PhUSE EU Connect 2018

Paper AR01

Developing an end to end Standard & Automated Risk Based Monitoring Process.

Giuseppe Di Monaco, UCB BioSciences GmbH, Monheim, Germany

Tim Williams, UCB BioSciences, Raleigh, USA

ABSTRACT

In the last few years the Pharmaceutical Industry has invested a considerable amount of resources in Risk Based Monitoring (RBM) activities. Regulatory agencies such as FDA and EMA have provided guidance documents in support of risk based approaches in the conduct of clinical trials. RBM implementation has taken a big step forward, thanks to this investment. Now companies face challenges related to data standards, process automation, and proper knowledge management.

This paper has three main objectives. It demonstrates why a complete, automated, end to end RBM process is important in terms of return on investment and quality; how the automated process can be achieved using standardization; what challenges relate to data standards, process automation, and knowledge management.

INTRODUCTION

Clinical research organizations are adopting RBM processes to reduce costs and to ensure quality in clinical trials by identifying, assessing, monitoring and mitigating the risks which could affect quality or safety in a study. A metric used to assess risk associated with an activity is called a "key risk indicator" (KRI). It provides an early signal of increasing risk exposure in various areas. At UCB, the following three major areas were defined to asses these KRIs: Data Quality & Validity, Patient Safety, Site Compliance & Performance. The KRI monitoring plan includes only those KRIs necessary for the specific trial since every clinical trial focuses on different objectives or aims. We have a standard set of the KRI and in addition we may discuss study specific KRI for a study.

BACKGROUND

Given the broadness and complexity of the three major areas in which UCB KRIs have been defined, the RBM analytics team focused initially on fulfilling KRI requirements when creating programs, leaving the tasks of standardization and automation of KRI programming for later. This resulted in monolithic SAS programs that are difficult to understand, change and maintain. In addition, the process for creating KRI datasets requires many manual steps. The approach to solving these problems was to create new standard macros and run them parallel to the legacy programs until the new macros were stable and validated by the old programs.

OBJECTIVES

PRIMARY OBJECTIVE

The primary objective of this paper is to show how an end to end Standard & Automated (S&A) RBM Data Surveillance process can be created where operator interventions are eliminated or reduced to almost zero, so that product quality, efficiency and knowledge management can be increased while reducing cost.

SECONDARY OBJECTIVES

The secondary objectives are to show the **strategy** used to achieve the primary objective and the actions taken to transform the inefficiencies and threats into opportunities for improving not only the RBM process but also other overlapping tasks performed by various stakeholders. The approach was to design the structure of the new process as a combination of zooming views, with a **ZOOM-OUT** to the big picture where RBM Data Surveillance process is emphasized and a **ZOOM-IN** in to the functionalities of the single macros and programs where details matter the most.

For the **ZOOM-OUT,** the design was to slice up the end to end process into ten major steps. For brevity, this paper will focus on the **ZOOM-OUT** high-level step number **8** (**Figure 1**) which is strictly relevant to RBM.

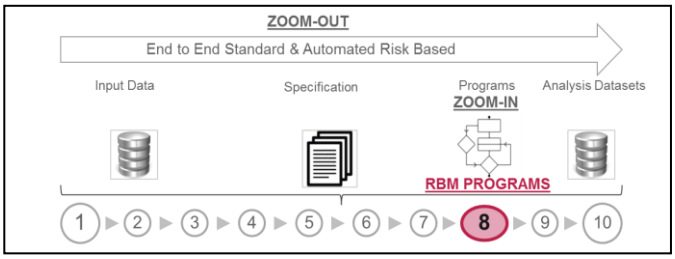

Figure 1 End to End S&A Process

The functionalities of each individual macro and program were revised and redesigned for the **ZOOM-IN**. Within step **8**, one KRI provides an example that describes the low-level aspect and how it fits into the high-level (**ZOOM-OUT**) design of the entire end to end process, which involves other dependent processes outside the RBM area. Two SAS macros (ut_tstcd_mahap_dat_u.sas and v_dva_mahap_dat_u.sas) are used in one example of the **ZOOM-IN,** low-level macro functionalities of the KRI "Low Variability", which detects fraud and misconduct. The other nine steps are briefly described in the section "Design & Implementation of the New Process"

APPROACH

The "Evolutionary" software engineering approach was used to redesign and build the process. In this approach, components of the original process are progressively replaced by newly re-engineered components. These parts of the new process were chosen based on their functionality and not on the structure of the existing system. Thus, re-engineering efforts focus on identifying objects regardless of where tasks reside in the old process.

METHODOLOGY

The methodology includes four major steps:

A. Define S&A process goals
B. Analysis of the old process
C. Design & Implementation of the new process
D. Code Management with a knowledge graph

A. DEFINE S&A PROCESS GOALS

Goals were defined to:

- Automate the process of creating KRI analysis datasets.
- Streamline communication across teams by creating an automated email notification system.
  These notifications alert the advanced analytics programmers of any defect, data managers for any data issue, and Clinical Study Reporting Managers (CSRMs) for the availably of the analysis datasets. In this way, all data and RBM stakeholders automatically receive relevant notifications.
- Identify and correct data issues before KRI creation.
- Implement error-handling techniques and anticipate, detect and resolve programming or communication errors.
- Enable handling of any number of studies (increase scalability).
- Create an efficient and reliable knowledge management and impact analysis process.

B. ANALYSIS OF THE OLD PROCESS

PROCESS

In the old process, programs are monolithic and lack modular design. What is more, program complexity increases during maintenance when parts are added or removed for new functional specifications or bug fixes. Obsolete and unnecessary code is often not removed from the programs. Because of the scope and lifespan of the programs, code control can become problematic.

