**PhUSE 2009**

**Paper RG02**

# Requirements, Specifications and Programming – How to … cooperate with Statisticians in an effective way … accelerate the production of report objects … create technical specifications out of programs.

Dipl. Dok. Christoph Ziegler, F. Hoffmann - La Roche AG, Basel, Switzerland

## ABSTRACT

User requirements and programming specifications are essential aspects when it comes to the reporting of clinical trials by statistical programming. Essential, because program validation needs to be based on requirements/ programming specifications and because requirements need to be defined prior to unblinding, database closure and in general, before programming activities can start at all. Regulatory expectations also include the creation of requirements and specifications.

There are a lot of challenges when dealing with user requirements and programming specifications. Quite often, final and detailed user requirements are only available just before the reporting event.

This paper is discussing challenges but also options and methods to accelerate the production of report objects. It shows a method to automatically create technical specifications out of programs and shows a way to automatically create mock-up tables. Finally, it looks at ways of how to cooperate with statisticians in an effective way.

## INTRODUCTION

User requirements are the basis of all statistical programming activities. The problem is, that they usually keep changing and/or get updated on a regular basis. This is, for example, due to unforeseen data constellations and due to changes in the minds of the requestors/clients. As a result, final user requirements are usually only available just before the reporting event.

Programming specifications and user requirements should go hand in hand, which means that the specifications also need to be maintained on a regular basis in order to reflect changes and updates to the requirements. This process is quite time-consuming and resource-intensive. Programs are usually created on specifications which are based on non-final data and on non-final user requirements. This is because it is impossible to wait for really final requirements before writing program specifications and before doing the actual coding. Appropriate versioning and tracking of requirements, specifications and programs also need to be considered from a regulatory point of view.

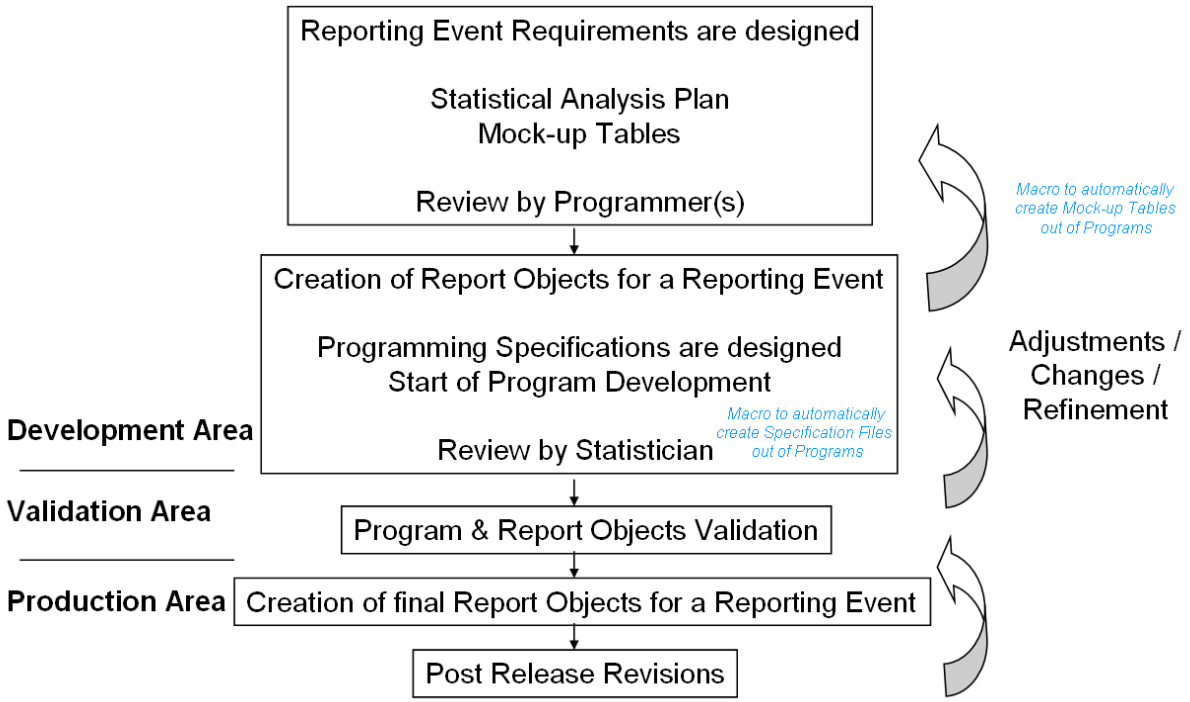
## HOW TO ACCELERATE THE PRODUCTION OF REPORT OBJECTS?

