Analytics8™

ETL Projects Don't Have To Be So Hard

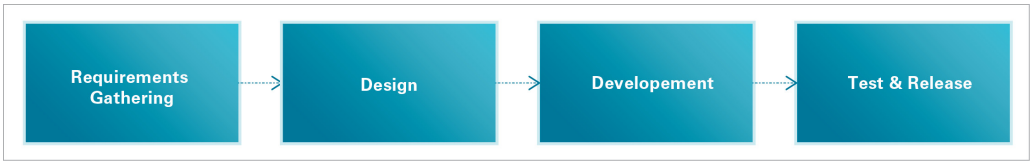
Project stakeholders are consistently baffled that implementing ETL (Extract, Transform, Load) on a typical Business Intelligence (BI) initiative consumes most of the project's resources. Most BI project stakeholders, especially Business Analysts and Project Managers, tend to think of ETL as plumbing: it happens behind the scenes and is as simple as moving data from source to target. However, ETL is more than just plumbing. It has the power to correct data errors and transform raw data into information that can be readily consumed by business users. In The Data Warehouse ETL Toolkit, Ralph Kimball and Joe Caserta state that the ETL portion consumes upwards of 70% of all resources required to build a data warehouse. Our experience confirms that the ETL portion consumes the lion's share of project resources, is difficult to estimate the effort involved, and is often a point of friction between developers and project managers. So what can be done to manage ETL projects to success?

The Crux of the Problem

Poor data quality is the biggest contributor to delays in the ETL portion of the project. Inevitably, all of the data problems that have been lurking in source systems, e.g. summaries that do not tie to source or transactions missing critical data elements, are exposed to the light of day when an ETL system is built. Common culprits are business rules not supported by the data, missing or incomplete data, and source data structures that do not comply with designs. While facing the reality of source data, quality is often an important and unexpected benefit of data warehousing; these kinds of discoveries are often made well into the development phase of the project with deadlines looming and no time left for revisiting requirements and design, let alone revising development. In short, there is a great deal of "you don't know what you don't know" about the data adversely impacting project execution.

Additional problems result when requirements gathering and design activities are not done thoroughly. For example, users might want data refreshed by 8 AM but because the project team did not prioritize performance, the data refresh batch process runs well into the afternoon. Or the design calls for the extraction of customer information from a legacy system that is soon to be retired. Project managers tend to not place a great deal of importance on these activities, thinking that the time is better allocated towards development. In our experience this is a mistake. Without taking the time to do thorough and detailed requirements gathering and design activities, the unexpected problems multiply and cause further delays.

The complex problems peculiar to ETL suggest that there is no "silver bullet" to these types of projects; there is no "one-size fits all" solution. Analytics8 recommends a series of best practices for limiting the impact of the unexpected. Specifically, we recommend managing the project in iterative cycles and including best practices specific to ETL in all requirements and design phases.



continue »

ABOUT ANALYTICS8

Analytics8 is a business intelligence (BI) and data warehousing enablement and optimization consulting firm. Though we are experts with many tools and technologies, we never lose sight of the business reasons for implementing technology.

Let us show you how to make the most of your data.

ITERATIVE CYCLES ARE KEY

The most effective way to manage ETL projects is to deliver manageable chunks of value in iterative cycles.

Best Practice: Manage the Project in Iterative Cycles

The most effective way to manage an ETL project is to deliver manageable chunks of value in iterative cycles. The elements of each cycle include business assessment (requirements gathering), design, development, testing, and release. The scope of effort for each cycle needs to be both meaningful and manageable, yet not so large as to attempt to deliver the entire data warehouse in a single release (e.g. the waterfall method). The iterative approach limits the impact of unanticipated data quality issues and provides a means for establishing trust and credibility among the project stakeholders by successful delivery of the discrete components in each iteration.