NAMING CONVENTIONS

Names are necessary in everything we do, whether referring to ourselves or everyday objects. The more descriptive the name, the more it intrinsically conveys meaning. Naming and organizing folders, files, software code, and associated documents when building a software program creates consistency and eliminates ambiguity, the enemy of efficiency and quality.

The old RBM Data Surveillance process has issues related to naming conventions. Some of the problems could be fixed while others could not because those changes would impact other processes. Two examples in which corrections were possible are presented. In the first example, some programs and output datasets were named <name>_vs, likely because the draft versions of the files were initially written and tested using a Vital Signs (VS) dataset and an inconsistency was introduced when additional datasets (FA, QS, *et cetera*) were added later. In the second example, SAS program names did not facilitate easy categorization. For example, utility programs were not easily distinguished from the programs that create analysis datasets.

COHESION

Cohesion is one of the most important concepts in software design (1). It is the core of most good design principles because it guides separation of distinct concepts and functionality. High cohesion means that the maintainability of the software increases because a program is focused on a primary goal or function. In contrast, low cohesion is associated with traits such as being difficult to maintain, test, reuse, or even to understand. Programs with high cohesion are more robust, reliable, reusable, and understandable. Programs from the old KRI programming effort did not have an acceptable level of cohesion. For example, some programs produced many different output datasets and many contained multiple SAS macros with diverse functions. In addition, study-specific information and programming algorithms were repeated across multiple files, making the code hard to understand and maintain.

NON-FUNCTIONAL REQUIREMENTS

As previously mentioned, the KRI programs were initially created with the focus on their Functional Requirements (FRs) and not *how* they were created or maintained. However, non-Functional Requirements (NFRs) are as important as FRs because they define a program's quality characteristics such as reliability, robustness, performance, maintainability, scalability, usability, modularity, *et cetera*. Hence, NFRs are just as critical as FRs. Failing to meet any one of these quality criteria could result in a process that fails to satisfy business requirements.

Assessing the lack of non-functional requirements is difficult and out of scope, so the approach was to model prior failure data to predict future behavior for the management of KRI programs. The analysis of the NFRs on these programs showed weakness in areas including:

- Unnecessary complexity. High complexity
  - is the major factor contributing to software reliability problems.
  - opposes **maintainability**. Maintainability is defined as the degree to which code can be understood, repaired, or enhanced
- The code often lacked effective defensive programming techniques. "Quick-fix" solutions focused on defect reduction, adversely affecting the program's **robustness**.
- Workflows may not be **scalable** for a steep increase in the number of studies.
- KRI programs were mostly **monolithic** programs, composed all in one piece. Development of monolithic programs may be faster at the start, but this advantage is quickly lost due to less control over the workings and higher long-term maintenance costs due to an inevitable need for workarounds. The result is that when a new study is set up, many programs need hardcoding modification to accommodate the specifics of the new study.
- Inexperienced users encounter problems with the intricate process of linking programs to inputs, outputs, and related information. As a result, usability, **learnability**, **understandability** and **communicability** are negatively affected.

SCOPE OF KRI PROGRAMS AND PROCESS VERSUS STUDY-SPECIFIC PROGRAMS

The scope of the KRI programs is very different from that of study or project-specific programs. KRI programs must be **reliable, robust, scalable, maintainable, usable**, **learnable**, **understandable** *et cetera*. In an environment of multiple studies, KRI programs are too often based on code and approaches encountered in a single study or project. The design of KRI programs is necessarily unlike programs developed for clinical study work. Clinical study programs are designed around an individual study/project so their lifespan is limited to the length of that particular study. Also, the number of programmers working on a study/project over its lifespan is lower than the number working on KRI programs where the lifespan is much longer and there are multiple studies. These fundamental differences make the specification, development, validation, and release of KRI programs much more difficult than study/project-specific programs. Elevating KRI programs to become part of a Corporate Macro Program Library (CMPL) for KRI is preferred. When KRI macros are designed and developed within the CMPL macro structure and rules, their deployment on a new KRI project requires much less effort. The validation phase of these macros would be faster, and controlled using UCB's automated environment for regression testing. (2)

KNOWLEDGE MANAGEMENT

In general, the old process showed a lack of capturing, distributing, and effectively reusing knowledge between stakeholders such as programmers and CSRMs. This highlight both the need to capture knowledge and to communicate that knowledge effectively, going beyond the realm of good programming practice.

SOFTWARE LIFECYCLE

All programs should have a defined software lifecycle process which controls the history, defects, release and associated documentation for each version of each program, following the path of:

Requirements/Analysis > Design > Implementation > Test/Validation > Deployment > Maintenance (2)

C. DESIGN & IMPLEMENTATION OF THE NEW PROCESS

PROCESS

The design and implementation of the new process faces the same problems mentioned in **section B**. The new **S&A** process was designed around the concepts of **structure** and **simplicity.**

Structure provides the project's foundation, ensuring that every subsequent component is secure and stable.

Simplicity facilitates understanding of the structure and provides everyone with the knowledge needed to add or build modules on top of existing process.

Not only was the RBM Data Surveillance process re-engineered, but improvements were also made in adjacent areas upon which RBM depends. This included standardizing the location of inbound data, automating the email notifications, and planning a new project to improve the data review/oversight model.

Primary and secondary objectives are illustrated in **Figure 2**. Daily inbound data transfers from our partners are received and automatically decompressed into standard study-specific folders. The KRI programs inspect these

folders for new data and there is no need to update programs at each delivery. Agreeing a standard process and location for inbound data may seem trivial but involved several stakeholders including external partners, IT department, Data Managers, and RBM programmers.

Data files are unzipped and the 10 steps of the new automated RBM Data Surveillance process starts (**Figure 2**).

