through data models. In contrast, Business Information Systems are the enterprise procedures through which policy is executed and ultimately retained in a persistent state. Data Models and Business Information Systems go together as do Policies and Procedures. They are hand and glove.

Given there are significantly fewer well-engineered data models than business information systems that use data from data models, building and/or having a data model first, but certainly not in isolation, is ontologically prior to building and/or having a well-ordered set of business information systems.

The basis for this ontological priority is simple. Specifying a business information system without first having an overarching data model that acts as both a target and a constraint, produces 4.6 times more business information system work products to construct. It needs to be stated, however, that there is a synergistic relationship between the state of a data model and the unfolding and evolving state of a business information system.

Changes to the data model certainly occur during the development of the business information system. These changes are unlikely, however, to ever surpass the 4.6 work product multiplier caused by not having a well-designed data model. Thus, building the data model remains ontologically prior to building the business information system.

Because conceiving, constructing and maintaining an enterprise set of business information systems is never a small amount of money, it is important to accomplish business information development in the most efficient and effective manner practical. A key question at the outset of an business information system effort is just what should be in the business information system? Simply stated, what should be its complete functionality?

The Information Systems Plan is the key to unlocking the answer to this question. The right process content of a future Business Information System is the correctively identified set of existing Business Information Systems and Database Objects that have been assigned to a particular Resource Life Cycle Node. Once a collection of Business Information Systems and Database Objects are chosen, the overarching scope of the future Business Information System is thus constrained by the requirements of a successful Resource Life Cycle Node state transformation.

Only after the Business Information Systems and Database Objects are determined, can the database models and the business information system models finally be fully specified. At that point, the key objectives for future Business Information System development are:

- Maximize non-redundant development
- Maximize development of reusable data and processes
- Maximize ability to evolve and maintain
- Maximize Use of Metabase System

The basis for dimensioning this issue and for the computation of traditional approach costs and changed approach costs and finally the determined ROI is based on a “rule of thumb.”

The size and cost of a business information system is based on the count the quantity of third normal form database tables, which is then multiplied by 80 to get the total of function points. That result is then multiplied by the function point cost, which can range from $200 to $400 (depending on the mode of traditional implementation) to get a ball-park estimate for the business information system.

Thus, if a database has 200 tables, the function points is 16,000, and the ball-park cost, at $200 per function point, the cost of a single business information system is about $3.2 million.

Many enterprises have 20 to 50 major business information systems. Thus the total implementation cost of just the business information systems inventory is about $64 million.

2.0 Solution Approach

The accomplishment of the solution involves not just absorbing the cost reductions provided from those derived from both the ROI for Data-Centered Development and Management, and also the ROI from Manufacturing Integrated, Interoperable and Non-Redundant Data Models but also from dramatically improving the approaches, methods, and infrastructure employed to develop business information systems. Four such improvements are described in this section.

2.1 Maximize Non-redundant Development

The first item, Maximize Non-redundant Development, is accomplished by ensuring that every enterprise database and business information system only contains the components necessary for

the scope of either the database or business information system. This is accomplished through their development according to the sequence set within the Resource Life Cycles Nodes.

As stated in the Issue section, constricting the scope of the business information system to just that which is required to achieve the state change represented by the Resource Life Cycle Node is important. This ensures non-redundancy, and a "bright line" set of conditions of where the business information system starts, and finally a bright line of where the business information system ends.

The most immediate vehicle for setting out the business information system's bright lines is the scope of the enterprise data model subset that is allocated to achieving the state change for the Resource Life Cycle Node. If a properly defined the Resource Life Cycle Node's subset data model does not have the database tables needed for an existing business information system, that business information system's scope is too large and must be reduced to fit just the needs of the Resource Life Cycle Node. The only exception to this would be any tables that bridge one Resource Life Cycle Node's ending state to the starting state of the next Resource Life Cycle Node.

2.2 Maximize Development of Reusable Data and Processes

The second item, Maximize Re-Use of Data and Processes, is accomplished through the generation of business information systems. To achieve that, these generators must have at least two main layers. The first is the business information system specification layer in the form of metadata. The second layer contains the resultant generation of the actual business information system language statements, that, when compiled result in an executing business information system.

Supporting the first layer must be an Integrated Development Environment (IDE) that manages all the created business information system specifications. This IDE environment must provide facilities for defining, managing, and employing all the specification objects that are able to be included in any generated business information system. Supported too must also be text editors, compilers, linkers, and debugging tools. Beyond this IDE infrastructure there must be:

- Control Templates
- Template Language
- Procedure Language Programming
- Object Oriented Programming
- Data Access Language
- Near Real Time Programming & Execution

Control Templates. Because most aspects of a business information system are similar from one business information system to the next, a very key component of the specification layer are "specification writing" templates that when invoked for every business information system control such as for menus, windows, browses, buttons, and the like, produce the specifications of these controls that, in turn enable the generation of the program language statements that accomplish the business information system control.

