PharmaSUG 2014 - Paper AD21

Clinical Trial Data Integration: The Strategy, Benefits, and Logistics of Integrating Across a Compound

Natalie Reynolds, Eli Lilly and Company, Indianapolis, IN
Keith Hibbetts, Inventiv Health Clinical, Indianapolis, IN

ABSTRACT

From the 2012 FDA/PhUSE Computational Science Symposium Working Group 3 Vision Statement: “Data integration has always been a challenge both for industry and the agency.” One approach to address the challenges of integrating and converting data across studies is to build a clinical trial integrated database (IDB). This paper explores the strategy of creating and maintaining IDBs across studies and throughout the life cycle of a compound, the benefits IDBs provide and an effective and efficient metadata-driven system to deal with the logistics of delivering a portfolio of IDBs. First, we’ll explore the principles behind the strategy of clinical trial data integration. These principles drive decisions regarding when to create and update IDBs for a compound. Next, we’ll discuss how IDBs provide cost and time savings by supporting many objectives including regulatory submissions, answering customer questions and data mining. Finally, we’ll examine the logistics of delivering on that portfolio by examining the metadata-driven system that allows efficient delivery across different compounds. Metadata is commonly used to define standards across the industry, but this system illustrates the benefits of using metadata to define data transformations as well. By utilizing metadata concepts, the same sets of tools can be utilized regardless of the standards used in the input studies. Readers of this paper will leave with a solid foundation of the strategy, benefits, and logistics of clinical trial data integration.

INTRODUCTION

Pooling and analyzing data for a multitude of studies across a compound can be a challenging task. It takes years for a compound to go from discovery to FDA approval, and during this time the world of pharmaceuticals continues to evolve: data collection and reporting standards change, personnel changes, new indications are identified, and so forth. One approach to address these challenges is to build a clinical trial integrated database (IDB). This paper details a strategy to govern decisions on when to build integrations. This strategy, along with the availability of the integrated data, has proven to have a large number of benefits. A metadata-driven system was designed to help with the logistics of creating and supporting these integrations.

THE STRATEGY

BACKGROUND

IDBs utilize a consistent format to pool together all clinical trial patient data (typically Phase II - IV, non-observational trials) for a compound into one database. IDBs support many objectives including regulatory submissions, answering customer questions and data mining.

WHY ARE IDBS SO IMPORTANT?

When compound IDBs do not exist, answering questions can be timely and costly.

- Example #1 from Pharma ABC - no compound IDB
  - A regulatory agency question with a 30 day deadline
  - No IDB exists and only 70% studies in the electronic data repository
  - Regulatory agency asked for data from the remaining studies
  - Team created hand generated tables for the remaining studies and was given 30 more days to finish
- Example #2 from Pharma XYZ - no compound IDB
  - Customer question - Response cost millions of dollars and took 6 months
  - 70% of total time spent finding & preparing data (See Figure 1)
- Example #3 from Pharma 123 - compound IDB exists

- A regulatory agency request - search compound trials for specific terms, per regulatory agency specifications
- Utilizing the compound IDB, completed the search and validated the results in the required time