STEP 1: Autoconfiguration. Create study folders as needed.
STEP 2: Check data against company standards.
STEP 3: Copy data and create folders for KRI analysis. Create SAS program for library assignment.
STEP 4: Identify data issues and email managers and stakeholders as necessary.
STEP 5: Ensure a clean run environment (no hold-over processes from previous runs).
STEP 6: Automatically adapt files to standard conventions.
STEP 7: Load study settings.
STEP 8: Run all KRI analytics programs. Validate and compare runs.
STEP 10: Perform quality, sanity and consistency checks on all KRI analytics datasets.

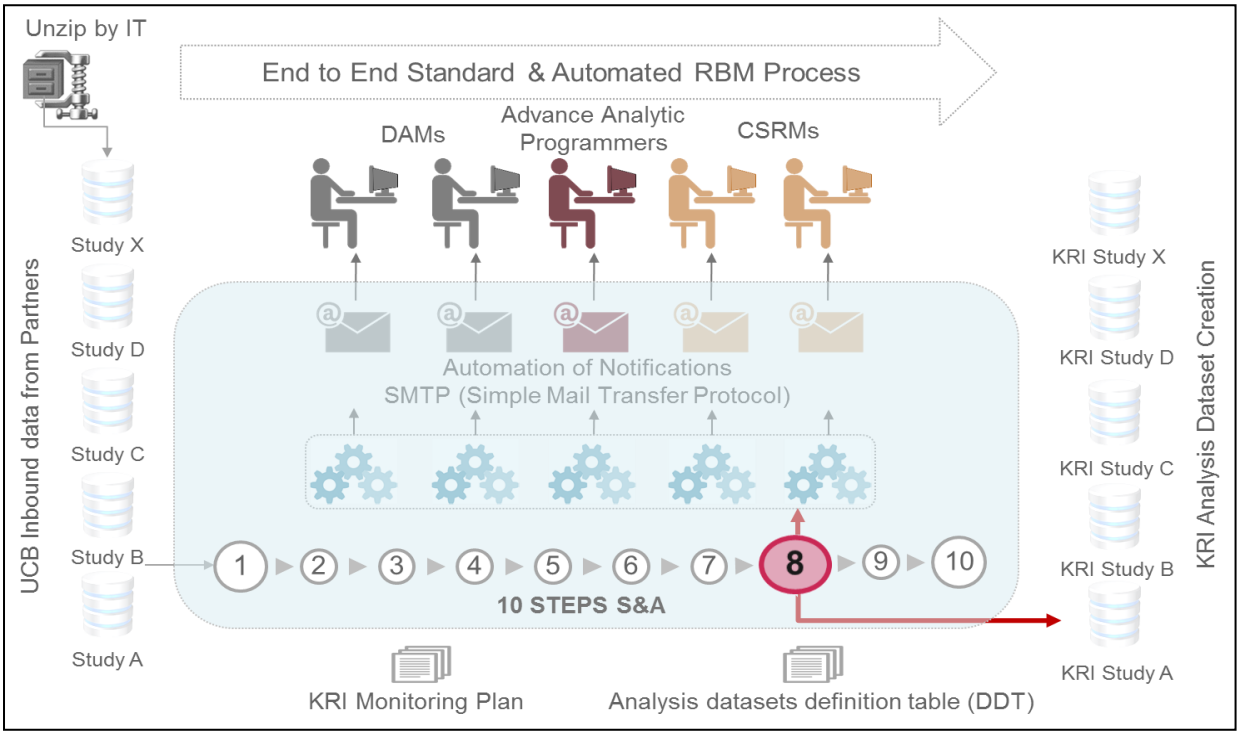

Figure 2 Design of the End to End S&A Process

PROCESS ARCHITECTURE

This paper concentrates on two SAS macros (ut_tstcd_mahap_dat_u.sas and v_dva_mahap_dat_u.sas) within step **8** (**Figure 3**) as an example to illustrate the process. The architecture of the S&A process for this step consists of one program (Master_run.sas) which drives the entire process. The program consists of two major blocks, one for the old process and the another for the new process. Master_run.sas sets the sequential order of the steps for the entire S&A process. (It is also the only program present; all other SAS files define macros.) The new process follows a star schema design where a macro called ut_kricreatval.sas is the center of the star and calls all KRI macros and utility macros.

The macros dedicated to the creation of the KRI analysis datasets are found within the KRI macros block. All other macros are considered to be utility macros. Separating macros by their objective makes them simpler and shorter and this is why the code is separated into two main areas. The KRI macros generate the output analysis datasets and utility macros generate values for macro variables and temporary datasets needed for the creation of the KRI analysis datasets.