The iterative release approach provides the user community with visibility into the eventual end-product from the earliest phases of the ETL project with the delivery of specific data sets at the end of each phase. Enabling direct data access to project stakeholders provides leeway for discovering unknowns and fine-tuning requirements and design. This is particularly important because most users take it for granted that data is moving into the target system and do not anticipate data anomalies in their source systems. For example, a particular source system may be updating records more frequently and in ways not fully documented during the initial requirements phase. By detecting this issue in an early iteration it can be captured and modified in the requirements and incorporated for delivery in a later phase.

Best Practice: Data Profiling

Data profiling is the process of verifying business rules and data integrity. It must begin during requirements gathering so rules for handling data anomalies can be presented to the users in a readable format. For example, there may be records of retail transactions in the source system without a retail location or store specified. In this case the users may want the record filtered out, defaulted to a generic 'not-specified' location, or assigned to a specific location. This style of surveying the quality of the data in the underlying source systems and validating it against requirements and business rules is essential for preventing problems downstream in the BI stack. Active data profiling needs to be an ongoing part of all requirements gathering and design phases.

TAKE A CRAWL-RUN-WALK APPROACH

Data profiling is complicated, but there are tools available to automate the process and reduce the overall time and efforts required to complete this activity.

Data profiling is a complicated and painstaking process, but there are tools available to automate the process and reduce the overall time and effort associated with this activity. It often makes sense to take a crawl-walk-run approach to data profiling. The initial effort is often creating SQL queries against relational data sources or manual inspections of data files. The next step is to move into automated validation scripts, or, for example, running a Business Intelligence report to verify all customers meet profile criteria. The highest level of maturity is to implement a data quality tool. Naturally, much of the profiling you will do is dependent on the type of project and type and availability of data sources. The important point is to gain as much familiarity with the data as early in the project as possible. Questions to keep in mind include:

- How much data is available?
- What is the range of values that I am seeing?
- How complete is the data?

Thorough data profiling may reveal the need for correction of the source systems or indicate the need for a master data management (MDM) strategy. For instance, there may be different standards between departments for identifying products with different codes referring to the same product. In cases such as these it makes sense to develop an MDM strategy for standardizing data conventions across an enterprise. The earlier these issues are identified the better.

continue »

Best Practice: Requirements

Understanding the full set of business requirements on a typical ETL project is a mammoth undertaking. In our experience it is nearly impossible to articulate the full set of requirements up front given all the unknowns in the source systems. If the project is managed in an iterative fashion, it is possible to solicit a limited set of requirements that can be met in a short time frame, typically six to eight weeks. The idea is to have a concise set of requirements that deliver value to the end users with the understanding that they will be refined and augmented in later cycles. For instance, the first iteration may only include requirements for bringing in transactions at their most granular level with a limited set of dimensions (i.e., time, product and customer). Later iterations can refine these requirements to add summaries, complex dimensions and hierarchies, even budget and forecast data. Very often it makes sense to limit the in-scope data sources to a single system.

User communities will typically have a list of data elements or reports that are considered high priority or "must haves" and possibly a longer list of elements that are "nice to have" but not required. The high priority items are what should be tackled in early iterations. The business users are often most familiar with the data from high-priority items, and are able to clearly articulate requirements, making the ETL development a straightforward process.
Some requirements germane to the ETL space are typically outside the range of experience of most in the user community. For this reason, the following topics must be presented to the user community, with care and skill, so they understand the importance of their participation in the endeavor.