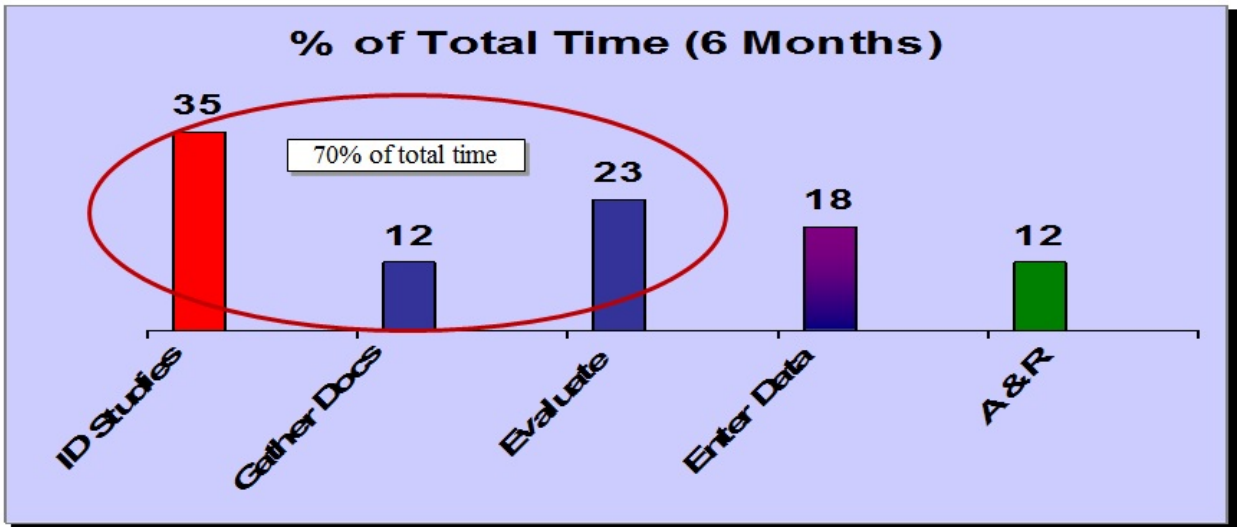

Figure 1. Total Percentage of Time Spent for Example 2

IDB STRATEGY - FORMAT CHANGE

As an industry, we must align to CDISC data standards and output format. By doing so, this allows investigators, researchers and regulators to combine data easily from studies across companies for their research. This is important for FDA submissions and data transparency projects.

As a sponsor, we made a conscious decision to update our internal clinical data flow (See Figure 2) to deliver datasets formatted to industry standards. The new process takes advantage of metadata to automate data transformations. The current IDB process takes advantage of the newly available study metadata and aligns with the new data flow.

IDB format aligns with the strategy for Trial Level Data where:

- Observed data are in SDTM format
- Reporting and Analysis data are in ADaM format

New IDBs will be in industry standard of SDTM and ADaM format. Currently, most IDBs are in company proprietary format. Each compound team will follow and complete a transition plan that will determine IDB format and when to transition to the new instudry standard. The transition plan should take into account many factors, including where the compound is in its life cycle.

