PhUSE 2010

Paper RG04

Validation Strategies for Report Objects and Programs – Challenges and Obstacles in a Risk Based Reporting World

Christoph Ziegler, Dipl.-Dok., F. Hoffmann – La Roche AG, Basel, Switzerland
Beate Hientzsch, Accovion GmbH, Eschborn, Germany

ABSTRACT

When reporting clinical trials by statistical programming, several validation methods are available for report objects (analysis datasets / outputs / etc) and their underlying programs. User requirements and specifications are the basis of all validation activities. Validation methods are usually applied based on the risk assessment of programs and / or report objects.

Common validation methods for programs and report objects are a) standard code review, b) extended code review, c) double coding, d) regression testing and e) unit testing. Each validation strategy has its challenges, advantages and disadvantages and is applied on risk based decisions. Time and resource considerations also need to be taken into account.

This paper is explaining and discussing the different validation strategies and shows advantages, disadvantages, challenges and raises various questions around validation of programs and report objects. It also tries to look for answers and best practices and touches related aspects of validation such as version control and quality of code.

INTRODUCTION

Completeness, accuracy and traceability of clinical trial results and summaries are essential with respect to human health and the associated regulatory requirements. Therefore validation of reporting programs and report objects became common practice but remain a hot topic. High quality is the ultimate goal but due to continuous resource and budget constraints as well as time pressure the implementation of the most effective processes and methods is the common objective across the pharmaceutical industry and within CROs.

This paper focuses on validation strategies involving a 2nd line-, i.e. QC-programmer:

- review of report objects
- standard code review
- extended code review
- double coding

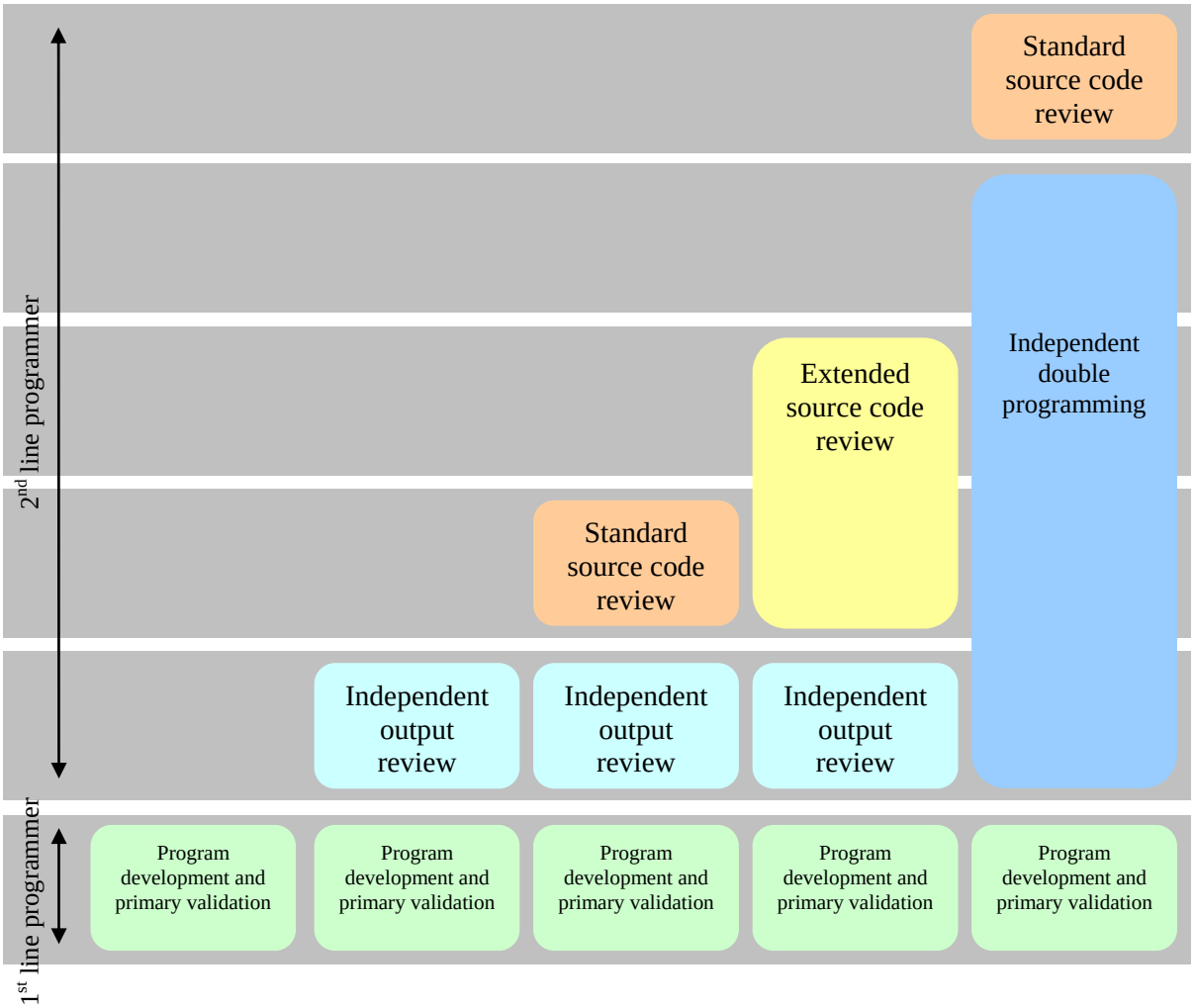
These validation strategies became standard for the validation of reporting programs and report objects for the analysis of clinical trials and overall clinical summaries. Besides assuring the correctness of report objects, validation also covers compliance checking of source code against industry and company standards, e.g. Good Programming Practice Guidelines.