It is essential that these specification templates be such that the end-user created special purpose procedure language code can be specially embedded "before, during, and after" any control. The embedded procedure language code must be segregated such that the template control is unaffected, and vice versa. Such end-user embedded procedure language code isolation ensures business information system regeneration.

Template Language. The Template Specification Language is a special language that generates Control Templates, Procedure Language, and the Object Oriented Class and Method Language. The template language can be used not only to generate Control Templates, but also IT Technique Templates and Application Templates.

IT technique templates include the production of control to display and manipulate calendars as well as perform calendar-base calculations, the generation of single file recursion hierarchies, networks, and many other IT techniques such as email processing, encryption, FTP, and the like. These facilities are then able to be integrated within business information systems without exiting, accomplishing the task, and then restarting the business information system.

Application Templates are pre-written templates that accomplish whole or in part applications that can be included within a normal business information system. Examples would include Application Templates that integrate HR or Accounting.

Procedure Language Programming. Procedure language program first and foremost indicates the focus of its purpose: the procedure. Here the action word is programming, and the vehicle for accomplishing the program is through a language. Procedure language programming has been here since the dawn of the stored program computer. Such computers were without purpose until directed into action by a stored computer program. Computers then accomplished the creation of specific outcomes such as generating a payroll, accounting for finances, scheduling product deliveries, and the like.

Over the years, stored program computers shrank in size from room-sized ones to hand-helds, while at the same time, grew in performance from the slow execution of a very small stored programs, to the near simultaneous execution of many complex programs.

A procedure language contains the traditional set of commands such as assignment, looping, case, insert, change, and delete. Procedure languages also contain a methods for processing strings, calendars, and the like, and also the ability to create data structures and to add, delete, and modify records within these data structures. Included as well is the ability to define and automatically process relationships (both data and pointer-based) relationships across records from different data structures. All these fundamental capabilities in programming languages have not significantly changed over the past 50 years.

What has changed, however, is the ability to reuse common collections of procedure language statements by initially creating generalized variables within the procedures that can, at run time, be bound to local variables and/or database table names, columns, or keys. Once the binding has been invoked, the generalize procedure routines can be invoked and execute just as if they had been originally created with the local variables and/or database table names, columns, or keys.

Object Oriented Programming. Object oriented programming takes its name from its focus, the object, which represents something concrete (in IT terms), such as a file, person, account, transactions, and the like. The word, oriented, means that the action word, programming has to be oriented towards the object. Here, action words are things like Create, Destroy, Insert, Modify, and the like.

The various other language processes that can occur to instances of the object are the traditional set for example, assignment, looping, If-Then-Else, case, string processing, calendar-based processing, and the ability to create data structures and to add, delete, and modify records within these data structures

Taken together object oriented programming represents a conceptual inverse to procedure language programming. In procedure language programming, the procedure is the essential property, and what and how the procedure acts are the accidental properties. In object oriented programming, the object is the essential property and the actions performed on the object are the accidental properties.

The paramount objective of object oriented programming is to build multiple higher-levels of generalization and abstraction with the goal of removing the lower level bindings that make code brittle, unable to be re-used, unable to be re-purposed, and expensive to maintain. These levels of generalization and abstraction enable encapsulation and inheritance.

Process objects should always be in “Process Third Normal Form,” that is, the name of the object reflects its complete and singular purpose, internally employed data is defined within the object, and there is one entrance, one exit.

Object Oriented Programming capabilities and their definition, execution and evolution infrastructures and environments have grown over the years. It is now easily possible to create many layers of generalization and abstraction along with encapsulation and inheritance. These enhancements enable the development of many layers that, when compared to traditional procedure language program collections that accomplish the same result, are many times smaller, equivalent in execution performance, and far more efficient in development, reuse and evolution.

Near Real Time Programming & Execution. It matters not one wit about the sophistication of the control templates, the template language, and as well the differences in procedure language programming and object oriented programming if the Integrated Development Environment (IDE) is primitive. Here, primitive means that the various steps to create programs had to be created, compiled, executed, and evolved in a serial fashion. Or that only whole programs could be created, executed, and evolved versus small well-bounded subsets of functionality within the overall program.

Sophisticated IT environments accomplish the creation, execution and evolution of requirements, architectures, designs, implementations, and continuous improvements, in near real time. Such environments support repeatable cycles of "Conceive-Posit-Prove-Reflect-Optimize."

2.3 Maximize Ability to Evolve and Maintain

The third item, Maximize Ability to Evolve and Maintain, is greatly affected by three factors: The IDE, the re-use abilities, and the ability to deploy a near-real time environment within which business information systems are conceived, created, deployed, and maintained.