Slowly Changing Dimensions

Project teams often do not understand the implications of slowly changing dimensions for historical analysis that may be of great value to the user community. Nevertheless, the possibilities for historical analysis enabled by slowly changing dimensions must be presented to the project stakeholders, and presented in a way that is easily understood. Slowly changing dimensions are data mart or warehouse dimensions (e.g. Geography, Customers, and Products) whose data change over time. For instance, a retailer may open and close stores over time or periodically rearrange geographic regions while a manufacturer very typically creates new products and product lines while retiring older ones. These changes will almost certainly be reflected in the dimension's source system(s) and the possibilities for study must be fully vetted with the stakeholder community. The process may start out with questions such as "Does your product hierarchy ever change?" and "Do you need to be able to look at current sales from different historical perspectives?" The demonstration of specific and graphical examples is often helpful, as in Figure 1 below. In technical terms, gathering requirements around slowly changing dimensions suggests a particular design in terms of the dimension types (e.g. Type II-all history is maintained vs. Type 1-no history maintained). Row-level metadata is required to capture the relevant information (e.g. row status being and end timestamps, row active flag). This topic is articulated in greater detail under the Data Audit best practice.

FIGURE 1: SLOWLY CHANGING DIMENSION

Some stores have been reassigned to the "NORTH" region from the "WEST."

STORE_CODE	STORE	REGION TY	REGION LY	SALES TY
4803023	San Jose, CA	WEST	WEST	$234,567
4941840	San Bernadino, CA	WEST	WEST	$412,898
4991111	Portland, OR	WEST	WEST	$561,189
4991112	Phoenix, AZ	WEST	WEST	$898,011
4991417	Tacoma, WA	WEST	NORTH	$213,482
4991426	Boise, ID	WEST	NORTH	$771,832

	WEST TY	WEST LY
Current Year Total Sales by Region	$3,091,979	$2,106,655

continue »

Missing or Late-Arriving Dimension Values

Late arriving dimensions are unique values, such as product codes or customer numbers, referenced in the source for fact data but not in their respective dimension table. These values cause significant data problems and must be handled by the ETL team. The missing dimension table record will cause the fact row to be flagged as invalid and appears to be a loss of data. This is typically addressed by inserting the missing dimension into the fact table with the appropriate default attributes. Later, when the full set of attributes becomes available, the dimension table can be updated.

DON'T FORGET ABOUT THE LATE COMERS

Having a process for updating requirements, even for late-arriving dimensions, assures that requirements are kept up to date and accurate throughout the process.

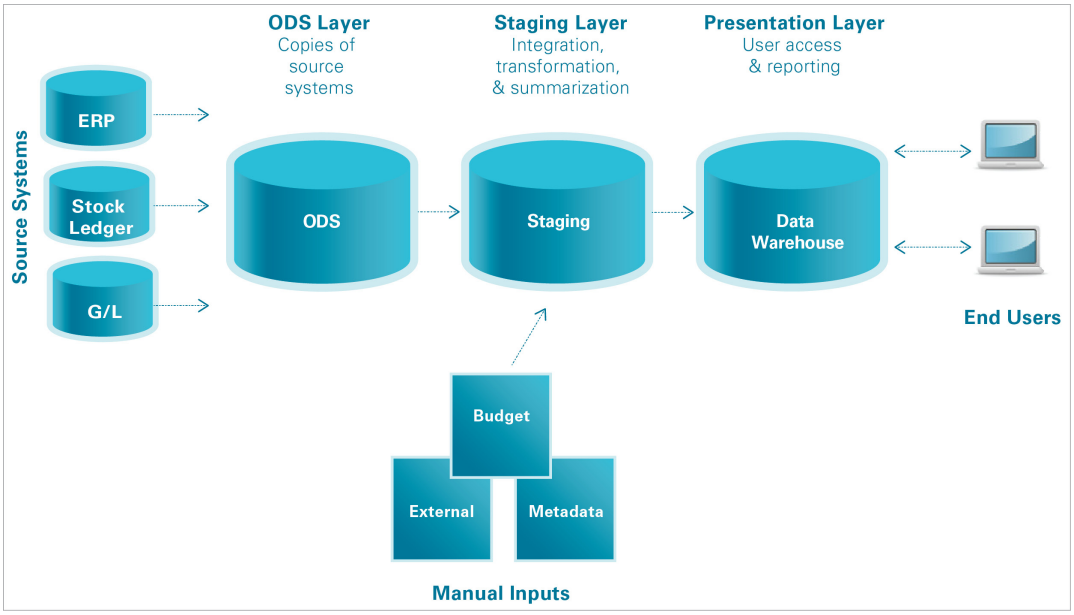
The Best Practice is to spend time and effort up-front in gathering the requirements for handling these values. Defining dimensions in the design phase avoids having to adjust to surprises later. Having a process for updating requirements, even for late-arriving dimensions, assures that requirements are kept up to date and accurate throughout the project. Accurate requirements lead to high quality software and accurate information. Consistent, valid results are the key to user confidence and acceptance of a new system.