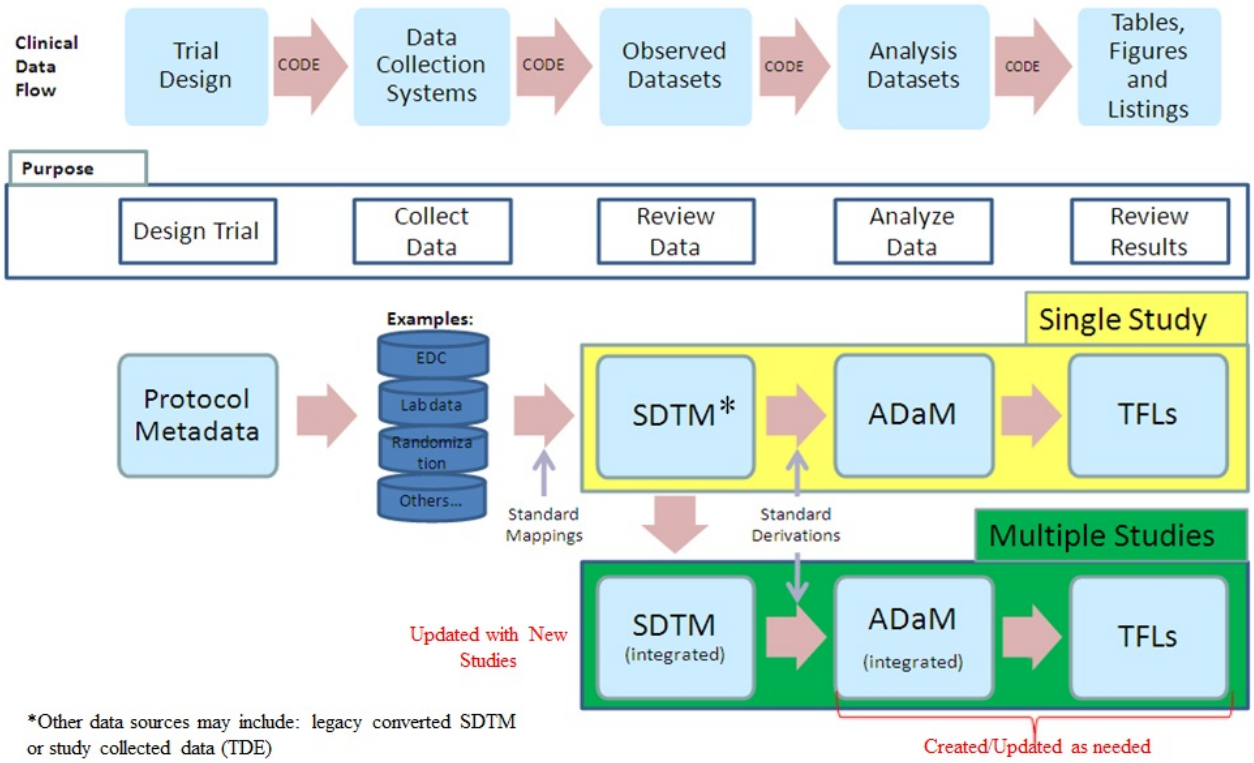

Figure 2. Clinical Data Flow

DATA INTEGRATION PRINCIPLES

We carefully crafted data integration principles to guide our work and decisions in this area. Below are three principles to consider when crafting a data integration strategy:

- Principle 1: Integrated Databases (IDBs) should include all data points (safety, efficacy and study metadata) for patients across all indications for the compound.
- Principle 2: Data integration can begin for a compound when two or more studies have been locked.
- Principle 3: Compound IDBs should remain current throughout the lifecycle of the compound.

THE BENEFITS

Listed below are some benefits found when implementing compound IDBs:

- One standardized data source for all integrated analyses, including regulatory submissions, answering customer questions,data mining and enabling data transparency.
- Facilitates quick and accurate decision making utilizing data from across a compound.
- By allowing data mining of all compound data, this results in lower compound development costs, such as reduction in number of additional studies needed to answer clinical questions.
- Simplifies integrated analysis by utilizing the IDB as your one source of input data.

BEST PRACTICES

- Comply with the Data Integration Principles and build your IDB as early as possible. Do not procrastinate!
- Create and maintain the compound's IDB per the clinical development plan.
- Align IDB maintenance schedule with compound analysis needs (such as submissions, regulatory commitments and internal compound reviews).
- Integrated analyses utilize one source.

Our main goal is to have compound IDBs that align with the data integration principles per their transition plan. By following data integration principles, compounds are positioned to answer internal, regulatory and customer questions in a timely and cost effective manner.

THE LOGISTICS

The strategy in place results in many projects happening concurrently. At any given time, there are 10+ active data integration projects ongoing. Because of this, these projects cannot be treated as one-time programming challenges; it requires a consistent, efficient process to deliver in a timely and cost-effective manner. Whenever the topic of efficient programming is brought up, the conversation typically steers toward macro development. That's true here as well, but there is an added difficulty; our process and tools must be flexible enough to deal with any input data standard and any output data standard.

These projects include study-level data in any of the following standards (note, even within a project there is often a lack of consistency in the input study standards):

- Raw data, which can come from multiple data capture systems and follow a number of different standards
- SDTM data
- Sponsor proprietary analysis data set standards
- A separate, retired sponsor data set standard
- A variety of standards used by various Third Party Organizations who have conducted trials on a sponsor's behalf. This can also include studies conducted by other pharma companies on compounds that a sponsor has acquired.