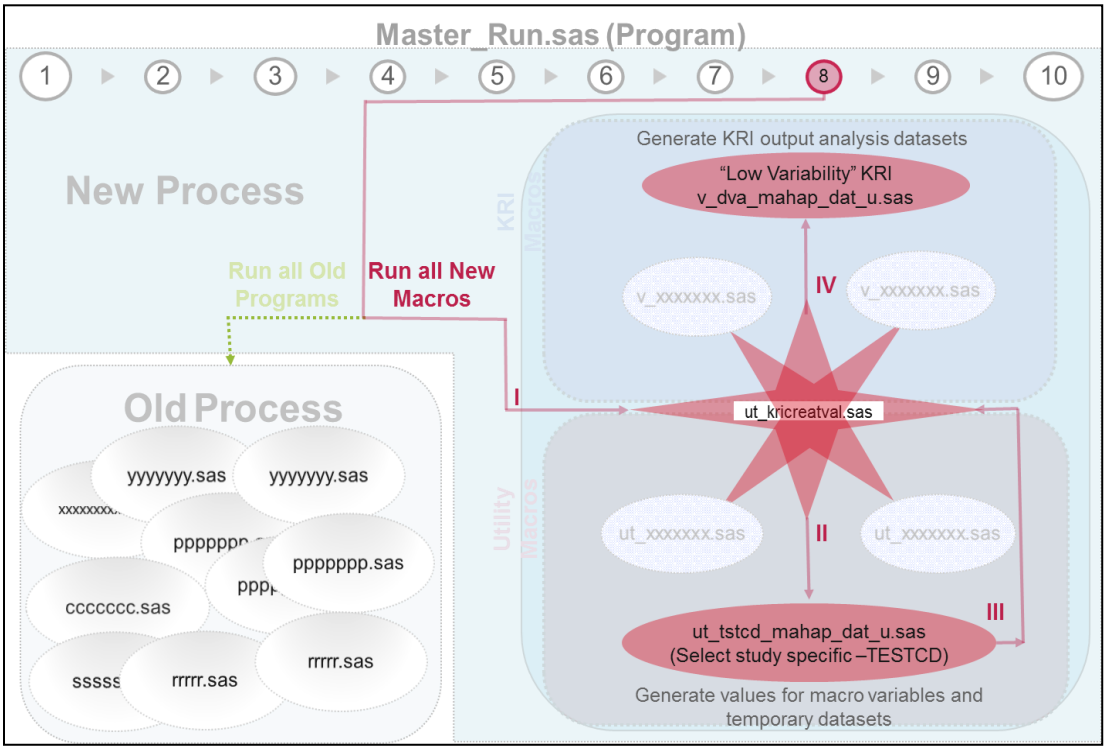

Figure 3 Star Schema – New Process

Another crucial design feature is that the center of the star is the only place where study-specific information needs to be stored. No other macros contain any study-specific information. This NFR makes the system as a whole more manageable and scalable. The center of the star serves as a container from which all the other macros pick up the information they need as shown in **Figure 4.** This simple schema fits well with the KRI non-functional requirements.

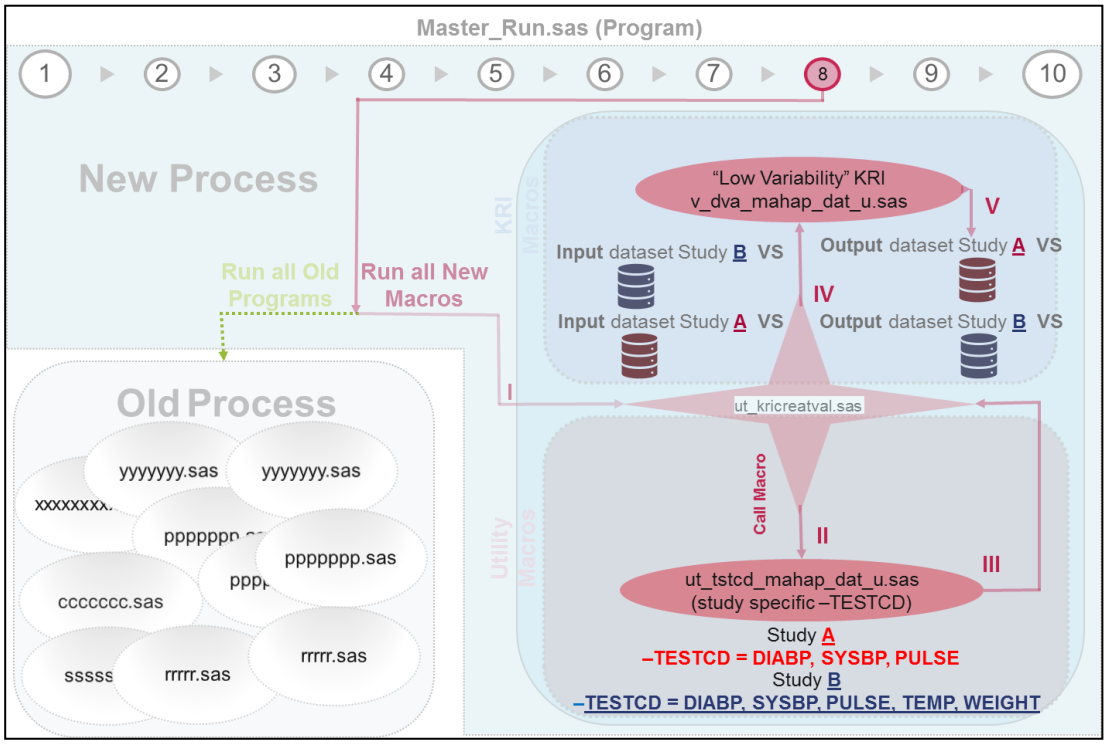

Figure 4 Creation of values for --TESTCD

To summarize **Figure 4: Master_run.sas** executes all ten steps in sequence. At step **8** it calls **ut_kricreatval.sas** (the center of the star), which in turn calls the utility macro ut_tstcd_mahap_dat_u.sas (II). This utility generates a list of test assesstments (--TESTCD) and passes it to the center of the star schema (III). This last macro then passes the list to the KRI macro **v_dva_mahap_dat_u.sas** (IV), which uses it to generate the dataset needed for the "Low Variability" KRI.

BENEFITS OF THE UTILITIES MACROS

ut_tstcd_mahap_dat_u.sas creates a list of assessments and passes it to **ut_kricreatval.sas**.(Step III, **Figure 4**). This list is later passed to the "Low Variability" KRI macro to create final datasets for the Low Variability KRI. Note that the list is study dependent and can vary from study to study. Code within the utility macro does not vary between studies. Only the values of the macro parameters change when the utility macro is called by ut_kricreatval.sas. By using this utility macro (**ut_tstcd_mahap_dat_u.sas**), all the study-specific information is external to the KRI macro, making the latter completely standard and generic.

Figure 4 shows the process flows for the "Low Variability" KRI for two different studies. The list of values of–TESTCD parameter differs between Study **A** (red) and Study **B** (blue). For study **A** the list resolves to Diastolic Blood Pressure (DIABP), Systolic Blood Pressure (SYSBP) and Pulse Rate (PULSE) whereas for study **B** it resolves to the same values as study **A** plus Temperature (TEMP) and Weight (WEIGHT).

