PhUSE 2015

Paper PP01

A Data Science tool for Data Reviewing

Giuseppe Di Monaco, UCB BioSciences GmbH, Monheim, Germany

ABSTRACT

Data Science arose from the blending of scientific and engineering methods. This new field requires expertise from different areas such as data cleaning, statistics, and software engineering (or IT). Naturally, a significant challenge is associated with obtaining in-depth knowledge in all of these areas. In order to overcome this, each member of our team is assigned to the tasks that best fit their strengths. We developed tools that cover different parts of the areas mentioned above. The architecture of these tools was designed using rigorous techniques for development, testing and validation with a view to enabling fully scalable and automated maintenance. This paper will focus on a tool called "Data Review" that ensures the data is as clean and consistent as possible. This tool can either work as an independent application or can be part of a greater project covering more aspects of data science.

INTRODUCTION

Often reviewers are asked to review data within a short timeframe. Our solution to this challenge is a software application called "Data Reviewer". Our goal is to improve the quality, efficiency and cost-effectiveness of the data review process. This project was implemented using software engineering methodology. This method allows the application to include ever more components needed during the review process (for example TFL creation or validation.) The application could also become part of a much greater Data Science project.

The application already contains about twenty general checks (also called modules). For brevity, this paper will focus on the process used for the creation of the application rather than on the details of the particular check. The check called "DRCHECKCODEDECODE" will be taken as an example to explain the processes and methods used. Each component of the application will be briefly described with a focus on user interface, application settings and output report.

PHASES FOR SOFTWARE DEVELOPMENT AND THE INCREMENTAL MODEL

Software development phases describe particular stages of the software lifecycle as well as the order in which those phases are executed. Each phase produces deliverables required as inputs by the next phase in the life cycle. In the development phase, requirements are translated into designs and program code is produced based on these designs. The testing phase covers activities aimed at validating the deliverable of the implementation phase against the requirements.

ANALYSIS

Formalizing user requirements is often a tough task; in fact, requirements are usually expressed in natural language which is imprecise and may contain ambiguities. This was not the only problem we encountered. Another important issue was that reviewers could not articulate concerns about their lack of confidence in the application. On the other hand, we were not able to formalize requirements that the reviewers did not express. This problem, together with others (such as a lack of standardized input data) slowed down the development during the analysis phase. As a result, the collection of the requirements was labor-intensive.

Below is a sample of some general user requirements collected for the Data Review application

- Expert Users have full rights to create, update and/or parameterize the modules.
- Non-Expert users are provided with a set of standard modules without the ability to change the tool's settings.
- The user Interface must be clear, concise, consistent, efficient and intuitive.
- Output Reporting must be effective and well-structured. It should follow the four C's rule: Clear, Concise, Complete and Correct.

Some definition created to meet user and system requirements are listed here:

- Modules are portions of programs that carry out a specific check needed by reviewers.
- A Component is a set of modules.
- The Application is the Data Review as a whole. Other components can be added as the project expands. The application must be scalable and accept any kind of (structured or unstructured) input data.

Finally, the application should also serve as the basis of a much greater Data Science project.