The standard applied to the output integrated database also varies from project to project. These standards include:

- SDTM
- ADaM
- Sponsor proprietary analysis data set standards
- A separate, retired sponsor data set standard (used only on rare occasion)

Traditional macro development will not result in a sufficient system to support these projects. The system in place must be highly flexible in order to be useful across multiple projects. The macros themselves cannot have assumptions about the input data or output data built into them; those assumptions would not hold true across

projects. The answer is to develop a system where the requirements specifications themselves serve as metadata for a suite of macro tools to automate as much of the data creation as possible. The advantages of such an approach include:

- It provides a comprehensive set of tools that can be used across any data transformation project. Since the macros used in the process have no assumptions about the data, changing standards no longer means changing tools.
- It reduces time spent in programming and validation. Because much of the data creation is automated, the programming can be completed more efficiently. Also, there is less room for human error in the programming, leading to faster validation.
- It provides a mechanism to check the requirements themselves for potential issues. Since the requirements are metadata, they can be checked programmatically for potential errors, thus finding and correcting errors earlier in the process.
- It reduces the impact of requirement changes. Throughout the life of most projects, updates to the requirements are necessary. Since much of the data creation is automated, changes to requirements will have no impact on the automated code. The requirements are updated and programs refreshed; therefore if you change your metadata, your output changes seamlessly.
- The tools provide a standard approach to programming. The ideal scenario is that every programmer finishes the programs they start. However, circumstances sometimes dictate that assignments need to transition to other programmers. This system provides a standard approach to programming, thus reducing the amount of time it takes for a new programmer to update existing programs.
- All of these advantages combine to greatly increase the efficiency of the project.

THE REQUIREMENTS TEMPLATE

The first step in developing such a system is to create a requirements template. The requirements template needs to be programmatically consumable, while also maintaining readability outside of the SAS® environment. We selected an Excel spreadsheet as the method of storing the requirements metadata. The requirements template contains multiple sections to define all of the metadata needed.

While the requirements document contains several tabs to document the project and define all necessary metadata, the three most important types of tabs are highlighted below. Two of these are specialized tabs that define data relationships and could potentially be used across multiple domains/datasets. The other type of tab highlighted is a domain/dataset tab. One such tab is created for each domain/dataset included in the project.

The first of these tabs is the DECODES tab. The DECODES tab defines relationships between related variables. In the below example, the values of FATESTCD (BASE column) define the values of FATEST (VARIABLE column). The BASE_VALUE column documents the values of FATESTCD, and the VARIABLE_VALUE column defines the corresponding value of FATESTCD.

BASE	VARIABLE	BASE_VALUE	VARIABLE_VALUE
FATESTCD	FATEST	AJCETP	Cerebrovascular Event Type
FATESTCD	FATEST	AJDEGIPR	Degree of Impairment
FATESTCD	FATEST	AJDTHSTP	Adjudication Death Subtype
FATESTCD	FATEST	AJDTHTP	Adjudication Death Type
FATESTCD	FATEST	AJECRES1	Adj confirms Myocardial Infarction
FATESTCD	FATEST	AJECRES2	Adj review cnfirm Unstble Angina

Table 1. DECODES Tab

The second specialized tab is the CONVERSIONS tab. This tab defines finite relationships between input values and output values. In the example below, two different relationships are defined. The FORMAT column defines a name for each relationship (allowing that relationship to be referred to elsewhere in the requirements). The INPUT column defines all possible input values, and the OUTPUT column defines the corresponding output value. The following example below demonstrates how the CONVERSIONS tab would be used in a situation where the input values of a

variable (FATEST in the example) are slightly different than what is desired in the output. It also shows an instance of defining a transformation from 0/1 to N/Y.

FORMAT	INPUT	OUTPUT
FATEST	AJCTP	AJCETP
FATEST	AJDEGIP	AJDEGIPR
FATEST	AJDTHS	AJDTHSTP
FATEST	AJDTH	AJDTHTP
FATEST	AJECRS1	AJECRES1
FATEST	AJECRS2	AJECRES2
YN	0	N
YN	1	Y

Table 2. CONVERSIONS Tab

Each domain/dataset in the project will have a tab defining the requirements for its creation. These tabs share the name of the domain/dataset being defined (i.e. the tab for the FA domain would be named FA, etc.). For illustration purposes, the domain/dataset tab will be divided into two sections: Target Definition and Transformation Definition.

The Target Definition portion of the tab defines the attributes of the domain/dataset to be created. These attributes include the variable names, order, type (character or numeric), length, output format (if any), label and sort key. An example is below for an FA domain. For display purposes, only a subset of the typical variables is shown.