ODS, Staging, and Data Warehouse Layers

Analytics8 recommends delineating layers of data storage between ODS (Operational Data Source), Staging, and Data Warehouse layers. Typically the ODS is a replication of data from the source systems, such as a General Ledger system or ERP, with no intervening data transformations. The staging area is where the work of the ETL system is done, including data transformations, dimension key lookups, data summaries, data integration, and the maintenance of historical data. The Data Warehouse layer is the final home for the data (see Figure 2 below). Delineating these layers streamlines the design process and strongly suggests where the various ETL processes need to occur, simplifying the development process.

FIGURE 2: DATA WAREHOUSE DATA FLOW

The flow of data in a data warehouse goes between ODS, Staging, and Presentation layers.



Data Purging or Archiving

Data purging is often neglected during requirements but should be included because it impacts the longevity and usability of the system. User communities rarely think about how much data to keep beyond the minimum required for analysis and generally prefer to have everything available. However, excessive data volumes can ruin report query performance and extend the time it takes to refresh data on a periodic basis. It is a good idea to establish requirements for data retention up front and account for it in the ETL design so volume related performance problems do not rear their ugly head at some later point in the project.

continue »

Data Granularity and Summary Reporting

Data granularity needs to be communicated during the requirements phase so any requirements for summary reporting can be understood and taken into account during design activities. Typically, requirements for summarization are not included as part of the ETL requirements because they affect reporting requirements most directly. However, for performance reasons, it is important to keep the need for data summaries in mind because they are often most efficiently accomplished in the ETL. The need for summary tables is easily added to later development phases as users gain experience with the system.

Data Availability

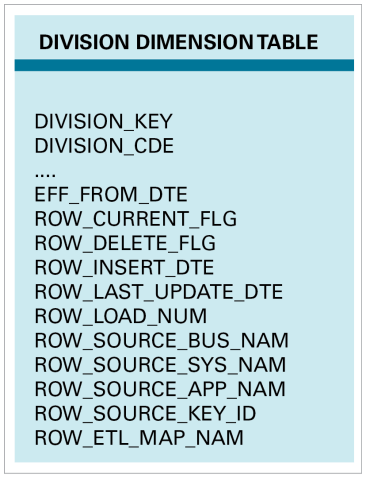
Data availability refers to the 'by when' users will have access to the latest data for reporting. With demands increasing for near real-time data and mixed workloads being applied to source and target systems, it is important to uncover this requirement early and start setting expectations around what can be accomplished given the operating environment and tools available. The urgency of this requirement will be amplified in the design phase once the availability of the source systems is determined following the inventory of source systems.

Data Audit

Requirements for data audit should be fully vetted with user and technical communities. Briefly, data audit is the reporting of "Who? What? When?" for every row in the data mart, or warehouse, enabled by row-level metadata (See Figure 3 below) or even a separate audit dimension. Typically, an organization will have its own standards for tracking metadata at the row and column level but it will be the development team's responsibility to determine if more metadata needs to be captured to satisfy the requirements of the project's stakeholders. Recall that row level metadata is particularly important for enabling Slowly Changing Dimensions.

FIGURE 3: DIVISION DIMENSION TABLE

Row level metadata fields required for capturing data lineage and history.