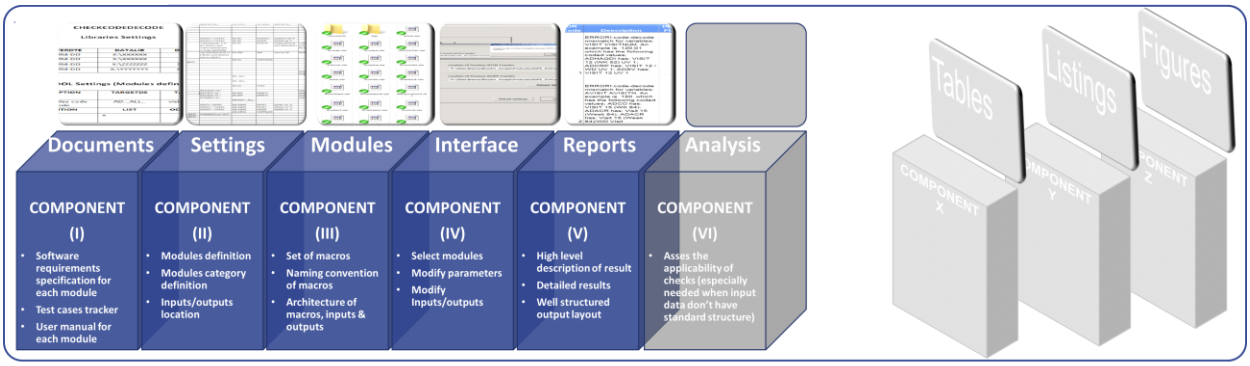
DESIGN

Designing the application was an enjoyable phase since it involved much creativity. During this phase, the architecture of the application was established. Components were defined together with their behaviors (see figure 1). How the application was going to meet the reviewers' needs and also how to make it effective and efficient were also aspects covered in this phase

Below are some of the concepts used during this phase:

- **Modularity** is a software design technique which breaks the functionality of a program up into independent components called modules. Modularity allows easy maintenance while facilitating the level of abstraction.
- **Design Patterns** are general, repeatable solutions to commonly-occurring problems in software development.
- **Abstraction** is used to identify which functions performed similar tasks so that we could combine them into a single more abstract function.
- **Cohesion** is a quality measure of the software modularity. High cohesion means the module is well focused on what it should be doing.
- **Coupling** refers to how dependent modules are on each other. Low coupling means that changing something major in one module should not affect the others.

FIGURE 1



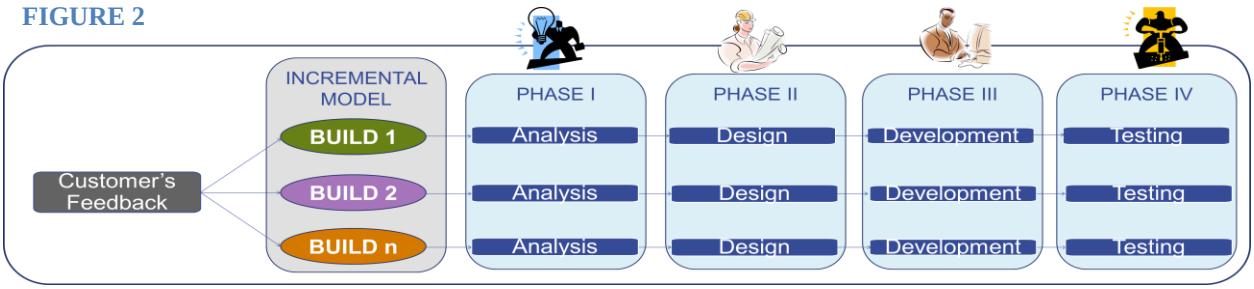
DEVELOPMENT

During this phase we put the incremental model into practice. Each component and each module was implemented incrementally and sent to the reviewer. This allowed utilization of the incomplete product while avoiding a situation in which reviewers would have to wait until development was fully complete. Based on reviewers' feedback, changes were planned to be ready for the next build. Dramatic changes were excluded since this model did not introduce a completely new system all at once. Incremental tests were easy to perform since few changes were made within any given iteration. This allowed for more targeted and rigorous testing of each element within the overall product.

INCREMENTAL MODEL

Selection of the model for the software development model was based mainly on business needs. Reviewers urgently needed automatic checks to facilitate the reviewing process. The most appropriate model that met this requirement was the so-called "incremental model". Here the product is designed and implemented incrementally. Each build is delivered to the reviewer (see figure 2) as soon as it is minimally functional. As mentioned, this avoids reviewers having to wait until development is fully complete. In addition, this model facilitates testing and debugging because few changes are made within any single iteration. Also, data reviewers can respond to each build.

FIGURE 2



TESTING

Although testing cannot establish that a product functions as intended under all conditions, it can certainly establish whether the software does not function properly under specific conditions. Hence, testing is the phase where can determine the correctness of software under the assumption of specific hypotheses. There are a number of software testing methods. We used the following methods: unit, integration, black box, white box, regression testing.
Here is some information on each testing methodology:

- **Unit testing** is the method whereby individual units of source code are tested, together with associated control data. They can be thought of as the smallest testable part of an application.
- **Integration testing** is a software testing methodology used to test the interactions between various software modules
- **Black box** testing is a technique in which the internal operations of the item being tested are unknown to the tester (also called functional testing).
- In **White box** testing, where the internal workings of the item being tested are known to the tester.
- **Regression testing** is the process of re-testing changes in programs to make sure that older versions of the same program still work with the new changes. (For more information see EMM presentation.)