Traceability: File names in the new process are based on their associated macro names. For example, the macro ut_tstcd_mahap_dat_u has a corresponding SAS file named ut_tstcd_mahap_dat_u.**sas,** log file ut_tstcd_mahap_dat_u.**log,** text file, ut_tstcd_mahap_dat_u.**txt** for proc compare output comparison, and finally a dataset named ut_tstcd_mahap_dat_u.**sas7bdat.** When a macro creates multiple outputs (based on different inputs), it is called multiple times and the different inputs are passed via parameters, thus avoiding hardcoding.

Macro Code Conventions: Uppercase font is only used for variable names, with all other code in lower case. Local Macro variables start always with "_" and Global Macro variables contain no underscores. Keyword parameters are used for the entire process and positional parameters are not allowed by design.

Nested macro definitions inside other macros are not allowed because these are usually unnecessary and inefficient. In fact, nesting macro definitions causes the macro processor to recompile the macro each time the outer macro is executed.

COHESION

Design & Structure of the Standard Macros:

In the old process, blocks of identical code were repeated within many programs, or the same program created many permanent output datasets (low cohesion). This is not allowed in the new process. In addition to separating KRI analysis macros from utility macros, very strict rules were adopted to increase cohesion. One of the most fundamental rules is that one SAS file can only contain one macro and can only create one permanent output analysis dataset. This brings many benefits, one of the most important being that one macro per file forces the algorithm and data to descend through the macro, hence, ensuring sequential flow of data and algorithms in one direction, from start to end.

Identical, repeating blocks of code in different macros were replaced by a single code block in one macro. If this piece of code produces a temporary dataset, then this dataset is created only once and shared for the entire duration of the process.

Modular Macros:

Modular macros make the code more readable, maintainable, portable, and reliable. It is much easier for individuals on a team to work independently on separate areas of the implementation without getting in one another's way. In modular programming, different types of modules can be created, such as data abstraction, process support, or a functional module. The functionalities of programs in the old process overlapped, with different programs performing similar (often identical) tasks. This resulted in much redundant code. A prime example is the loading of external files, where the same files are loaded multiple times by different programs. By contrast, in the new process they are only loaded once and their content made available to other macros where it is required.

In summary, the new process follows a modular design with algorithms separated into multiple modules based on their functional requirements. Switching from monolithic programs to modular macros takes longer to implement at the onset of a project but allows better control over the code, and long-term maintenance is cheaper. Modular macro development is a one-time cost whereas the cost of monolithic program development is ongoing for the duration of the project. Finally, modular macros increase cohesion, allowing use of SAS auto-call macros.

NON-FUNCTIONAL REQUIREMENTS

Reliability: Software Reliability reflects the software design. The more complex the software, the higher the probability that the software will lead to problems. Expressing reliability in quantitative terms is difficult and is out of the scope for this paper. The approach taken was to model historic failure data as a predictor of future behavior of the KRI macros. In this way macro errors can be anticipated and captured before they result in output errors or produce misleading errors. The highly organized code in the new process is more reliable because of its design.

Robustness: Within the new process, defensive programming is implemented within each SAS macro. Each macro includes checks for the existence of source datasets. Another example is when datasets contain known data issues, these are flagged and an algorithm exits the macro gracefully, without generating misleading errors, and allows the process as a whole to continue.

Scalability: Consider what happens in the monolithic structure of the old process, where the same programs are run for all studies within the same location. If a program is updated for a change specific to one study, that program can no longer be run for all other studies. These types of possible scenarios are accounted for in the new process. Scalability for the RBM Data Surveillance process is facilitated by parallel working environments for testing and production. The new process avoids hardcoding of any information that is likely to change, such as file locations and dataset names, by passing these values as macro parameters or within temporary datasets. This allows the macros to work in different environments without updates.

Maintainability: This is defined as the degree to which program code is understood and the effort required for repairs or enhancements. Complex programs are difficult to maintain. By contrast, maintainability is easy in the new process due to its modular design and strict adherence to design rules. Each macro is divided into blocks as described in **Table 1**

Table 1 Code Blocks Within Macros

Block	Description
1	Set SAS options, local SAS macro variables, debugging mode flag
2	Check for existence of datasets
3	Main body of the macro: extraction, merging, derivation, and statistical procedures
4	Create output datasets
5	Delete temporary datasets, restore user system options

Usability

The old process is hard to debug compared to the increased usability of the new process. For example, when running in interactive mode, a feature was implemented to automatically open the SAS macro in question whenever it throws an error. This feature makes the development phase much more efficient by avoiding time wasted in searching for the offending program file. A similar feature was added to the data validation steps where proc compare results are also opened automatically.

Handling of Errors, Warnings, Notice, User messages

For example, when the macro v_dva_mahap_**dat.**sas runs, it checks for the existence of the input dataset MAHAPDATLB and if it is not found in the temp library of SAS then the following message is written in to the log:
WARNING: [RBM UCB] V_DVA_MAHAP_DAT Temporary dataset MAHAPDATLB does not exist

Checking Logs

As explained in **Figure 3** Step **8**, the new process runs all programs of the old process, as well as all the new macros of the new process, then it parses all the logs to search for errors, warnings *et cetera* and finally compares the datasets for validation. A macro that checks the logs is called after each utility macro and KRI macro, searching the macro logs for errors, warnings, notes and user messages (these latter recognizable by the tag "[RBM UCB]") and outputting only these user messages in the log window for a more compact and readable log.

Dataset Validation

Validation results are nicely and **tidily** printed in a final log of the process by two utilities macros. These two macros do not only perform a traditional SAS proc compare between the development and validation datasets, they also compare the data observation by observation by the key variables of the datasets. Non-matching observations can be easily spotted because superfluous information produced by the output of the proc compare is removed helping both the developer and the validator to troubleshoot the issues.

