PharmaSUG 2019 - Paper SI-243

Dataset Specifications: Recipes for Efficiency and Quality

Dave Scocca, Rho, Inc.

ABSTRACT

Specifications are the backbone of dataset programming; quality specifications are the recipes with which we produce quality datasets. While we frequently talk about programming strategies and techniques, we spend much less time focusing on dataset specifications. Having high-quality specifications can greatly improve programmer efficiency and speed up validation. Developing specifications and programs hand-in-hand can reduce overall review time and avoid a great deal of unnecessary rework.

Questions addressed in this presentation include: How do dataset specifications fit into the clinical trial process? What makes a high-quality dataset specification? How can we make the overall process of dataset production—from specifications through programs—as efficient as possible? Are there ways to standardize the specification development process?

INTRODUCTION

When you need to create datasets, you write specifications to document what you expect to see in the datasets and how you expect them to be created. Well-written specifications make for faster and more effective dataset programming and validation, and can provide more consistency across datasets.

This paper will focus on SDTM and ADaM data packages; these standards provide dataset structures which serve as an excellent foundation for your dataset specifications.

Specifications for data are like recipes in your kitchen: a set of instructions telling you how to use the ingredients to produce the final result you want.

ABOUT SPECIFICATIONS

THE AUDIENCE

The programmers who create and validate the datasets are the primary users of your specifications, but there are other audiences. Sponsors review specifications to assess the work of contract research organizations (CROs); specifications can be the basis for the documentation in your submission package, particularly for derivations; and people working on studies in the future can use past specifications as a starting point.

WHY SPECIFICATIONS?

Why do we have specifications in the first place? Why not simply write dataset programs and use them as the documentation of what was done? Having specifications exist separately from programs has a number of advantages. First, it allows validation: by independently writing two dataset programs to implement a single specification, you reduce the risk of having programming errors produce incorrect data. Second, a single specification document is easier to read and review than a series of programs. Having independent documentation of what each program is supposed to be doing improves traceability and makes it more straightforward to develop data package documents such as the define file and reviewer's guide.

SPECIFICATIONS AND METADATA

The most useful dataset specifications are in electronic form with machine-readable metadata. When you can read variable and dataset attributes from the specifications, you can develop standard, validated code to apply these attributes and free programmers to focus on the more challenging parts of dataset creation.

MAPPING DATASETS: MENUS AND INGREDIENTS

Before you can write a recipe you need to know what dish you're writing the recipe for. Similarly, to write dataset specifications you need to know what datasets you're documenting. The set of datasets you want to produce is like a menu, and the data sources you're using are your ingredients. Getting a picture of the overall menu is going to be the first step in writing your recipes.

At this point, the culinary comparison becomes a little different depending on whether your goal is tabulation (SDTM) or analysis (ADaM).

SDTM: INGREDIENTS

For an SDTM package, your goal is to accurately and completely tabulate the data that has been collected from the CRF, central lab vendors, and any other data providers involved in the study, and to thereby provide the basis for creating ADaM datasets. In this situation your menu is expected to incorporate all the ingredients you're given, while meeting the dietary needs of your customers.

In planning out an SDTM package, it is useful to develop a pair of cross-referenced lists: a list of source data files, annotated with the domain or domains where the data will be stored in SDTM, and a list of SDTM datasets, annotated to identify the source data that will be incorporated into each one. These initial mappings will be a working document that will guide you in developing more detailed specifications.

ADAM: CUSTOMERS

For an ADaM package, your goal is to provide datasets that will support the displays that will be used for analysis. This means your menu must be designed to meet the tastes and desires of your customers. You are still limited by the ingredients available to you in your SDTM or other source datasets, but your success will be determined by whether your ADaM datasets support all the tables and figures that will be expected for your study.

In planning out an ADaM package, it is helpful to start with the statistical analysis plan to identify the displays that will be your end goal. Once these have been identified, you can structure the ADaM datasets to best support the production of these displays.

GROCERY SHOPPING: COORDINATION

In neither of the examples above is there any discussion of how you get to choose your groceries: SDTM is seeking to use up the contents of a pre-stocked pantry, and ADaM is limited to the available ingredients. So how do you actually control what you have to work with?

In order to have any control over your source data, your SDTM and ADaM teams need to be involved in the CRF design process. This may not be in your control, particularly at a CRO where the sponsor may bring you into the project after the CRF has been created and the study begun. But when your company is involved in the entire process, you will find it easier to develop your SDTM and ADaM specifications when you participate in CRF development.