Variable	Type	Output Length	Output Format	Label	Sort Key
STUDYID	Char	11		Study Identifier	1
DOMAIN	Char	2		Domain Abbreviation	2
USUBJID	Char	21		Unique Subject Identifier	3
FASEQ	Num	8		Sequence Number	4
FATESTCD	Char	8		Findings About Test Short Name	
FATEST	Char	40		Findings About Test Name	
FAOBJ	Char	37		Object of the Observation	
FABLFL	Char	1		Baseline Flag	
VISITNUM	Num	8		Visit Number	
VISIT	Char	21		Visit Name	

Table 3. Target Definition Section of a Domain/Dataset Tab

The remainder of the tab contains the Transformation Definition portion. For each input study, this portion defines how the data is transformed. Below is an example of the Transformation Definition for the FA domain for theoretical study EFGH. In this example, study EFGH has an existing FA domain which contains most of the values needed in the integration, but there are a few exceptions: it does not contain the DOMAIN or FASEQ variables, the FATESTCD and FATEST values don't match exactly the desired values in the integration and FABLFL has values of 0/1 instead of N/Y.

Variable	Source Study	Source Dataset	Variable	Format	Status Flag	Derivation
STUDYID	EFGH	FA	STUDYID		Y	
DOMAIN	IDB	FA			Z	AE
USUBJID	EFGH	FA	USUBJID		Y	

Variable	Source Study	Source Dataset	Variable	Format	Status Flag	Derivation
FASEQ	IDB	FA	USUBJID VISITNUM FATESTCD FAOBJ		N	Sort by USUBJID VISITNUM FATESTCD FAOBJ. Set FASEQ = 1 for the first record per USUBJID and increment by 1 for each subsequent record per USUBJID
FATESTCD	EFGH	FA	FATESTCD	FATEST	C	
FATEST	IDB	FA			D	
FAOBJ	EFGH	FA	FAOBJ		Y	
FABLFL	EFGH	FA	FABLFL	YN	C	
VISITNUM	EFGH	FA	VISITNUM		Y	
VISIT	EFGH	FA	VISIT		Y	

Table 4. Transformation Definition Section of a Domain/Dataset Tab

The columns in Table 4 have the following meanings:

- Variable (first column) – This is a repeat of the Variable column from the Target Definition section. It is included here for readability. This defines the name of the output variable.
- Source Study – This defines the library that contains the values used as input for the variable. In this example, EFGH refers to the study-level EFGH data library. The physical path for EFGH is defined in the requirements document. IDB is a keyword value used across all projects to denote that the variable input comes from within the integration being created, not from an input study library.
- Source Dataset – This defines the dataset that contains the values used as input for the variable.
- Variable (fourth column) – This defines the variable(s) that contain the values used as input for the variable.
- Format – This identifies the conversions relationship if one applies. See Table 2 for an example.
- Status Flag – This controls the type of processing that occurs for each variable. The meanings of the values used in this example are:
    - Y – The input variable (defined by the Source Study/Source Dataset/Variable columns) contains the exact value needed in the output variable. Creation of this variable will be automated.
    - Z – Each output record contains the same constant value. Creation of this variable is automated.
    - N – This variable is not automated; it is created by custom code.
    - C – The values in the input variable do not exactly match the desired values in the output, but a conversions relationship can be defined to transform the values. That relationship is defined on the CONVERSIONS tab. Creation of this variable is automated.
    - D – This variable's values are defined based on the values of another variable within the integrated dataset. The relationship controlling these values is defined on the DECODES tab (see Table 1). Creation of this variable is automated.
- Derivation – This stores constant values (if Status Flag = Z) or describes derivations for custom code (if Status Flag = N).

MACRO TOOLS

The requirements document stores all of the metadata, thus macro tools are built to to utilize it. These tools are completely metadata-driven. There are no assumptions about the input or output data built into any of the tools, therefore the same set of tools can be used across any project. There are four main macros in the suite of tools.

IDB_METADATA_CHECKS

The first macro is IDB_METADATA_CHECKS. This macro examines the requirements metadata to search for potential issues. This allows requirement issues to be caught and corrected earlier in the process, saving time during the programming and validation. These checks include, but are not limited to:

- All output variables have complete attributes.
- All input datasets/variables exist.
- All CONVERSIONS/DECODES relationships exist on those tabs.
- The Status Flag column contains only valid values.
- All individual variable mappings are complete.