SOFTWARE LIFECYCLE

The modular approach adopted for the new process greatly increased the number of macros. Updates for bug fixes or new functionality must be handled efficiently and account for interactions between modules KRI macros must be deployed in a flexible and user-friendly way while having an efficient method of validation and updates. As mentioned previously, this can be achieved with an automated system which manages the entire lifecycle of SAS macros that includes automated regression testing which we already have at UCB. (2)

D. CODE MANAGEMENT WITH A KNOWLEDGE GRAPH

Managing the complex interactions between KRI programs, supporting documentation, standards, data, and metadata is a daunting task. Traditional approaches using spreadsheets and other electronic files result in disconnected and inconsistent silos of information that are difficult to locate and reconcile. How can programmers quickly determine the impact of changes to the code or troubleshoot problems? What is the downstream effect of a code change? Where is the related documentation? Which SAS macros calculate a specific KRI and who authored the latest changes to the code? These are but a few examples of the challenging questions facing the RBM team.

Linked Data as a Resource Description Framework (RDF) (6) provides solution. When Linked Data is constructed using ontology (7) for classification, defining relationships, and enforcing rules, it is known as a Knowledge Graph (not to be confused with Google's Knowledge Graph: https://en.wikipedia.org/wiki/Knowledge_Graph). Knowledge graphs use semantic reasoners (8) to infer information not found in the original data. RDF graph databases are multi-dimensional in nature, allowing incremental schema development and the ability to overcome the restrictions of traditional row-by-column data structures. Therefore, a graph database can represent and link together the many disparate elements in an RBM project.

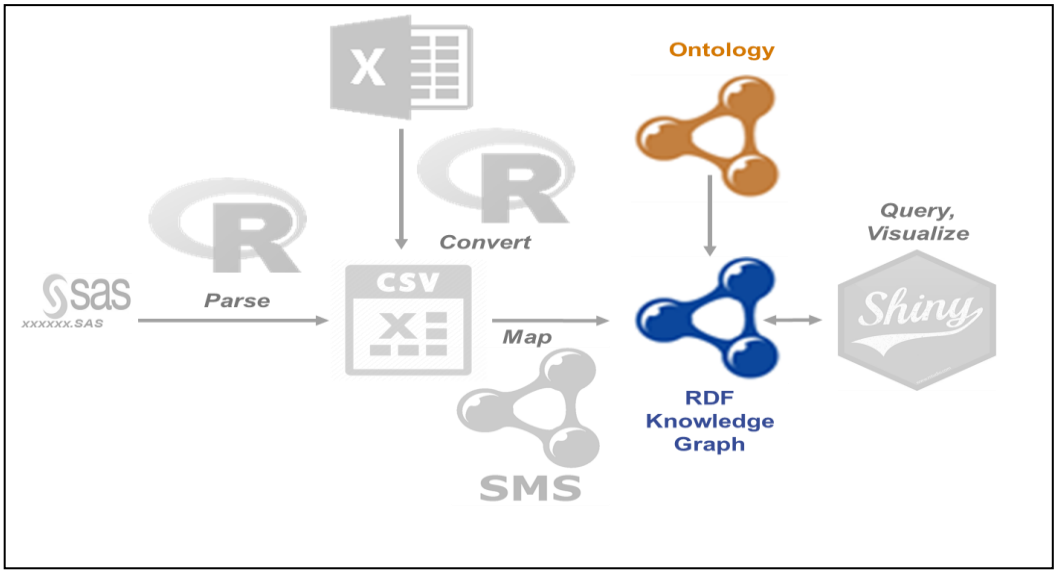

Figure 5 Conversion to Knowledge Graph

DATA CONVERSION

The first step in creating a knowledge graph for the project was to identify the many entities represented in the process and the relationships between these entities. Ontology development followed the approach outlined in Noy and McGuiness (9). Existing ontologies were leveraged where possible. R scripts extracted information from the SAS programs, including the macro calls, the location of the call within the program, and the parameter=value pairs within each call. Extraction of this information is only possible when the source SAS file is highly-standardized with consistent naming conventions throughout. The extraction and conversion process provided a feedback loop to the code developers, resulting in increased code quality and consistency. RDF's use of Unique Resource Identifiers (URIs), where an entity is represented using a globally unique identifier, ensured consistent and unambiguous representations across all aspects of the project.

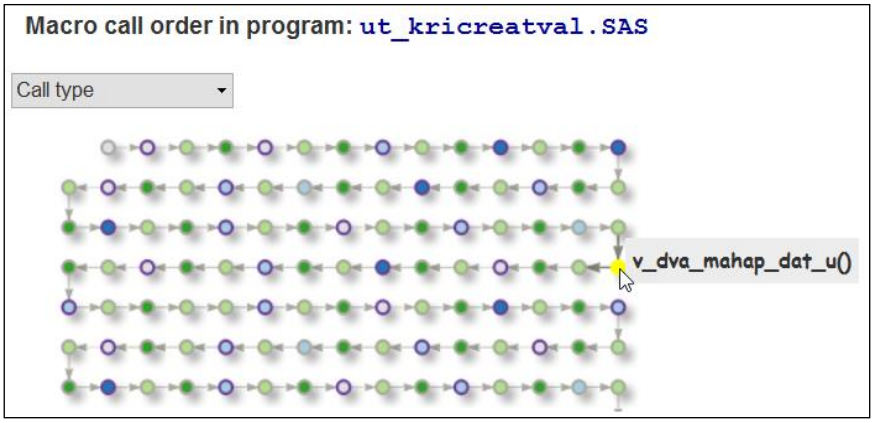

Figure 6 Macro Calls Within SAS Program ut_kricreatval.sas

The same macro is often called multiple times, with each call representing a unique execution of that macro. A unique identifier for each macro call was created using the first ten digits of an SHA-1 (10) hash ID created from the combination of the program name that contains the call, the line number on which it is located, and the full text of the call that includes the macro name and all parameter=value pairs. The hash value was appended to the macro name to create unique identifiers for each call, resulting in names like `macroname_7e0836cf03`, `macroname2_b81c1be4a1`. While not very human-readable, this method ensures that each macro call in the project was assigned a unique identifier. The output from this part of the process was a comma separated value (CVS) file containing unique macro call identifiers, their names, parameter=value pairs, and call order within the program. Additional information was converted from spreadsheets that the support the RBM Data Surveillance process, like descriptions of KRIs that are linked to specific macros using identifiers in both the spreadsheets and SAS program headers.