In order to address the aforementioned challenges and to accelerate the production of programs and report objects, several options are available. Improvements should be made on the requirements-side and on the specifications-side, i.e. both Statisticians and Programmers are affected. The communication paths and the way of working together are also of importance.

A macro to automatically create mock-up tables and a macro to automatically create a program specification document out of a program can be used to support the acceleration of the production of programs and report objects.

**THE SPECIFICATION AND PROGRAMMING WORKFLOW**

The rough process of creating report objects for a reporting event is as follows:




**USER REQUIREMENTS – WHAT DO GOOD USER REQUIREMENTS LOOK LIKE?**

User requirements are a key element in the reporting of clinical trials. The statistical analysis plan (SAP) should contain a description of the statistical methods (detailed if possible), derivations and calculations used for the efficacy and the safety analysis of a clinical trial and should be in harmony with the trial protocol. The SAP is written by the statistician in collaboration with clinical science.

The description of the statistical methods should be accompanied with mock-up tables (table shells) for each unique data display in order to be able to produce the requested report objects. Mock-up tables should always include annotations, i.e. the statistician should define all related calculations, derivations, exception rules etc. together with the mock-up tables. Those definitions should be done in text-form. The advantage of this approach is, that a lot of data problems, obstacles and possible difficulties can be identified beforehand, i.e. a lot of frontloading is done. Annotations done by the statistician lower the need for later questions by the programmers. By annotating mock-up tables, the statistician is able to identify possible layout problems and is able to see possible difficulties in derivation rules.

A typical mock-up table e.g. looks as follows:

```
xxx Summary Of xxxx
Protocol(s): xxxxxx
Analysis: Full Analysis Set
-------------------------------------------------------------------------------
                                        Drug A                  Drug B
                                    Score   No.  (%)        Score  No.  (%)
-------------------------------------------------------------------------------
Summary of the Means (Total Score)    xxx    xxx  (xx.x)     xxx    xxx  (xx.x)
Mean (Baseline)                       xxx    xxx  (xx.x)     xxx    xxx  (xx.x)
Mean (Treatment Timepoint 1)          xxx    xxx  (xx.x)     xxx    xxx  (xx.x)
Mean (Treatment Timepoint 2)          xxx    xxx  (xx.x)     xxx    xxx  (xx.x)
Mean (Treatment Timepoint 3)          xxx    xxx  (xx.x)     xxx    xxx  (xx.x)
Mean (Follow-Up Timepoint 4)          xxx    xxx  (xx.x)     xxx    xxx  (xx.x)

Change from Baseline to Timepoint 1   xx.x                  xx.x
Change from Baseline to Timepoint 2   xx.x                  xx.x
Change from Baseline to Timepoint 3   xx.x                  xx.x
Change from Baseline to Timepoint 4   xx.x                  xx.x
-------------------------------------------------------------------------------
```

Quite often, the statistical analysis plan does not include a detailed description concerning derivation rules and calculations. Therefore the programmer might come back to the statistician with a lot of questions in order to be able to produce the table described above:

- what to do with duplicates?
- how is the total score by time-point derived?
- should assessments or patients be counted?
- should sub-scores be included?