DEVELOPMENT PHASES AND THE INCREMENTAL MODEL APPLIED TO ONE MODULE

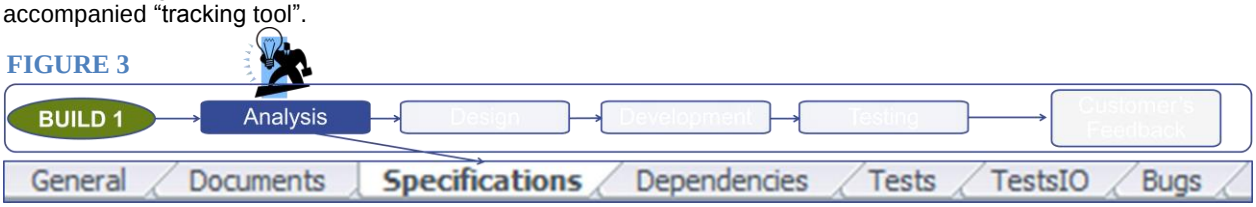
In this section of this paper, we discuss how we moved from the idea of a check, to a fully working module (BUILD 1), and from there to a much more useful and generalized module (BUILD 2). This was made possible by applying software engineering methods and the incremental model.

BUILD 1 OF "VISITREPORT" MODULE

ANALYSIS OF BUILD 1

There was a need within our department for the reviewers to have a report showing all instances of VISIT/VISITNUM used in all ADaM datasets for a given study. The report was needed to spot inconsistencies in the mapping of VISIT to VISITNUM. The information gathered during this phase was recorded in the "specification" sheet of a document called "tracking tool". This document contains the entire history of each module. Also, each module has its own accompanied "tracking tool".

FIGURE 3



DESIGN OF BUILD 1

For simplicity we focus on the design of the output report. The output report had the first two header columns dedicated to library names and dataset names. All other columns contain the actual VISITNUM. Hence, column visit10 stands for VISITNUM = 10. The contents of the report were reserved for the VISIT name.

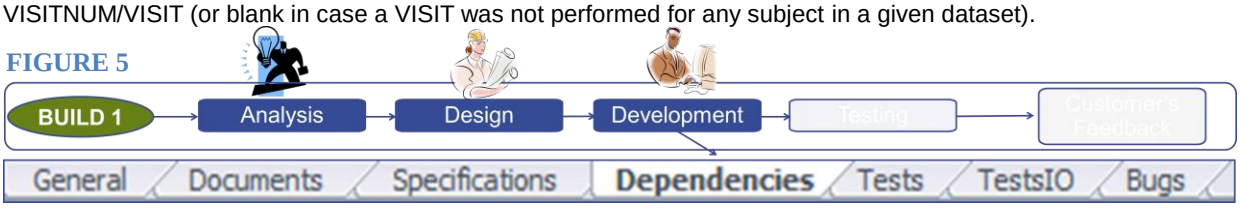
LIBRARY	DATASET	visit10	visit10_01	visit20	visit30
AD	ADLB				
SD	LB				

FIGURE 4

DEVELOPMENT OF BUILD 1

The development of this module was an easy task. The output should have listed all values of the tuple VISITNUM/VISIT (or blank in case a VISIT was not performed for any subject in a given dataset).