INPUT NOTES: THE INGREDIENTS

Most recipes begin with a list of ingredients. Similarly, at the dataset level a specification begins by identifying the data sources that will be used to create the dataset, which we call the "input notes". Just as recipes include preparation instructions ("one medium onion, diced") so too should your data input notes provide the necessary information about how to prepare the data. Too frequently, specifications are presented as a simple spreadsheet of variable mappings; documentation of data sources is an essential but often overlooked part of a specification.

Data commonly need to be prepared by simple filtering. If your adverse events dataset from the CRF contains rows for subjects who did not have an adverse event, your input notes to create SDTM.AE should indicate that rows where adverse events did not occur should be dropped.

Data preparation can also include transposition; when creating the SDTM VS or EG domain, you will often need to transpose a vital signs or ECG clinical dataset where multiple results are stored

horizontally. Your input notes should also include information on such transposition: what values from each row should be maintained, and which columns should be transformed into rows for the final dataset.

DOCUMENTING MERGES

Frequently you will need to merge input data to create your dataset. Your specification should indicate what data sources to merge and the fields to use for merging. If you expect any mismatches, indicate how to handle records that are not in all datasets. In some instances, you can indicate that data anomalies should be reported to the data provider. For example, if you're merging single records from a CRF page header data with multiple detail records from the rest of the page, you would want to notify the data provider if you find detail records that don't match any header records.

When you specify merge variables, be complete; your specification should identify all variables necessary to make the merge. If some variables need transformation to be merged—if a record identifier is stored as a numeric value in one data source and a character value in another, or if subject identifiers are formatted differently in different data sources—the specification should indicate this.

STACKING DATA FROM MULTIPLE SOURCES

Frequently, different rows of the final dataset will come from different data sources. In SDTM, the LB dataset may combine electronically provided central lab data, local lab data entered on the CRF, and results from a CRF page for pregnancy testing. In this instance, you can say you want to "stack" the data; you are combining the rows from different sources but not using multiple sources to populate a single row. Your input notes should indicate which data sources will be stacked to produce the target dataset.

TEMPORARY DATASETS

In some instances, your specification will call for making a large or complicated combination of datasets; an oncology study might require combining tumor response assessments with multiple adjudication data sources to identify the assessors, the individual assessments, and the results of adjudication. When a complex merge like this is required, you can use the input notes to describe how to create a temporary dataset and then refer to that temporary dataset when describing how to populate your individual variables.

SPECIAL-USE DATASETS

While your input notes should identify all input data sources, not all data sources need to be listed in the same way. For example, for an SDTM dataset you will often have to derive a study day variable by using DM.RFSTDTC, derive EPOCH from treatment dates in DM, or standardize a visit by using values from TV. You can use a standard phrase in your input notes to indicate that one of these is the case: "Use DM to derive study day and EPOCH", or "Use TV to derive VISIT and VISITDY". For ADaM, all datasets will pull subject-level column values from ADSL. Documenting these standard uses apart from the main list of source data can make your specification clearer.

SEQUENCING AND DEPENDENCIES

One use of complete input notes is to help document the dependencies between datasets in your package and allow you to find the appropriate sequence in which to run programs. In addition to the obvious cases—DM before any dataset which derives study day; TV before any dataset which needs to standardize VISIT; ADSL before other analysis datasets—you may identify cases where the primary inputs for a dataset are other datasets in the same package. In SDTM, you often want to derive the subject elements domain (SE) from other SDTM datasets rather than the raw data; clear input notes will document these dependencies and help you create your datasets in the appropriate order.

THE INSTRUCTIONS

Once you have the ingredients, you need the actual recipe instructions to combine them into your final product. Lists of variable mappings are usually the longest portion of a written specification, although many of those mappings will be straightforward.

RECORD CREATION

For some datasets, there is a simple one-to-one mapping from the source data: one DM or ADSL record per subject, or one CM record per medication recorded. In other cases, your variable-level mappings should incorporate instructions about the circumstances under which to create records in the output dataset. For an ADaM dataset using the basic data structure (BDS), this may be a list of PARAM and PARAMCD values for which rows should be created. For SDTM, this may be a list of TEST and TESTCD values for a findings dataset, or a list of conditions under which records should be created for other datasets.

For example, if a vital signs CRF dataset records test results horizontally, and the height field is missing at visits where height was not collected, the specification can document that no VSTESTCD=HEIGHT record should be created for those visits.