…

In order to avoid this vagueness and the questions, annotations done by the statistician will help. For the example mock-up table above, annotations could e.g. look as follows:

- Get the data from the XY dataset (parameter XY =’ab’,…,’ah’ and parameter XY is in (‘BASELINE’, ‘TIMEPOINT 1’, ‘TIMEPOINT 2’, ‘TIMEPOINT 3’, ‘TIMEPOINT 4’)
- The score computation should be done per patient (remove duplicates by using the sort key xy).
- Compute the number of scores being present per time-point and per patient as follows: a=number of assessments present, if a>2 then compute the score as follows: …
- Sub-scores should not be considered
- Split total score by treatment and compute the means and the change from baseline summarized for all patients
- The following footnote should be added: total score derived according to the algorithm described in xxxyyyy

A good mock-up table should include the following elements / information:

- clear and understandable structure of the layout
- detailed wording of the report object title and possible footnotes
- information concerning the population and/or sub-group analysis which should be applied
- information concerning decimal places
- annotations in written form in order to clarify all possible questions and vagueness beforehand
- a report object should be self-explaining, i.e. all important aspects of a report object should be present in the title / footnotes
- possible references to standard programs / macros which should be used in order to produce the report object

Though having annotations, some vagueness might remain and questions might come up during writing technical specifications and / or programming. However, well-defined mock-up tables with annotations help to frontload and to

avoid discussions at a later stage.

**USER REQUIREMENTS – HOW TO AUTOMATICALLY CREATE MOCK-UP TABLES?**

Mock-up tables need to be specified to a sufficient level of detail in order to be able to create the programs and to produce the report objects. Reality shows that problems might only be seen at the programming (programming specification) stage i.e. the specifications might need to be changed or updated. In order to avoid discrepancies between the actual programs (report objects) and the defined mock-up tables a macro to automatically create a mock-up (dummy) table for each report object can be used. It is only executed in the development area and the produced mock-up table can be used by the statistician in the statistical analysis plan. In the dummy report objects, all numbers are replaced with “x” (or any other character). The created dummy outputs are also helpful to confirm the layout with the initial requestor (e.g. clinical science). Programs are usually developed before database closure and before the actual statistical analysis plan is signed, i.e. the analysis plan is still draft and can be updated with the latest mock-up (dummy) report object by the statistician. The automatically created mock-up report objects are showing the detailed number of decimal places for numeric values, all footnotes, the title(s) etc. one to one.

It is important that the dummy report object is only created in the development area, i.e. before a program is promoted to the next stage (validation stage). This is because the mock-up table and the technical programming specifications should be agreed with the statistician before the program is passed on for validation.

The program to automatically create mock-up tables is called at the very end of a reporting program, i.e. after the actual report object was produced. The macro reads in the physical report object produced and creates a so called “dummy” output in the same directory where the report object is stored. Dummy in this sense means that all digits (0-9) are replaced with “x” or any other character in the “body part” of the report object. The suffix “_dummy” is automatically added to the original report object file name.

The macro features four parameters:

- **PAGEONEONLY**: Default = YES. Usually it is enough to produce the dummy output for the first page of a report object (PAGEONEONLY=YES). In case PAGEONEONLY is set to YES, the macro only writes the first page to the *_dummy file (digits in the body part replaced by "x" (default). In case the complete report should be dummy coded (all pages), define PAGEONEONLY=NO.

- **REPLACESTRING**: Default = 'xxxxxxxxxx'. Macro parameter to define the symbol(s) which should replace all digits in the body part of a report object. The following digits get replaced by the content of *&replacestring*: '0123456789'. Please note that the SAS®-function TRANSLATE is used inside the macro, i.e. each digit (0-9) corresponds to a "x".

- **BODYPART**: Default = 3. Macro parameter to define the "body part" of a report object. Usually it is number 3:

```
1 Title Block
------------------ <- standard line
2 Column Header(s)
------------------ <- standard line
3 Body
xxxx
------------------ <- standard line
4 Footnote Block
```

  In case e.g. spanning headers are used and in case the length of the spanning header line is equal to the standard line, the body part would be number 4:

```
1 Title Block
------------------ <- standard line
2 Spanning Header
------------------ <- spanning header line
3 Column Header(s)
------------------ <- standard line
4 Body
xxxx
------------------ <- standard line
5 Footnote Block
```