FIGURE 5



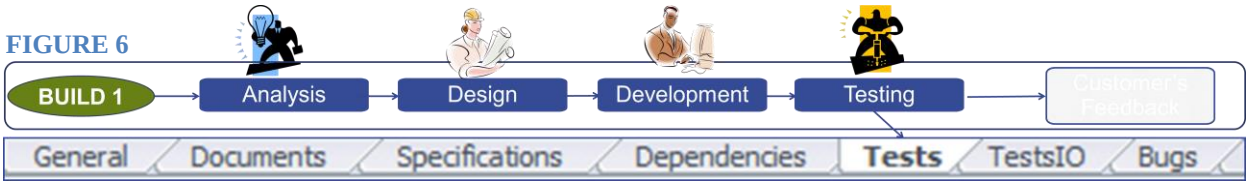
TESTING OF BUILD 1

After all planned tests had been performed and the bugs fixed, the module was able to list the issues as specified during the analysis phase. Table 1 shows a test case output created by this module. For VISITNUM =10, 20, inconsistencies were found. (e.g.: column visit10 shows “VISIT 1 (WK-2)” in ADEX, “VISIT 1 UV 1” in ADLB and “VISIT 1 (WK-2)” in ADLB).

TABLE 1

LIBRARY	DATASET	visit10	visit10_01	visit20	visit30
AD	ADEX	VISIT 1 (WK-2)	VISIT 1 UV 1	VISIT 2 (WK-0)	VISIT 3 (WK-0)
AD	ADLB	VISIT 1 UV 1	VISIT 1 UV 1	VISIT 2 (WK-0)	VISIT 3 (WK-0)
AD	ADLB	VISIT 1 (WK-2)	VISIT 1 UV 1	VISIT 2 UV 1	VISIT 3 (WK-0)

FIGURE 6



BUILD 1 SENT TO REVIEWERS FOR ACCEPTANCE

Once all development phases of the VISITREPORT module were complete, the output report was sent to the reviewers for acceptance. Unfortunately, the output report was rejected on the grounds that too much amount of information appeared in the output. This was true especially for those studies where the number of planned or unplanned visits was high. Therefore we needed a re-think to make the report more concise and useful for the reviewers.

FIGURE 7

BUILD 2 “VISITREPORT” BECOMES DRCHECKCODEDECODE

ANALYSIS OF BUILD 2

The following major user requirements were collected during the analysis phase of build 2:

1. This module should be able to accept any pair of variables. This change would make the module much more general and could thus be used in many other scenarios. (e.g.: VISITNUM/VISIT, PARAM/PARAMCD etc...).
2. The output report should contain two parts.
    a. The first part should list the issues in a concise and descriptive form. Only one instance of each issue should be listed. The selection of the issue to be listed should be based on the worst-case scenario.
    b. The second part should list all instances of the same issue. Also, this second part should only list real issues and filter out all pairs of VISIT/VISITNUM that had no issues. In other words, the check should be focused on the unwanted cases rather than listing everything and leaving the burden of filtering out the unwanted cases to the reviewers.)

Implementing these changes would meet the reviewers’ expectations to speeding up the review process of this module.

FIGURE 8

DESIGN OF BUILD 2

Before demonstrating the solutions we had found to the specifications of the analysis phase of build 2, we consider an example of a problematic mapping of VISITNUM/VISIT as shown in table 2. The gray section contains quite a few mapping issues. The aim of the specification of build 2 was to automatically filter out the green mappings while retaining the gray part in the report.

TABLE 2

LIBRARY	DATASET	VISITNUM	VISIT
XXX	YYY	1	VISIT 1
XXX	YYY	2	VISIT 1
XXX	YYY	3	VISIT 3
XXX	YYY	3	VISIT 1
XXX	YYY	4	VISIT 1
XXX	YYY	3	VISIT 5
XXX	YYY	6	VISIT 6
XXX	YYY	7	VISIT 7
XXX	YYY	8	VISIT 8
XXX	YYY	9	VISIT 9
XXX	YYY	10	VISIT 10

Solution for the implementation of the design specified in part "**1**" of the analysis phase of builds 2.

- "CODE" and "DECODE" as generic names to indicate a pair of variables.
- "DRCHECKCODEDECODE" as a name of the new module.

Solution for the implementation of the design specified in part "**2a**" of the analysis phase of build 2.