IDB_AUTO_POPULATE

The second macro is IDB_AUTO_POPULATE. Based on the requirements metadata, this macro:

1. Processes input study-level datasets and automatically creates output variables. Any variable with a Source Study value other than “IDB” and a Status Flag value other than “N” isautomatically created by this macro.
2. Runs various checks against the requirements and input data to find potential issues. These checks include, but are not limited to:
    - If utilizing a CONVERSIONS relationship, all input values are accounted for in the conversion mapping.
    - Input values are not truncated.

IDB_HEADER_DECODE

The third macro is IDB_HEADER_DECODE. Based on the requirements metadata, this macro:

1. Automatically creates variables that originate within the integration (as opposed to an input study location). This includes populated variables pulled from other domains/datasets within the integration (such as reading VISIT from the SV domain) or variables based on the value of other variables within the domain/dataset (such as basing the value of FATEST on the value of FATESTCD).
2. Creates variables that have constant or missing values for every output record. Any variable with a Source Study value of “IDB” and a Status Flag value other than “N” will automatically be created by this macro.
3. Runs various checks against the requirements and input data to find potential issues. These checks include, but are not limited to:
    - If utilizing a DECODES relationship, all input values are accounted for in the decodes mapping.
    - Ensures all other integrated domains/datasets identified in the requirements exist in the input sources.

IDB_APPLY_ATTRIBUTES

The final macro is IDB_APPLY_ATTRIBUTES. Once all variables are created, the macro:

1. Applies the attributes from the requirements metadata to the output data. This includes the lengths, labels, formats and order of the variables.
2. Drops any intermediate variables in the dataset that are not included in the requirements.
3. Runs various checks against the requirements to find potential issues. These checks include, but are not limited to:
    - All variables defined in the requirements have been created.
    - No values will be truncated when applying the length from the requirements.

USING THE TOOLS

The requirements template, which stores the metadata, and the tools built to process the metadata are two out of the three components within the IDB creation process. The final component develops programs that utilize the tools to automatically create most or all of the variables and if needed, custom code to generate the rest. These programs can be called by a single driver program and contains the following sections:

1. Setup/pre-processing: Handles the setup and pre-processing for the program, including:
    - Libname statements
    - Macro calls to import the requirements spreadsheet into SAS
    - Commonly needed code, such as reading in the MedDRA dictionary.
2. Transformation: Creates the output datasets by including a section of code for each study's contributing data to that dataset/domain, using the following steps:
    - Call IDB_METADATA_CHECKS
    - Call IDB_AUTO_POPULATE
    - Opportunity for programmers to insert custom code needed to generate variables that can not be automated (if necessary)
    - Call IDB_HEADER_DECODE
    - Another opportunity to insert custom code (if necessary)
    - Call IDB_APPLY_ATTRIBUTES
3. Post-processing: Concatenate, sort and output the data.

Below is an example of a program to create the FA domain for a theoretical project. In this project, there are two studies being integrated: ABCD and EFGH. ABCD has a very straightforward requirements mapping; all variables are automatically populated without custom code. EFGH uses the requirements mapping illustrated in this paper.

```
*******    insert setup/pre-processing steps                  *****;
*******     processing for study ABCD                         *****;
*******    no custom code required, only standard macro calls  ****;
%let study=abcd;

%idb_metadata_checks;
%idb_auto_populate;
%idb_header_decode;
%idb_apply_attributes;

**********  processing for study EFGH                         *****;
**********  custom code necessary for FASEQ variable          *****;

%idb_metadata_checks;
%idb_auto_populate;

*custom code to derive faseq;
proc sort data=fa_efgh_fa; *dataset output by idb_auto_populate;
   by usubjid visitnum fatestcd faobj;
run;

data fa_1;
    set fa_efgh_fa;
    by usubjid;
    retain faseq;
    if first.usubjid then faseq = 1;
    else faseq = faseq + 1;
run;

%idb_header_decode;
%idb_auto_populate;

************** final section to concatenate/sort/outut       *****;
data fa_final;
  set abde_fa_final efgh_fa_final; *datasets output by idb_apply_attributes;
run;

proc sort data=fa_final out=out.fa;
  by &fa_sort_key; *This is a macro variable storing the sort order;
run;
```