- **COLUMN1**: Default = 1. Macro parameter to define from which character onwards the digits should be replaced in the body part of the report object. In the example below, COLUMN1 was set to 1, i.e. the digits in the labels also get replaced (e.g. xx Months). If this is not wanted, COLUMN1 can e.g. be set to 20, i.e. the macro replaces the digits from character 20 onwards.

```
Summary of xxxxx
Protocol(s): AAAAAA
Analysis Population: SAP (N=AAA)

---------------------------------------------------------------------------
                                        Subgroup: Female
                                ----------------------------------------

                                Drug A        Drug B        Total
                                N=..          N=..          N=..
                                No.  (   %)   No.  (   %)   No.  (   %)
---------------------------------------------------------------------------

Observation Time (Months)
  Mean                           xx.x          xx.x          xx.x
  SD                             xx.xx         xx.xx         xx.xx
  Median                         xx.x          xx.x          xx.x
  Min                             x             x             x
  Max                            xx            xx            xx
  n                              xx            xx            xx

Number Of Patients Observed
  >  x Months                     xx ( xx%)     xx ( xx%)     xxx ( xx%)
  >  x Months                     xx ( xx%)     xx ( xx%)     xxx ( xx%)
  >  x Months                     xx ( xx%)     xx ( xx%)     xxx ( xx%)
  > xx Months                     xx ( xx%)     xx ( xx%)     xxx ( xx%)
---------------------------------------------------------------------------
Footnote aabb
```

In case no report object is produced, e.g. due to a program error, the macro recognizes that the output was not produced and skips the creating of the dummy-report object.

#### PROGRAMMING SPECIFICATIONS – HOW TO CREATE TECHNICAL SPECIFICATIONS OUT OF PROGRAMS?

Programming specifications should ideally be written before any coding starts and should reflect the user requirements. Specifications are usually written in external documents. It is often difficult to link versions of specifications to versions of documents, especially in case the company-specific development environment does not support this functionality.

A macro to automatically create a specification document out of a program can be used in order to have specifications and code closer together. This macro would create a specification document by reading program comments (which are marked in a specific way) for a reporting program and for additional called macros (if applicable). The macro is only executed in the development area before the respective programs gets loaded to the company specific version control system and before any validation starts. Later on, the created specification documents could easily be combined to a single programming specification document. The specification file created for a program can be sent to the statistician for review. As soon as the requirements are met and as soon as the specifications are confirmed (by e.g. another programmer), the program can be loaded to the version control system. Whenever a program gets updated and re-run, the specification file would automatically be updated. The advantage of this automated specification approach is the fact, that specifications and programs are written together, i.e. within the development system, and that the link between programs and specifications is easier.

The macro to automatically create a specification file out of a program works as follows.

The macro is able to gather specially marked comments in reporting programs (*.sas® - files) and to create two specification documents with identical content (a TEXT- and a PDF®-file) out of the comments gathered.

The code inside the macro itself is only executed if the program where the macro is called is run in the development area.

When developing / designing a new program in the company specific development area, comments can be marked using "_START_SPECS" and "_END_SPECS" at the beginning of a line (optionally with leading blanks). All specifications can either be written in one "_START_SPECS" and "_END_SPECS" block or in more than one block. Before loading the program to the company specific version control system, a specification file (*.txt / .*pdf®) can be created for the program.

The specification file is also created if an existing program is e.g. called again in the development area for another protocol or snapshot.

The naming convention is as follows:

```
Report objects folder/<program_name_in_which_the_macro_is_called>_specs.txt
Report objects folder/<program_name_in_which_the_macro_is_called>_specs.pdf®
```

Both files are produced using the following page-size settings: LS=156 PS=52
It is also possible to include relevant/similar specifications of called macros, if applicable. The text within the *.txt/*.pdf® files can then be used for the official technical specification document.

Single specification documents can be combined to one single technical specification file using e.g. Adobe Acrobat®.