TABLE 3

Description
ERROR!: Code-decode mismatch for variables: <Code name>, <Decode name>. An example is <Decode Name> which has the following coded values; <Dataset Name> has: <Code Value>, <Dataset Name> has: <Code Value>, <Dataset Name> has: <Code Value>, <Dataset Name> has: <Code Value>

- Worst-case scenario selection:
  This part of the report should only list only one instance of each unwanted case and leave all the others to the part "2b" where all issues are listed. The case to be selected should be based on the worst-case scenario. For example, in table 2, VISITNUM = 3 is mapped to VISIT 1, VISIT 3 and VISIT 5 (see blue part) but VISIT = VISIT 1 is mapped to VISITNUM =1, 2, 3, 4 (see red parts). Hence, the worst-case scenario is represented by the red part and this should be listed within the first part of the report. So the report message would be:

TABLE 4

Description
ERROR!: Code-decode mismatch for variables: VISITNUM, VISIT. An example is VISIT 1 which has the following coded values; YYY has: 1, YYY has: 2, YYY has: 3, YYY has: 4

Solution for the implementation of the design specified in part "**2b**" of the analysis phase of build 2.

- Table 5 shows the detailed representation of all instances of an issue. The column headers contain the visit names as per the worst-case scenario logic explained above. Also, in this report we can still see that VISITNUM = 3 is mapped to VISIT = VISIT 1, VISIT 3 and VISIT 5 (3 different tuples). However, since VISIT = VISIT 1 is mapped to VISITNUM =1, 2, 3, 4 (4 different tuples) the layout of the report should be focused on making the reviewing of this scenario as easy as possible while not losing information relating to other issues. Finally, as the table below shows, no information is present for VISITNUM =6, 7, 8, 9, 10 since no issues were found in their mappings.

TABLE 5

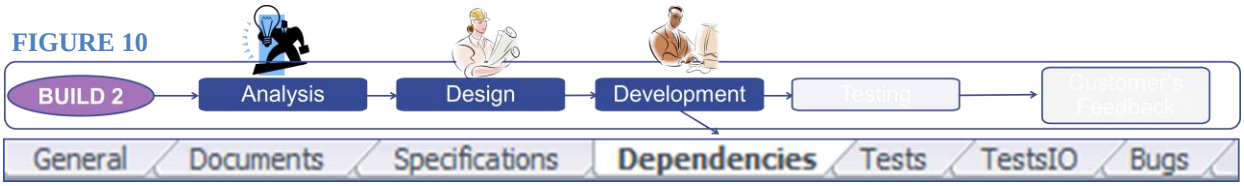
LIBRARY	DATASET	VISIT1	VISIT3	VISIT5
XXX	YYY	1	3	3
XXX	YYY	2		
XXX	YYY	3		
XXX	YYY	4		

FIGURE 9

DEVELOPMENT OF BUILD 2

The development phase of DRCHECKCODEDECODE was much more labor intensive than its predecessor, DRVISITREPORT. Only some parts of the previous code were reused for the new module.

FIGURE 10



TESTING OF BUILD 2

The testing phase for the DRCHECKCODEDECODE module was more extensive than its predecessor. This was needed to ensure that the module was really able to take any tuple of variables and to display the correct information. Some of the test cases used during testing are listed here:

- Test case for VISIT/VISITNUM where VISIT is mapped to multiple VISITNUM.

TABLE 6

Description
ERROR!: Code-decode mismatch for variables: VISITNUM, VISIT. An example is UNSCHEDULED which has the following coded values; ADDAS has: 11, ADDAS has: 4, ADLB has: 8.7, ADLB has: 996

TABLE 7

LIBRARY	DATASET	UNSCHEDULED	UNSCHEDUED VISIT 10.6	VISIT 1
AD	ADDAS	11	10.6	1
AD	ADDAS	4	10.6	1
AD	ADLB	8.7	7	2
AD	ADLB	996	4	2

- Test case for AVISIT/AVISITN where AVISTN is mapped to multiple AVISIT.

TABLE 8

Description
ERROR!: Code-decode mismatch for variables: AVISIT, AVISITN. An example is 100 which has the following coded values; ADACR has: Visit 10 (Week 36), ADDAS has: Visit 10 (Week 36)/WD

TABLE 9

LIBRARY	DATASET	40	100	110
AD	ADACR	Visit 4 (Week 2)	Visit 4 (Week 36)	Visit 11 (Week 52)
AD	ADACR	Visit 4 (Week 2)	Visit 4 (Week 36)W/D	Visit 11 (Week 52)
AD	ADACR	Visit 4 Week 2	Visit 4 (Week 36)WD	Visit 11 Week 52

- Test case for PARAM/PARAMCD where PARAMCD is mapped to multiple PARAM.