Best Practice: Design

Issuing a complete ETL project design at the start of a project is also a massive and nearly impossible task given all the inherent complexities. Instead, design phases should be conducted in the same fashion as requirements gathering: limited in scope and performed thoroughly with successive iterations adding to prior ones. At the end of every design phase there should be a deliverable document sufficient to start the next phase of development. The design needs to be simple enough to be easily understood by all project stakeholders so it may be necessary to create separate functional and technical documents. Once again, there are elements specific to ETL that need to be covered within each design phase:

continue »

Source System Inventory

Too often the development team takes it for granted that source systems are available for data extraction whenever they want it, so it is a good design practice to inventory the source systems for ownership and availability. This approach entails documenting the source systems and identifying potential road blocks that can be worked out before tensions and surprises derail the project (see Figure 4 below). This practice builds on the data profiling step started earlier in the project. The elements to document include:

- Where is the data?
- Who owns it?
- When is the data refreshed?
- When it is available for extraction?

Keep in mind that any source system probably has a host of users who do not like it when their systems suffer performance issues because someone in some far-away office is pulling data for their BI application.

FIGURE 4: SINGLE SOURCE SYSTEM INVENTORY

Documenting the source systems and identifying potential road blocks.

SOURCE	BUSINESS OWNER	IS OWNER	PLATFORM	LOCATION	AVAILABILITY	REFRESHED	DATA SOURCE DESCRIPTION
G/L	Bud Dennis	Julie Young	OS390/ MVS	Data Center	After business hours	Weekly- Monday AM	Production General Ledger
Stock Ledger	Bud Dennis	Julie Young	OS390/ MVS	Data Center	After business hours	Weekly- Monday AM	Production Stock Ledger
Manual Files	Human Resources	Chris Knight	Excel	Email	Ad-Hoc	Ad-Hoc	Headcount File

Source to Target Mapping

Laying out the source-to-target mapping on a field-by-field basis is often deemed too simple or intuitive to warrant the time spent on this step. However, we have found this to be a costly mistake. There is nothing in design that you can do to save more time in development and later phases than creating a complete source-to-target mapping. The mapping itself is typically a document with every column in the target data structure on the right-hand side with its source or sources on the left and all intervening transformations in the middle. The exercise of accounting for all sources and targets and intervening data transformations removes ambiguity and guesswork from the development process and immediately exposes where requirements are insufficient. It also provides a concise blueprint for rapid development.

One the next page is an example of a source-to-target mapping containing all target and source columns with all intervening transformations. Also documented are the types of operation against the target (e.g. 'Truncate and Reload' or 'Update else Insert') and any relevant comments or notes.

FIGURE 5: SOURCE TO TARGET MAPPING

This chart is an example of a source-to-target mapping containing all target and source columns with all intervening transformations.

Mapping Type	Truncate Reload	**Target Table Type**	Oracle
Target Table Name	PRODUCT_TO_LINE_STG	**Load Frequency**	Quarterly file feed
Target Table Keys	PRODUCT_NUMBER_KEY		
Comments	The staging table is truncated before processing. Source rows are checked for referential integrity. If a row has an -1 value returned from a dimension lookup, the row is written to an error table.		

Source Database Name	Source Table Name	Source Column Name	Target File Column Name	Transformation Rule
Flat File	product_to_line.csv	Product	PRODUCT_NUMBER	Upper(Ltrim(rtrim(product)))
		Product	PRODUCT_NUMBER_KEY	Lookup : PRODUCT_D Join: PRODUCT_NUMBER Return: product_number_key If return value is -1, flag row for error
		Product Line	PRODUCT_LINE	Ltrim(rtrim(product line))
		Product Line	PRODUCT_LINE_KEY	Lookup: PRODUCT_LINE _D Join: PCRL_ PRODUCT_LINE _CDE Return: PCRL_ PRODUCT_LINE _KEY If return value is -1, flag row for error
		MY	PRODUCT_MODEL_YEAR	Ltrim(rtrim(my))
		MY	MODEL_YEAR_KEY	Lookup: YEAR_D Join: YEAR Return: MODEL_YEAR_KEY If return value is -1, flag row for error