2.4 Maximum Use of Metabase System

The fourth item, Maximum Metabase System use, causes a savings in the initial development of specifications and in the maximum re-use of already developed specifications for use in evolution and maintenance.

The Metabase System models provide context for the business information system so that it is seen within an overarching enterprise framework related to the six interrogatives of: who, what, when, where, why, and how. These and others are the work products of Information Technology and are considered to be the intellectual property of IT.

3.0 Solution Engineering

Solution engineering is founded on a number of items. Some of these are accomplished by the enterprise and others are information technology procured products. Addressed are the solution engineering facilities that enable:

- Maximization of Non-redundant Development
- Maximization of Development of Reusable Data and Processes
- Maximization of Ability to Evolve and Maintain
- Maximum Use of Metabase System

3.1 Maximize Non-redundant Development

Accomplished, this ensures that any given business information system be only what is required virtually no overlap with adjacent business information systems. This is certainly addressed through the use of Resource Life Cycle Analysis as the mechanism to restrict the unneeded and overlapping scope of both data and business information system. Scope restrictions are accomplished during the Information Systems Planning process.

3.2 Maximize Development of Reusable Data and Processes

Accomplished, this ensures that every business information system object is created such that it can be used elsewhere in the business information system in a reusable manner without having to be duplicated.

First and foremost is the business information system's integrated development environment (IDE). Picking the right one that, at its very core, enables the automatic specification of about 95% of the business information system objects that, in turn generate the actually programming language statements which are compiled and linked to both executables and DLLs comprise the overall created business information system is critical. This alone causes the biggest savings in resources.

The types of business information system objects include for example, menus, lists, forms, and a myriad of other controls that complete windows, display calendars, and perform embedded processes such as pick lists, radio buttons for choices, sorting of browses by column headers, and the like that are required by the needed business information systems to be generated must be identified, specified, and then employed as selection criterial for picking the right IDE.

It is important too that these business information system objects include the ability to add procedure language or object oriented code within embed points that allow the accomplishment of business relevant processes.

Critically important too in the IDE is the ability to create business information system objects that may be missing. This can only be done if the IDE is fundamentally constructed through the use of business information system object templates that are constructed through a template-based language, that, in turn generates the business information system templates.

A final component of the IDE is the creation of a visible and maintainable data and business information system architecture that can be easily created and maintained. The IDE must contain easy to use button/icon-based processes that cause the generation of the computer language code, its compilation, and its linkage into an executable.

Once a highly engineered and work enhancing IDE is selected and deployed, the business information system component specifications need to be initially generated. Such a generation forms the basis for generated specification tuning and for the addition/modification of additional business information system functional specification.

Such an initial generation is best accomplished through the use of a subset of an overall enterprise data model appropriate for just that business information system's scope. That is because, a well-ordered data model automatically imposes functionally organized processes in support of database table record creation, update, and interrelationships.

The quality and sophistication of any specification generation, based on the scope of the control templates, automatically create these business information system functional specifications. There are two aspects of any control template. First is whether there are control templates for virtually every process or activity needed for the business information system, and second, when one or more of the needed business information system processes or activities are needed but are missing, are there control template language mechanisms that can be deployed to create the missing template?

Two examples, one simple and the other complex are: gender, and data-based "Bill of Materials" network data structure. For the simple example, gender, to automatically have the business information system component materialize the values for a gender column in a database table, where the determined values are to be Male, Female, and Unknown, the control template should support the specification of two types of "pick lists:" Radio Buttons, and Select Lists.

The "radio-button" alternative in which the value is displayed next to button, the button, when pressed, causes the selected data value, Male, Female, or Unknown" to be generated as the value

for the gender column. The "select list," alternative, when employed, displays the set of values from which one can be selected, which, in turn becomes the value for the gender column.

To enable the automatic appropriate choice from these two alternatives, the control template needs to have a rule that counts the pre-determined values set out in the application data model for the gender column, and if the count is at or below a certain quantity, the result is a radio-button specification and if above that quantity, the result is a "select list" specification.

For the more complex example, Bill of Materials, the control template is more sophisticated. First, it depends on a very sophisticated configuration of database tables, and second, the control template contains a whole collection of "prompts" that must be valued before the control template generates the complete specification for the Bill of Materials processes that includes 1) a window for the creation of Bill of Materials items, 2) another window that supports the creation of the interrelationships among the items, and finally 3) the process functionality that displays the hierarchy of interrelationships among the items despite the fact that these hierarchies can in fact be network based data structures.

In a highly engineered IDE that is buttressed by both control templates and a sophisticated control template language, vendors can dramatically increase work productivity, as each control template can have their own special ROI.

For example, the time needed to "hand code" a "Bill of Materials" network data structure and all supporting procedure/objected oriented windows and processes can take a staff month or more once all the IT specifications for such a facility have been created.