Stardog Mapping Syntax (SMS) (11) mapped the fields in the CSV file to the Stardog triple store. SMS is more user-friendly than R2RML (12) upon which it is based and can be ported back to the open standard. The SMS mapping files were created by manually cross referencing the content in the CSV files with the ontology developed using Protégé (13). The RDF query language SPARQL is not widely known in the pharmaceutical industry, so R Shiny interfaces to the data were developed. The example in **Figure 6** shows a subset of the call order and type of macros called within the program ut_kricreaval.SAS.

The graph is interactive, allowing identification of macros by their name (id) or type (group). Clicking on a node in the graph displays information about the macro call, including its location (line number), links to documentation for the macro, and all parameter=value pairs (**Figure 7**). The visualization and the macro details are generated at run-time by SPARQL queries executed from R Shiny to the RDF triplestore. Additional interfaces are in development, including the ability to quickly identify code and data (including down to the level of individual variables) for specific KIR impact analysis. Details will be provided in a future PhUSE presentation.

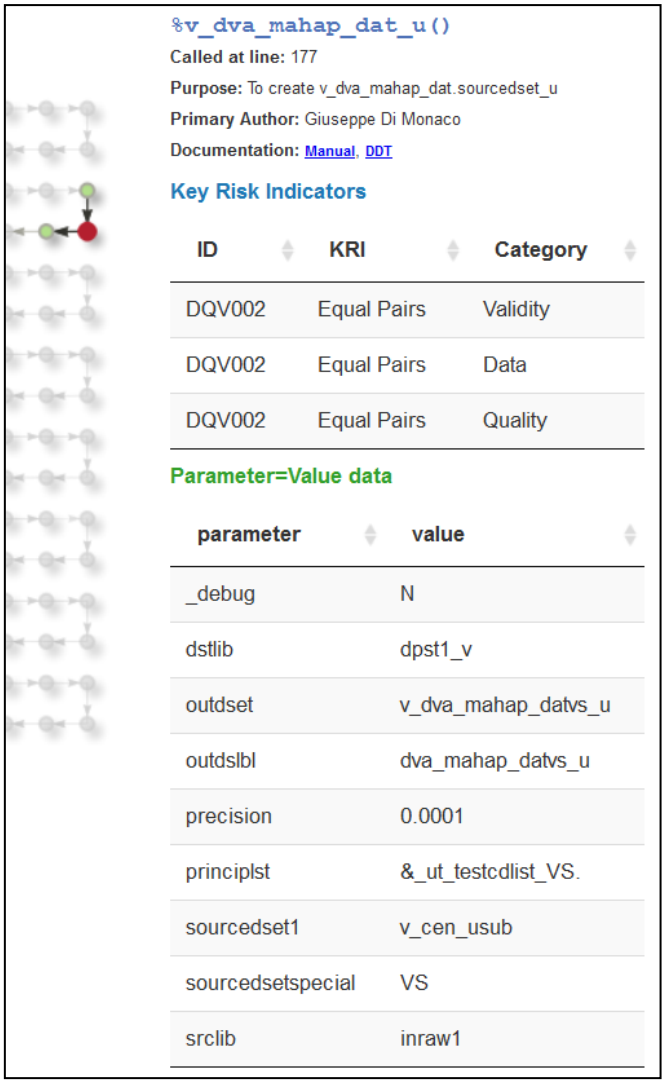

Figure 7 Macro Call Information

CONSIDERATIONS – (COMMON MISTAKES)

S&A IS AN ORGANIZATIONAL ATTITUDE

Having the right vision for an S&A process is a great way to start to focus on developing a proof of concept. However, you may soon realize that while vision and focusing are necessary conditions, project success depends on additional factors. Amongst these are the support of managers and other colleagues. Driving a significant level of standardization and automation can bring tremendous benefits, but to do that, organizations need to recognize that this is a fundamentally different way of doing things. It's not just like performing a task, writing a program or creating a document; S&A means ending up with digital processes where humans do not perform operational tasks anymore: they only provide process oversight. Therefore, if an organization does not recognize that S&A is a fundamental organizational attitude, rather than technology issue, S&A efforts will fail.

Another caution is to not fall into the "prototype trap" with the justification for picking a monolithic structure that "it's just a prototype!", with the intention that everything can be changed later. Too often, not everything can be changed, especially if changes are affecting other processes.

Finally, adopting standards and automating processes can help organizations become more accurate and therefore more efficient and potentially more profitable. However, if misapplied, automation will only give you bad information faster. Conventions and problems related to standards should be solved first, then the solution can be automated.

ONBOARDING PLAN AND DOCUMENTATION

Working on someone else's specification and programs is rarely easy, especially if those colleagues are no longer working at your organization. The incredible level of energy needed to understand things is astonishing, especially when working with monolithic programs. To keep things under control, the team should be as lean as possible until there is clear plan. Select your team carefully. The path to S&A requires people who are comfortable with change, keen to learn from others, and able to cooperate with each other while always aiming at the process objectives.