Clear documentation of when to create records is particularly important in SDTM when working with dispositions and subject elements. You will usually want to specify conditions for creating protocol milestones, e.g. “for every subject with a nonmissing randomization date in the IVRS data, create a record in DS with DSTERM=RANDOMIZED”. For SE, you will want to create records for an element based on the data showing that the subject entered that element, e.g. “for every subject in DM with DM.RFXSTDTC populated and DM.ACTARMCD=STUDYDRG, create a record in SE with ELEMENT=STUDY DRUG TREATMENT”.

MAPPING AND DERIVATIONS

When most people think of a dataset specification, what first comes to mind is the variable-by-variable list of mappings or derivations. The purpose of these mappings is straightforward: to convey how the values in the final dataset should be obtained. When writing mapping instructions, consider both your audience and the need to avoid errors in the final dataset.

In specifying a derivation or calculation, focus on the desired result rather than on providing a fragment of code that might implement it. If the specification contains a code fragment, and that fragment is used directly for both production and validation programming, then any problems in the code will not be detected by the validation process. In addition, anyone who may review the specification will not need to be as familiar with the programming language you are using.

For a USUBJID field, you could write “concatenate STUDYID and the subject identifier, separated by a hyphen” instead of “=STUDYID || '-' || SUBJID”. Or for a text field that needs to have special characters removed or remapped, identify the characters to be fixed rather than writing out a TRANSLATE() function as part of the specifications.

Some mappings are so standardized that relatively little detail is needed. For SDTM, all study day variables will be derived using the formula from the SDTM Implementation Guide; for VSDY, a specification of “study day derived from VSDTC and DM.RFSTDTC” is preferable to putting the SDTMIG formula in your specification.

COMPLETENESS

Your mapping specifications for a field should be complete. If you have a list of raw data values to be remapped, or a list of situations and instructions for what to do in each of them, you imply that the list is comprehensive. You do not want a case where the specifications do not say what should happen in a situation that actually occurs in the data. When you assume that a situation will not occur, you should make that assumption explicit in the specification; ideally, programs will include code that checks the assumptions and generates warnings if there are cases where the specification does not indicate what should happen to the data.

EXTERNAL REFERENCES

In some instances, you may need to specify a complicated or extensive set of remappings based on values from your source data. If a central lab provides data with tests and codes that do not match the SDTM controlled terminology, then the specification for the LB domain will need to map the nonstandard

codes to the standard ones. To implement this mapping, you may find it easiest to create a separate spreadsheet or data table associating the lab-provided tests and codes with the standard ones, and to write your specification referring to that spreadsheet. That makes the specification easier to read and is often more straightforward to update when new data requires additions or changes to the existing mapping. However, when the same mapping spreadsheet is to be used by production and validation programmers, you will need to review it carefully and consider if it can be independently validated.

Other parts of your specification may also be easier to maintain and use as separate documents. A SDTM package often uses the same derivation of EPOCH across domains, and that definition is sometimes modified or updated as the study progresses. Maintaining a separate document describing the derivation of EPOCH may be easier than keeping each domain's EPOCH specification consistent with each other and with the overall approach for the study. (In this instance, you will usually want to have common programming code that implement the derivation; you may implement separate instances of the algorithm for production programs and validation programs, but using the same code across domains guarantees consistency.)

STANDARD DERIVATIONS

When you work on multiple studies, you will see patterns in what needs to be done with the data. In SDTM a CRF field that allows free text entry will often lead to data values containing control characters or extra spaces. In this instance, you can standardize your derivation (and, ideally, the code that implements it) so that your specification can say something like "clean the free text value" rather than needing to have the specifics every time you encounter one of these fields. Implementing these standards across your company can save both specification time and programming time on future studies.

SPECIFICATIONS AND PROGRAMMING

The field of software development is full of specifications. Over the years, software practices have evolved from a "waterfall" methodology, where specifications are written before programming begins, to more agile methodologies in which specification development and programming go hand-in-hand.

SPECIFICATION REVIEW

The most common dataset workflow between sponsors and CROs looks a lot like a waterfall methodology. Sponsors ask for detailed specifications to be delivered, reviewed, and approved before programmers begin writing the programs that implement the specifications. Specifications are reviewed before being programmed in order to avoid the rework of updating programs in response to specification changes made during the review process. Independent validation programming makes sure that the reviewed specifications are being followed.

This approach has an appealing simplicity: of course it makes sense to make a plan before you start working. The original waterfall model came about in part because software development was influenced by engineering, and no one would pour the foundation for a bridge footer without a complete plan for the final structure. This analogy hides a fundamental difference between computer code and physical structural components: existing computer code is easy to modify. A computer program can be iteratively developed in a way that a house or a rail line cannot. This realization has driven the adoption of agile software development methodologies.