If there is a sophisticated control template creation language, an IDE-allied independent vendor could produce a control template that generates the same specification, which, by definition enables the actual generation of the procedure language or object oriented business information system code. If use of a Bill of Materials control template only takes one full staff day to deploy, there's a 20:1 ROI on just that one control-template based IT technique.

The final component of Maximizing the Development of Reusable Data and Processes is the application of a learned behavior coupled with an accommodating IT environment. The behavior, Near Real Time Programming & Execution, is either enabled or prevented by the choice of the IDE, its specification and code generation, and its ability to immediately compile and begin execution of just the business information system component that is currently being implemented.

If these facilitating mechanisms fail to exist, the creation of the business information system will be straight-jacketed into a serial process that is slow, error-prone, and expensive to evolve and maintain. That is because the straight-jacketed environment will result in a water-fall approach

that requires whole horizontal swaths of the business information system to be accomplished before proceeding to the next lower-level swath.

If the IDE et al environment is accommodating, however, higher layers of the business information system only needs be sketched to the extent needed for the next lower-level wedge of business information system implementation to begin. This cycle of "Conceive-Posit-Prove-Reflect-Optimize" invariably both uncovers new opportunities for better functionality, but also thd need to modify the specifications of the previous higher levels.

If a high quality IDE et al environment exists, the speed at which the most costly components are accomplished, that is, the detailed design and programming, especially when compared to the traditional water-fall approach to their creation, will more than compensate for any necessary redesigns and re-implementations of both the higher levels and of any re-design and re-programming.

3.3 Maximize Ability to Evolve and Maintain

Accomplished, this ensures that created business information system component are very close to 100% generated by the IDE and therefore not in need of any evolution and maintenance, or have been developed through embeds containing the procedure language code and/or the object oriented code in a form that is easily understandable and maintainable.

The two key components to this ability to evolve and maintain are the IDE et al and also the Metabase System. The IDE et al enables the complete exposition of the business information system, layer by layer in its specification form, its actual computer code form, and in its executing form. These three layers can be quickly and easily navigated, maintained, and reconstituted such that revised versions of business information systems can be released in a matter of days to weeks rather than months to years.

The Metabase System gives visibility across all enterprise projects, databases, business information systems and all supporting Metabase System models so that required changes in one database can be immediately shown to impact specific business information systems and then interrelated databases and business information systems.

3.4 Maximum Use of Metabase System

Accomplished, the Metabase System environment provides, through its capture of all necessary work products, the specifications of all other environment components that provide complete context and necessary definition. These include for example:

- Data models (Data elements, Concepts, Database Logical and Physical)
- Mission-organization-function models
- Resource Life Cycle Analysis Models
- Database Object Models
- User acceptance Testing
- Data Integrity Rules
- Information Needs Analysis
- Use Cases

Collectively these Metabase System models not only store and not only provide context for the business information system, but it does so from within the overarching enterprise framework. This quickly and easily enables the discovery of conflicts and duplication. Such discoveries are possible because the Metabase System database is designed to eliminate redundancy and to maximize interrelateability and integration.

Together this brings together the six interrogatives of: who, what, when, where, why, and how not just into single columns but whole descriptions of Information Technology intellectual property.

4.0 The ROI

4.1 Traditional Calculation

From the ROI on data models, an enterprise is likely to have 100 reasonable sized business information systems with about 200 database tables each.

Business Information System Traditional Cost								
Quantity Tables	**Function Points per Table**	**Total Function Points**	**Function Point Cost**	**Total Initial Cost**	**Main-tence Cycles**	**Percent Main-tained**	**Main-tenance Cycles Cost**	**Total Cost**
200	80	16,000	$200	$3.2 million	5	25%	$3.2 million	$6.4 million

Notes:

1. This includes the expected overlap resulting from not employing Resource Life Cycles Analysis.

2. This presumes 5 cycles of maintenance from not starting first implementation at Cycle 6 (ROI, Data Centered Development and Maintenance)

3. For an enterprise inventory of 100 such systems, the total initial cost is $640 million.

4.2 Changed Approach Calculation

Business Information System Traditional Cost								
Quantity Tables	**Function Points per Table**	**Total Function Points**	**Function Point Cost**	**Total Initial Cost**	**Main-tence Cycles**	**Percent Main-tained**	**Main-tenance Cycles Cost**	**Total Cost**
160	80	12,800	$50	$640K	3	10%	$192K	$832K

1. This excludes the expected overlap resulting from applying Resource Life Cycles Analysis. (200 * .8 = 160)

2. This presumes 3 cycles of only 10% maintenance by starting first implementation at Cycle 6 (ROI, Data Centered Development and Maintenance)

3. For an enterprise inventory of 100 such systems, the total initial cost is $83.2 million.

4.3 ROI Summary

The ROI created from the traditional approach cost to the changed approach is 7.7. That is a dramatic reduction is cost.