Figure 5 accounts for every column in the table we want to load and specifies its source and any intervening transformation logic. Looking at a couple of example rows in the figure itself, the target column named 'PRODUCT_NUMBER' is sourced from the product_to_line.csv file. The only transformation performed on its way to the target is a trim operation (represented as LTrim(RTrim()) pseudo code). The target column PRODUCT_NUMBER_KEY is sourced from the same field but is the product of a lookup against the PRODUCT_D table. It returns a -1 in the case of an error (e.g. the Part Number is not found in the lookup table). There are also header rows with information regarding the type of mapping and specifics of the target table.

We recommend engaging the user community in completing this task whenever possible. Staff analysts typically have a strong knowledge of the data sources and should be utilized. In addition, it gets users directly involved in the project at an early phase of delivery. Many ETL tools automate the creation of source to target programming code directly from spreadsheets and documents, so we recommend taking advantage of this capability.

Measure Complexity

Analytics8 recommends using some type of measurement to determine system complexity. "Complexity" can be measured with a count of the number of moving pieces, such as the number or tables and columns for both source and target structures. The need for understanding system complexity is greater when there are many data transformation and cleansing operations. In these cases it makes sense to count the number of columns and tables and then multiply them by weighting factors to gain a sense of where most of the complexity in a system lies and where more development and testing time may need to be allocated. Higher weightings can be given for more complex transformations. For instance, fact data from a source system may need to be aggregated and product codes cleaned up on the way to a target fact table. While this process is not a hard science, its value is in illuminating where system complexities lie, and where additional project resources are required.

Figure 6 (on the next page) is an example estimate of the complexity of various source-to-target mappings. In this case, all measures are multiplied to arrive at an overall score with a higher number suggesting greater complexity. To arrive at our score, we first multiply the number of columns in our source by the number of columns in our target to get a general

sense of the number of moving pieces. Then we adjust for more complex data manipulations (Transformations), large volumes of data (Row Count), and a large number of lookup operations (Lookups) by assigning a weighted factor for each (1.0 is neutral, > 1.0 suggests greater complexity). Thus, our measure of complexity can be calculated as follows:

Complexity Score = Source Columns (Count)* Target Columns (Count) * Transformations (Weighted Factor) * Row Counts (Weighted Factor) * Looookups (Weighted Factor)

FIGURE 6: SIMPLE MEASURE OF SYSTEM COMPLEXITY

The process for loading the GEOGRAPHY table is much simpler than loading CLAIM_FACT.

SOURCE	TARGET	SOURCE COLUMNS	TARGET COLUMNS	TRANSFORMATIONS	ROW COUNT	LOOKUPS	SOURCE
Claims_processing	CLAIM_FACT	33	24	2.7	1.4	1.5	3207.6
Office_master	GEOGRAPHY	22	28	1.3	1.0	1.2	960.96

Best Practice: Automating Data Validation

Automating data validation activities is an essential part of every ETL process. Testing an ETL process means confirming that it did what it was designed to do. If a job was constructed to load a month's worth of transactions from a point-of-sale system, it needs to confirm that a month of data was loaded; if a job was constructed to summarize those transactions, it needs to confirm that all the relevant transactions were included in the summary.

Building an automated regression test bed for validating ETL involves one of two proven methods. The first way is to create a QA database of expected results and comparing the those expected results to the Target data in the warehouse under development. The second method uses pairs of queries against both source and target to make sure data has been extracted, transformed and loaded successfully. Both methods are equally effective and the scale of the operation and the complexity of the ETL determine which method is right for the situation.