For dataset creation, the problem with reviewing specifications before starting to program is that the single most reliable form of quality control for a specification is writing and executing a program that implements the specification. As a sponsor, while it makes sense for you to review and approve the specifications, that review is going to be more efficient and effective if you do it after the specifications have been updated to resolve and correct issues that are discovered during programming.

The rework of updating programs to implement specification changes arising from the review process pales in comparison to the rework of updating already-approved specifications to resolve issues that were not apparent before programming.

To return to the culinary analogy: when writing a recipe, even an experienced chef working with familiar ingredients will insist on going into a kitchen and preparing the dish one or more times before declaring

the recipe finished. And in the case of programming, you don't even have to worry about running out of your ingredients if you iterate more times than planned.

The question then becomes as a sponsor, how can you guide the data package development process? One answer is to focus on the menu rather than the recipes: relatively early in the process, you should review the list of datasets to be created and the high-level mapping of data sources. At this stage, the review helps make sure the sponsor and the CRO are on the same page; once there is agreement on the high-level mapping, later revisions to individual datasets should be fairly straightforward to implement even if the programming has already been done.

PROGRAMMING AS QUALITY CONTROL

When programming from a specification, your program should both implement the specification and, where possible, try to check the assumptions and limits of the specification to verify that nothing has been overlooked. For example, if the specification calls for merging two datasets, your program should check for any mismatches or many-to-many matches and verify that the specification correctly describes how to handle them. As described above, when the specification enumerates a particular list of situations or of values to be remapped, your program should capture any instances in which an unexpected situation or value occurs.

When programming a data package, you are also performing QC on the source data you are using. SDTM programming can detect dates that are invalid or out of sequence, responses that are inconsistent with one another, and cases where expected fields have not been populated. When you are working with an ongoing study, you can communicate these issues back to the data manager to see if the issues can be resolved before the study is complete and the database is locked. The quality of your data package is limited by the quality of your input data, and to the extent that you can take steps to improve the quality of the input data you can thereby improve the quality of your own product.

EXPECTATIONS

Recipe writers make assumptions about the knowledge and capabilities of the cooks who will be using the recipes. Historic recipes sometimes seem terse and confusing because they assume knowledge that is no longer common. When writing a dataset specification, what should you expect from the programmers who will be using it?

At a minimum, you would hope your programmers are familiar enough with the programming language to use it effectively. It would also be reasonable to expect your programmers to have some familiarity with the appropriate CDISC implementation guide, for SDTM or ADaM, at least to the extent of understanding that visits are stored in TV and SV, and that the SDTMIG specifies a single formula to derive a study day variable which should be applicable everywhere.

The programmer’s work is as much to validate the specification as to implement it. A programmer should be expected to think logically about the specifications and to identify and communicate any inconsistencies or assumptions that aren’t met in the actual data.

MACHINE-READABLE SPECIFICATIONS

What if you didn’t need to actually write programs, only specifications? That is, what if it were possible to write specifications that could be automatically interpreted and used to generate the specified datasets? Would this be a great step forward for efficiency and productivity? This sounds like a great idea, but runs into logical problems.

Fundamentally, a computer program is a means of telling a computer what to do. In a system where your specifications can produce the final dataset without additional programming, all that would mean is that your specifications would themselves be a computer program. Any savings from not having to write a separate dataset program would be offset by the increased time and effort required to write the specification in a syntax that would be properly interpreted by the computer—assuming that the program parsing the specifications was sufficiently complete that all necessary manipulations and derivations were part of its repertoire in the first place.

In almost any programming language, you can implement another programming language that is less powerful and has more bugs. If you are writing a specification that will be transformed into executable code, you are basically choosing to write in that second language

Finally, there is the problem of validation: if a specification will be automatically executed, what is the equivalent of having two independent programs that implement the same specification? Making two independent specifications would be less efficient than writing two programs from a single non-computer-readable specification. Having a programmer duplicate the work of the parser by independently turning the computer-readable specification into a separate executable validation program would likely take more time and be more subject to error than the usual process of programming from specifications written for human programmers.

CONCLUSION

High-quality dataset specifications allow you to deliver quality data packages more efficiently. If you plan out your “menu” of data sources and datasets, thoroughly specify how to prepare your “ingredients”, and spend time in the “kitchen” actually testing your “recipes”, you can efficiently serve up a data package that will delight and satisfy your customer.

CONTACT INFORMATION

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

Dave Scocca
Rho, Inc.
dave_scocca@rhoworld.com