CHALLENGES IN DEVELOPING SUCH A SYSTEM

A metadata-driven process delivers significant rewards in project efficiency. However building it presents significant challenges. Among the challenges faced are:

- Documentation/Training – If the programmers working on the projects are not accustomed to a metadata-driven process, a learning curve exists that must be managed. Detailed documentation and training is a must for this type of process.
- Developing the Requirements Template – If the macro tools are completely metadata-driven and make no assumptions about the data, the requirements template must have ways to store every piece of metadata required. This goes beyond the examples illustrated in this paper. Key variables for combining datasets, source data locations and methods for handling multiple input datasets for a study must also be considered.
- Handling Conflicts Between the Input and Output Data – Because the process is designed to handle any input format and any output format, it accomodates situations where the input data has a variable name that matches a variable name in the output. For example, VISIT is a numeric variable on the input dataset but a character variable on the output dataset.
- Macro Validation – Since the process is designed to handle any input format and any output format, the macro requires many test cases to validate.

OBSERVED BENEFITS - METRICS

Using our time-reporting system, we compared hours spent on projects prior to and after implementing this metadata-driven process. Below are the observed impacts we've seen to project efficiency:

- On average, approximately 70% of variable transformation is automated using this process.
- Requirements Definition – 10% increase in hours. Because the requirements now need to be more specific, there has been a slight increase in the time it takes to define them. However, that small investment pays large dividends in the rest of the process.
- Primary Programming – 30% decrease in hours. Primary programming includes the time spent in creating the first draft of the datasets plus the time spent updating due to issues found in validation and/or change in requirements. The time savings stem from the increased automation and the lessened impact of change in requirements.
- Validation – 50% decrease in hours. Validation includes the time spent programming to create validation datasets (full replication of all data), time spent updating code due to issues found in validation and/or change in requirements, as well as time spent comparing and analyzing data to identify validation issues. Validation experiences the same type of reduction in hours as primary programming, and because there is less room for human error, it also takes less time to identify and analyze issues found.
- Overall – 35% decrease in hours. All of the benefits of this process combine to give a significant decrease in time spent and a sizable increase in efficiency.

CONCLUSION

The metrics presented provide evidence of the benefits of a strategy to integrate data early and often during a compound's life cycle. In addition, the metrics show that implementing this strategy via a metadata-based system provides a significant decrease in time spent and a sizable increase in efficiency. We hope our experience inspires others to review their data format, integration and implementation strategies to improve their processes of answering internal and external questions.

REFERENCES

Reynolds, Natalie. 2013. "Integrating and Converting Data Across Studies – Clinical Trial Integrated Databases (IDBs)." Proceedings of the 2013 FDA/PhUSE Annual Computational Science Symposium. Available at http://www.phusewiki.org/docs/Posters2013CSS/PP25%20.pdf.

ACKNOWLEDGMENTS

The authors would like to acknowledge their following colleagues:

Carol Frazee and Lisa Young – For their leadership and support throughout the life of the Eli Lilly and Company/Inventiv Health Clinical Partnership.

Rachael Birge and Jane Lozano – For their support, contributions and being champions of the IDB strategy and process at Eli Lilly and Company.

Steven Wang, Chris Pratt, Lana Astashinsky, Mitesh Patel – For their contributions in ideas and macro development in the IDB process.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Natalie Reynolds
Eli Lilly and Company
Lilly Corporate Center
Indianapolis, IN 46285
317-440-4781
nreynolds@lilly.com
https://www.linkedin.com/pub/natalie-reynolds/63/b13/7a2

Keith Hibbetts
Inventiv Health Clinical
225 S East St. Suite 200
Indianapolis, IN 46202
317-519-4372
keith.hibbetts@inventivhealth.com
www.linkedin.com/in/keithhibbetts