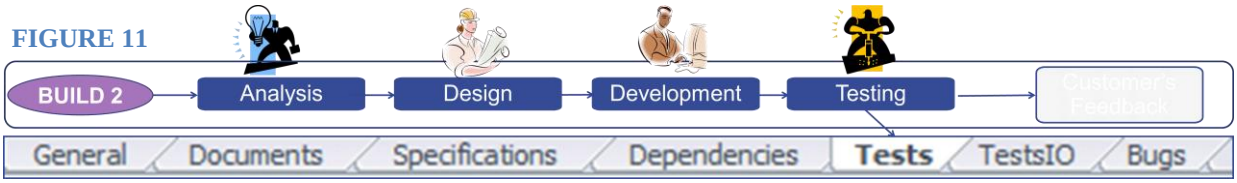
TABLE 10

Description
ERROR!: Code-decode mismatch for variables: PARAM, PARAMCD. An example is ALB which has the following coded values; ADLB has: Albumin, ADLB has: Albumin (g/L), ADLB has: Albumin (mg/L)

TABLE 11

LIBRARY	DATASET	ALB	ALBCREAT	ALT
AD	ADLB	Albumin	Albumin	Alanine Aminotransferase
AD	ADLB	Albumin (g/L)	Albumin/Creatinine	Alanine Aminotransferase
AD	ADLB	Albumin (mg/L)	Albumin/Creatinine	Alanine Aminotransferase (U/L)

FIGURE 11



BUILD 2 ACCEPTED BY REVIEWERS

When all development phases of the DRCHECKCODEDECODE module had been completed, the output report was sent to the reviewers for acceptance. This time the report was accepted and the module was integrated into the application.

FIGURE 12

APPLICATION SETTINGS & USER INTERFACE

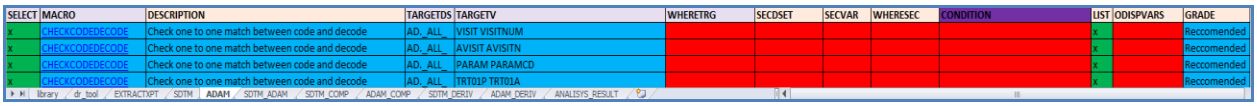
APPLICATION SETTINGS

The application includes a component called “settings” which holds the information to be passed to the modules. All modules can accept any type of input dataset, it is not necessary for the input datasets to be in SDTM or ADAM format. In other words, the same module can run on SDTM, ADAM or any other format of dataset. Only the parameters passed on to the module differ. For example, it would not make sense to run “DRCHECKCODEDECODE” with PARAM and PARAMCD over SDTM datasets. Hence the idea arose of categorizing the module’s definition based on the content of the parameters (where parameters are basically the contents of the column headers of figure 13).

CATEGORIZATION OF CHECKS

There are currently eight categories of checks: EXTRACTXPT, SDTM, ADAM, SDTM_ADAM, SDTM_COMP, ADAM_COMP, SDTM_DERIV and ADAM_DERIV. Each category has its own sheet. There are three additional sheets. Library and dr_tool are populated by the user interface based on the selection of checks to be run. The last sheet is ANALYSIS_RESULT and will hold information settings for a future component (which will analyze the data before running the tool).

FIGURE 13



Color Coding

Required parameter	Optional parameter	Not allowed parameter

PARAMETERS

- **SELECT**: To select the data review check to be performed.
- **MACRO**: The module definition. It specifies which module has to be selected and run.
- **DESCRIPTION**: Text describing the test being performed. This text is also used in the report generated by this tool.
- **TARGETDS**: This column identifies the dataset to be checked. The keyword _ALL_ can be used to indicate that all transferred datasets should be included in this data review check.