CONCLUSION

To conclude, clinical research organizations are investing huge resources to adopt RBM. Due to complexity of KRI programming, the common approach is to focus on the KRI functional requirements by developing prototype programs. Developing prototypes programs is especially useful to the development team during this phase. However, prototypes are often built as monolithic programs which are difficult to debug, learn, and modify. In the medium to long term this affects costs and quality. Although it can be challenging, implementing an S&A RBM Data Surveillance process is certainly the answer.

At UCB we are not just implementing an S&A RBM Data Surveillance process. We also took this opportunity to implement an end-to-end S&A process which covers RBM and its adjacent activities such as automation of inbound transfer data, oversight quality check *et cetera* as shown in **Figure 2**. To achieve that, we made a cohesive plan that did not address only one part of the project at a time, but instead looked at the overall process (**ZOOM-OUT**). Finally, the benefits brought by the end-to-end S&A RBM Data Surveillance process are countless. **Figure 8** shows just few.

We hope this paper will encourage the reader to invest some time exploring S&A solutions for the creation of KRI analysis datasets and associated documentation.

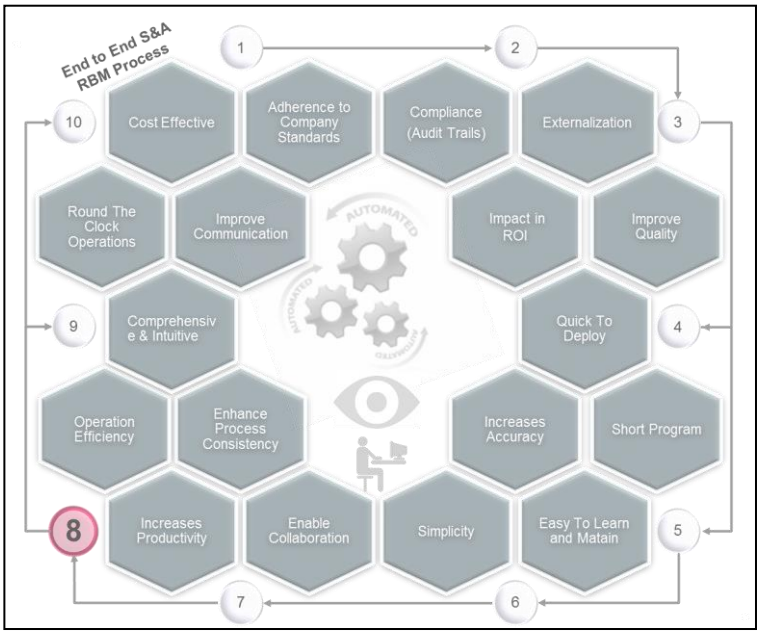

Figure 8 Benefits of the End to End S&A Process

REFERENCES

1. Cohesion (computer science)
   https://en.wikipedia.org/wiki/Cohesion_(computer_science)
2. Reengineering a Standard process from Single to Environment Macro Management by Giuseppe Di Monaco
   https://www.lexjansen.com/phuse/2014/pp/PP01.pdf
3. A Guide to Deploying Subversion for Version Control of SAS Programs in the Pharmaceutical Industry by Tim Williams, UCB BioSciences Inc, Raleigh, USA
   http://www.phusewiki.org/docs/Conference%202013%20AD%20Papers/AD01.pdf
4. Boosting the efficiency of data review by using SAS Integration Technologies and VBA by Giuseppe Di Monaco
   https://www.lexjansen.com/phuse/2016/cs/CS01.pdf
5. Software Re-engineering by Dr. Linda H. Rosenberg
   https://pdfs.semanticscholar.org/d6b2/89019f9fef924fd3300d1094f29c8bc079f9.pdf
6. Resource Description Framework (RDF)
   https://www.w3.org/RDF/
7. Ontology (information science)
   https://en.wikipedia.org/wiki/Ontology_(information_science)
8. Semantic reasoner
   https://en.wikipedia.org/wiki/Semantic_reasoner
9. Ontology Development 101: A Guide to Creating Your First Ontology by Natalya Noy and Deborah McGuinness.
   https://protege.stanford.edu/publications/ontology_development/ontology101.pdf
10. SHA-1
    https://en.wikipedia.org/wiki/SHA-1
11. Stardog Mapping Syntax. Stardog 5 "The Manual"
    https://www.stardog.com/docs/#_stardog_mapping_syntax
12. R2RML: RDB to RDF Mapping Language
    https://www.w3.org/TR/r2rml/
13. Protégé. A free, open-source ontology editor and framework for building intelligent systems.
    https://protege.stanford.edu/

ACKNOWLEDGMENTS

The authors would like to thank Brigitte Bernhoff for supporting this initiative. Without her contribution all this would have not been possible.

RECOMMENDED READING

Oversight of Clinical Investigations - A Risk-Based Approach to Monitoring by FDA
https://www.fda.gov/downloads/Drugs/GuidanceComplianceRegulatoryInformation/Guidances/UCM269919.pdf

Position Paper: Risk-Based Monitoring Methodology by TransCelerate
http://www.transceleratebiopharmainc.com/wp-content/uploads/2013/10/TransCelerate-RBM-Position-Paper-FINAL-30MAY2013.pdf

Reflection paper on risk based quality management in clinical trials by European Medicines Agency
http://www.ema.europa.eu/docs/en_GB/document_library/Scientific_guideline/2013/11/WC500155491.pdf

CONTACT INFORMATION

(In case a reader wants to get in touch with you, please put your contact information at the end of the paper.)
Your comments and questions are valued and encouraged. Contact the author at:

Giuseppe Di Monaco
UCB BIOSCIENCES GmbH
Alfred-Nobel-Straße 10
40789 Monheim, Germany
Tel: +49.2173.48.2091
Fax: +49.2173.48.1947
Email: Giuseppe.DiMonaco@ucb.com
Web: www.ucb.com