The macro to automatically create a specification file out of program comments should be called in the main reporting program. It reads the physical file of the reporting program (also the files of dependent /additional macros if specified in macro parameter ADDITIONAL_FILES) and creates two specification documents (in *.txt and *.pdf® format) out of the marked comments. The files are stored under the directory where the program is stored. The layout of the *.txt / *.pdf® files looks as follows:

```
Programming Specification For XXXXXXXXXXXX.SAS®                          1
-------------------------------------------------------------------------
Program Name : /path/xxxxxxx.sas®
Date/Time : DDMMMYY/HH:MM
Specs xxxxxx
```

The specifications between the _START_SPECS and _END_SPECS tags are printed one to one, i.e. formatting / layout etc. can be done within the comments.
There is a limitation of 156 characters per comment-line. The report itself is produced with PROC REPORT, but no FLOW option is used to print the specifications.
In case the macro is called within a program which contains no marked comments, no *.txt or *.pdf® files are produced within the respective folder.
Some reporting programs include other dependent macros. e.g. test.sas® calls test_macro.sas® which is stored in another folder of e.g. the respective directory. The macro features a parameter (ADDITIONAL_FILES) where dependent / additional programs / macros can be listed (separated by blanks and without the suffix *.sas®). In case additional files are specified, the macro also reads the specially marked comments of those programs. The macro uses the company specific autocall- search order. In case the program is found in one of the directories defined in the autocall order, files with the same name which are higher in the hierarchy are ignored.

Within the respective *.txt / *.pdf® files, the comments / specifications of the additionally specified files are separated as follows:

```
---------- ADDITIONAL SPECIFICATIONS ----------
Program Name : <path and name of additional program x>
```

In case a program name is specified which does not exist, a WARNING is printed to the LOG:

```
WARNING: The specified program xxx was not found within the paths of the autocall
order defined in xxxx
```

In case a program is specified which contains no specially marked comments / specifications only the following part is printed:

```
---------- ADDITIONAL SPECIFICATIONS ----------
Program Name : <path and name of additional program x>
```

A specially marked comment / specification could look as follows:

```
/*_START_SPECS
1. Read in data where xxx and yyy
2. Derive zzz as follows
- xxxxxxx
- yyyyyyyy
...
_END_SPECS*/
```

The complete formatting (indentation / numbering / empty lines etc) should directly be done within the tags. Specifications should not be written directly after _START_SPECS / before _END_SPECS on the same line because those lines will not be kept in the specifications-document.
If e.g. runtime code should also be included in the specification document, this can be done as follows:

```
/*_START_SPECS*/

proc print data=test;
run;

/*_END_SPECS*/
```

The code between _START_SPECS / _END_SPECS is then taken over to the specification file.

#### PROGRAMMING SPECIFICATIONS – HOW TO MEET QA RELATED EXPECTATIONS?

Reality shows that it is very difficult and error-prone to write program specifications without writing a single line of code. In order to meet regulatory expectations, the macro to create a specification file out of specially marked program comments gets only executed in the development area, e.g. the specification files are available before the actual programs get loaded to the version control system. The macro to automatically create a dummy report object should have the same behaviour. This ensures that the requirements document (SAP) is updated before the program creating the report object gets loaded to the version control system.

#### HOW TO COOPERATE WITH STATISTICIANS IN AN EFFECTIVE WAY?

Automatically created specification files should be reviewed by the statistician and can also include questions, comments and notes. The statistician should directly comment and answer within the specification file. This ensures traceability, documentation and consistency. If necessary, open questions can be discussed face-to-face, but the outcome should be documented within the specification file. The programmer will then implement any possible changes or updates and a new specification file is automatically created. By annotating mock-up tables in the SAP, a lot of frontloading is done by the statistician and (data) problems can be detected at an early stage.

## CONCLUSION

Well annotated table shells help to accelerate the production of programs and report objects for a reporting event. By using a macro to automatically create mock-up (dummy) report objects, possible late changes in the layout of report objects can easily be harmonized with the statistical analysis plan. The macro to create automated specification files helps to bring technical specifications and programming closer together. Programs and technical specifications go hand in hand, i.e. code and specifications get updated at the same place. Due to the fact that that both macros are only executed in the development area (before programs get loaded to the version control system) it is ensured, that the mock-up tables and specifications are present before the program starts the development life-cycle.

## REFERENCES

n/a

**CONTACT INFORMATION**

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

Christoph Ziegler
F. Hoffmann – La Roche AG
Malzgasse 30, Bldg. 670
4070 Basel
+41 61 68 89200
christoph.ziegler@roche.com
www.roche.com