- **TARGETV**: When performing a data review check, specific variables will be evaluated. It can be a single variable or a group of variables.
- **WHERETRG**: There may be times when sub-setting the "target" dataset is required to evaluate data quality. Examples of sub setting include "VISIT=2", ETH=90, or MEVERB ne ''.
- **SECDSET**: Often another dataset is required to perform a data review check. For example, a check is needed to determine if the last dose date is after the end of the study. This would require the merging of ADMIN with EOST, with EOST as the secondary dataset. The secondary dataset can be the same as the target dataset.
- **SECVAR**: In addition to defining a secondary dataset, a secondary variable is also available. For some quality checks the secondary variable can be the same as the target dataset.
- **WHERESEC**: In some cases, a sub setting of the secondary dataset is required.
- **CONDITION**: Define the "description" in SAS ® code.
- **LIST**: An 'X' indicates that this test is to be displayed in the output report.
- **ODISPVAR**: List only the variables of interest to be displayed. These variables are in addition to the target (TARGETV) and secondary (SECVAR) variables.
- **GRADE**: To be used to determine the level of quality of the data being transferred.

USER INTERFACE

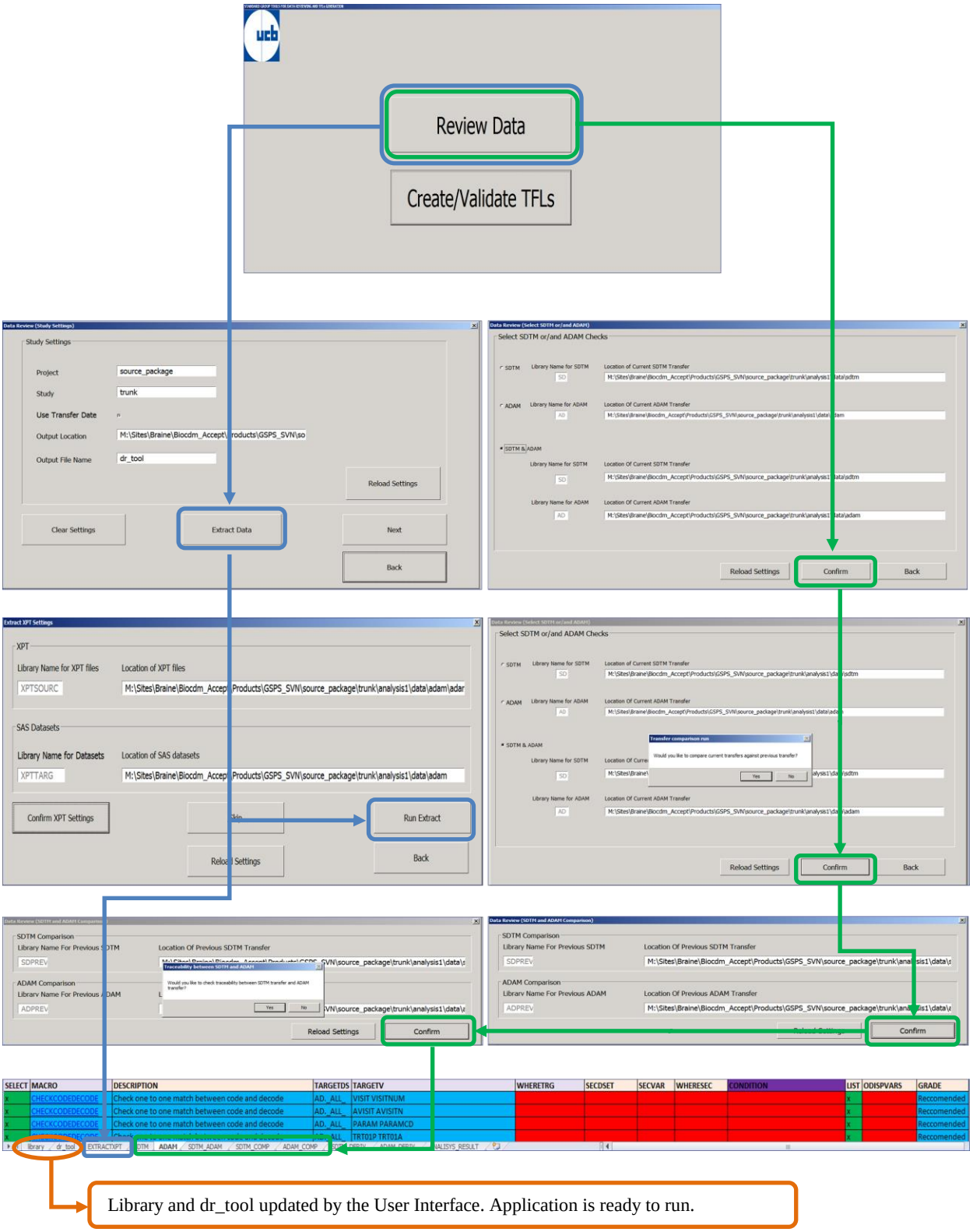
Initially the Excel spreadsheet was used as an interface by the users. However, after having some difficulties updating it, it was decided to build a proper interface to make the application easier to understand and to speed up the selection and the parameterization of the modules. There are still many edit checks to be moved from single modules to the user interface (e.g.: the presence of a certain dataset or variable should be checked by the interface before running any module). This avoids unnecessary runs of the application and subsequent log-checking. Using a proper interface made the application much more stable and easier to use.