The first method requires the QA team to derive an independent ETL process. This is done in isolation of the development team so any differences can be traced back to possible sources of error. The sources of error could be requirements, incorrect interpretation of the requirements in creating the ETL scripts, ETL scripting errors by the development team or ETL scripting errors by the QA team. Once the ETLs are executed, an automated comparison of the two data sources (QA and the Target system) is performed, differences are analyzed, and defects identified.

The second method requires the QA team to account for the source-target data mapping in the creation of SQL query pairs. The source query results in the data to be processed by the ETL. The target query is created to return the data after the ETL. Both result sets are loaded into a data base repository where multiple validations are performed to evaluate if a query pair test case passes or fails.

The results can be reviewed and failures can be quickly evaluated to determine if a defect has been detected. Due to the rapid nature of automated testing, defects can be swiftly detected, remediated and re-tested. Even when the defects are traced back to requirements, and significant changes are necessary, automated data validation provides an efficient means for assuring the accuracy of your data. ETL testing is vital to providing feedback on the design and construction of your data warehouse.

TESTING IS VITAL

Hundreds of hours can be saved by identifying ETL errors, missing requirements, and more.

Hundreds of hours can be saved by identifying ETL errors, missing requirements, and more at this stage of the data warehouse development project. Automated data validation regression tests allow for higher test coverage, faster release cycles, and better data quality in your production systems. Automated testing enables large numbers of BI and DW systems to be supported by a single QA team.

USE AN AUTOMATED ETL TOOL

Using an automated ETL tool reduces development time and eases ongoing maintenance.

Best Practice: Tool Selection

We recommend using an ETL tool for all development. The alternative is hand-coding SQL scripts or executables in a high-level programming language. Using a tool reduces development time and eases ongoing maintenance. Furthermore, tool based solutions make it easy to collect and publish metadata- how many rows were loaded, how many rejected, etc.

In general, ETL tools come in two varieties, file based and SQL based. File based ETL tools (e.g. Ab Initio, Ascential Data Stage, Informatica PowerCenter) are always faster and scale with large data volumes more effectively than SQL based tools (e.g. Oracle Warehouse Builder, SQL Server Information Services). Choose a file based tool when performance or scalability is a priority. However, since many of the SQL based tools are shipped with their relational database counterparts, they make a compelling case when software budgets are strained and performance is not a top priority.

It is a mistake to assume that the native scripting capabilities of modern data warehouse appliances (e.g. Netezza etc.) are sufficient for handling all the required ETL transformations. While these tools are certainly impressive in their ability to support large volumes of data and heavy query traffic, they are not ETL tools and thus do not provide the full range of functionality for transforming raw data into usable information.

Conclusion

ETL projects are complex, with unknown data problems lurking in source systems and too many important details glossed over in requirements and design in a rush to meet a deadline, with each new complication reducing the chances for project success. Implementing ETL-specific requirements gathering and design best practices, in conjunction with an iterative project life-cycle, thoughtful data profiling and ETL tool use, offers the best chance for delivering a value-rich BI implementation without ETL-inspired budget and timeline overruns. Finally, managing the project in such a way as to keep the users and stakeholders involved greatly enhances the chances for success. Remember: no project is successful unless your customer says it is!

Recommendations

- Start Data Profiling in conjunction with or even before requirements gathering activities
- Manage the project in iterative cycles of limited scope and optimally a single data source
- Present topics germane to the ETL space so the user community understands:
  - Slowly changing dimensions
  - Missing or late-arriving dimension values
  - Data purging or archiving
  - ODS, Staging, and Data Warehouse Layers
  - Summary reporting
  - Data availability
  - Data audit
- Include ETL best practices during the design phase
  - Source system inventory
  - Source to target mapping
  - Measure complexity
- Automated testing of the ETL process results in greater test coverage, faster cycle times and more accurate information
- Use an ETL tool and select a file-base tool when performance or scalability is a priority, a database-centric tool when cost is a constraint and performance is not a concern