All study settings, including inputs/outputs and the selection of modules is automatically handled by the user interface. The default values of the inputs/outputs are based on the location from which the interface runs.

Figure 14 shows an example of how the user interface is used to configure the settings. For instance, let's assume we want to run all SDTM and ADAM checks. In addition, we also want to run comparison of current SDTM and ADAM datasets against previous transfers of SDTM and ADAM. Finally, we also would like to check for traceability between SDMT and ADAM. Here follow the steps needed via the user interface (Figure 14).

1. Extract datasets from ".xpt" files, see blue line in Figure 14 (If the datasets are already in ".sas7bdat" format then this step can be skipped.)
2. Select SDTM & ADAM option buttons in the form called "Select SDTM and/or ADAM checks" and click confirm.
3. A message pops up asking if modules for comparing previously generated datasets against the current should be included. After confirming, a new form called "SDTM & ADAM Comparison" pops up. If default locations are fine than a confirmation opens another message box asking if modules for checking traceability between SDTM and ADAM should also is included. If the user confirms that then the traceability modules are also included.
4. Finally, the user interface modifies the settings (dr_tool sheet) of the application according to the selections. See green line in Figure 14.

FIGURE 14


Review Data
Create/Validate TFLs
Study Settings
Project
source_package
Study
trunk
Use Transfer Date
Output Location
Output File Name
dr_tool
Reload Settings
Clear Settings
Extract Data
Next
Back
Select SDTM or/and ADAM Checks
Confirm
Extract XPT Settings
XPT
Library Name for XPT files
Location of XPT files
XPTSOURC
SAS Datasets
Library Name for Datasets
Location of SAS datasets
XPTTARG
Confirm XPT Settings
Run Extract
SDTM Comparison
ADAM Comparison
SELECT
MACRO
DESCRIPTION
TARGETDS
TARGETV
WHERETRG
SECDSET
SECVAR
WHERESEC
CONDITION
LIST
ODISPVARS
GRADE
Library and dr_tool updated by the User Interface. Application is ready to run.

INTEGRATION WITH OTHER TOOLS

Integrating other tools with the Data Review should not be difficult since the application is made up of modules and components having high cohesion and low coupling. Our team can quickly assess the complexity of integrating the Data Review application with another application for TFLs generation.

CONSIDERATIONS

Though the overall experience was positive and helped with identifying issues, the user requirements phase turned out to be the most problematic. This reaffirmed that a “Data Scientist” needs to have really broad knowledge.

CONCLUSION

This paper gave an overview of the Data Review application and its main components. The application was created by following rigorous techniques for development, testing, verification, and making maintenance fully scalable and automated. The ease of the application setup (ensured by the user interface) and the structure of the output reports provide great benefits to those charged with reviewing data.

REFERENCES

Reengineering a Standard process from Single to Environment Macro Management by Giuseppe Di Monaco, Alfred-Nobel-Straße 10, 40789 Monheim, Germany
http://www.phusewiki.org/docs/Conference%202014%20PP%20Papers/PP01.pdf

ACKNOWLEDGMENTS

The author would like to thank David Hartman for providing an early version of the component module and all Statistical programmers and staff at UCB Biosciences Inc. for their continuing support.

CONTACT INFORMATION

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