Formal validation of programming specifications is not established as common standard even though programming as well as validation mainly relies on these specifications. Specifications are often developed with minimal or no data being available and the risk of incomplete or inaccurate specifications should always be taken into consideration during program development and validation.

Regression and Unit Testing are mainly applied to global standard reporting macros and are not covered in this paper.

Double coding (also known as double programming or qc-programming) is usually seen as the strongest validation method in the pharmaceutical industry and is mostly applied to high risk report objects and programs. However, this validation strategy is quite time consuming and resource intensive.
Extended Code Review is a combination of normal code review and testing of code by running it (e.g. interactively).
Standard Code review is defined as reading through source code and understanding it.

The overview below shows the different models, responsibilities and associated workload:



VALIDATION STRATEGIES FOR REPORT OBJECTS AND PROGRAMS

VALIDATION PROCESS

In accordance with FDA recommendations the validation method applied to reporting programs and report objects (analysis datasets, tables, listings, graphs) is usually determined based on a risk assessment at the planning stage of a reporting event (delivery of a certain set of report objects at a certain time-point to satisfy business needs, e.g. Clinical Study Report, Data Safety Monitoring Board or an ad-hoc request for publication work). The respective processes are specified in the company specific SOPs, usually along with associated supportive documents, e.g. checklists and documentation templates.

For the generation of a report object
the 1st line programmer will

- develop the program in accordance with the respective specifications
- check that the program was developed according to the applicable programming standards
- review the log-file thoroughly, i.e. no errors or warnings should remain, all notes need to be acceptable and correct
- run program step by step and check interim results

- review the outputs via checking results and metadata as well as adherence to the respective specifications
- pass the program to the 2nd line programmer, e.g. via promoting the program to validation

the 2nd line programmer will

- validate the program and the respective report objects according to the validation model assigned during risk assessment and according to the processes specified in the respective SOP
- pass the program together with the validation findings back to the initial programmer if corrections are required
- move/promote to production

REVIEW OF REPORT OBJECT

Regardless of the validation model assigned, review of the report object itself is essential and needs to be done at least by the 1st line programmer but preferably also by an independent reviewer. Output review comprises but is not limited to the following:

- Check adherence to requirements and specifications, e.g. Statistical Analysis Plan, Table Shells, Dataset specifications
- Check accuracy of layout
- Verify plausibility of results, e.g. via spot checks, checks across outputs, checks against source data

All report objects should undergo a review cycle before delivery. Outputs already validated but recreated at a later stage e.g. for the final run should be compared against the validated version preferably supported by a programmatic approach, i.e. automatic archival of previous versions and electronic comparison of analysis datasets as well as table outputs.

STANDARD CODE REVIEW

Standard Code Review is characterized as reading through the program source code and checking

- adherence to underlying specifications, e.g. statistical analysis plan, dataset specifications, table shells
- logic of the program
- plausibility of assumptions made
- correct selection of datasets / variables / formats
- correct merging of datasets
- handling of exceptions
- adherence to standards, e.g. program header, comments, Good Programming Practice Guidelines

Besides review of the report object only, this validation method is considered being the most weak form of program validation and it is recommended to use this method for low risk report objects e.g. listing report objects only. This is because

- the qc-programmer does not run the source code interactively
- it is necessary to understand the code and its functionality just by reading through the code
- certain aspects of the code might be overseen or missed

EXTENDED CODE REVIEW

Extended Code Review is defined as a combination of Standard Code Review and testing of code by running it (or pieces and parts) interactively. Extended Code Review is regarded as a stronger validation method than Standard Code Review because the respective programming code is not just only “read through”, it is also tested and challenged by running certain parts of the code. This enables the qc-programmer to

- test and check certain parts of the code or the entire program
- look at interim datasets
- verify the expected results of interim datasets
- review the log-file thoroughly
- check that all possible data scenarios are covered by the code
- understand the code and its functionality better than when just reading through the code

One disadvantage of Extended Code Review is that the code usually does not get re-checked and re-tested once updated data comes in. This means that the qc-programmer needs to check if all possible data scenarios are correctly covered by the program. Issues related to unexpected changes in the data will only be detected if unexpected results will be seen when reviewing the report object again.

DOUBLE CODING

Double programming is particularly relevant for the validation of high risk report objects. Nevertheless there are different implementation concepts ranging from doing double programming within the same working environment to letting it be done by another company. The quality of the underlying specifications and the status of the clinical data are crucial for success in terms of time and resources needed as well as in terms of quality. Automatic comparison of the results and user-friendly documentation strive for efficiency but also for traceability of the process itself.

DEFINITION

Double Coding means verification of a report object by another program written by the 2nd line programmer and reproducing the results of the report object independently, i.e. without referring to the initial program and the results themselves.

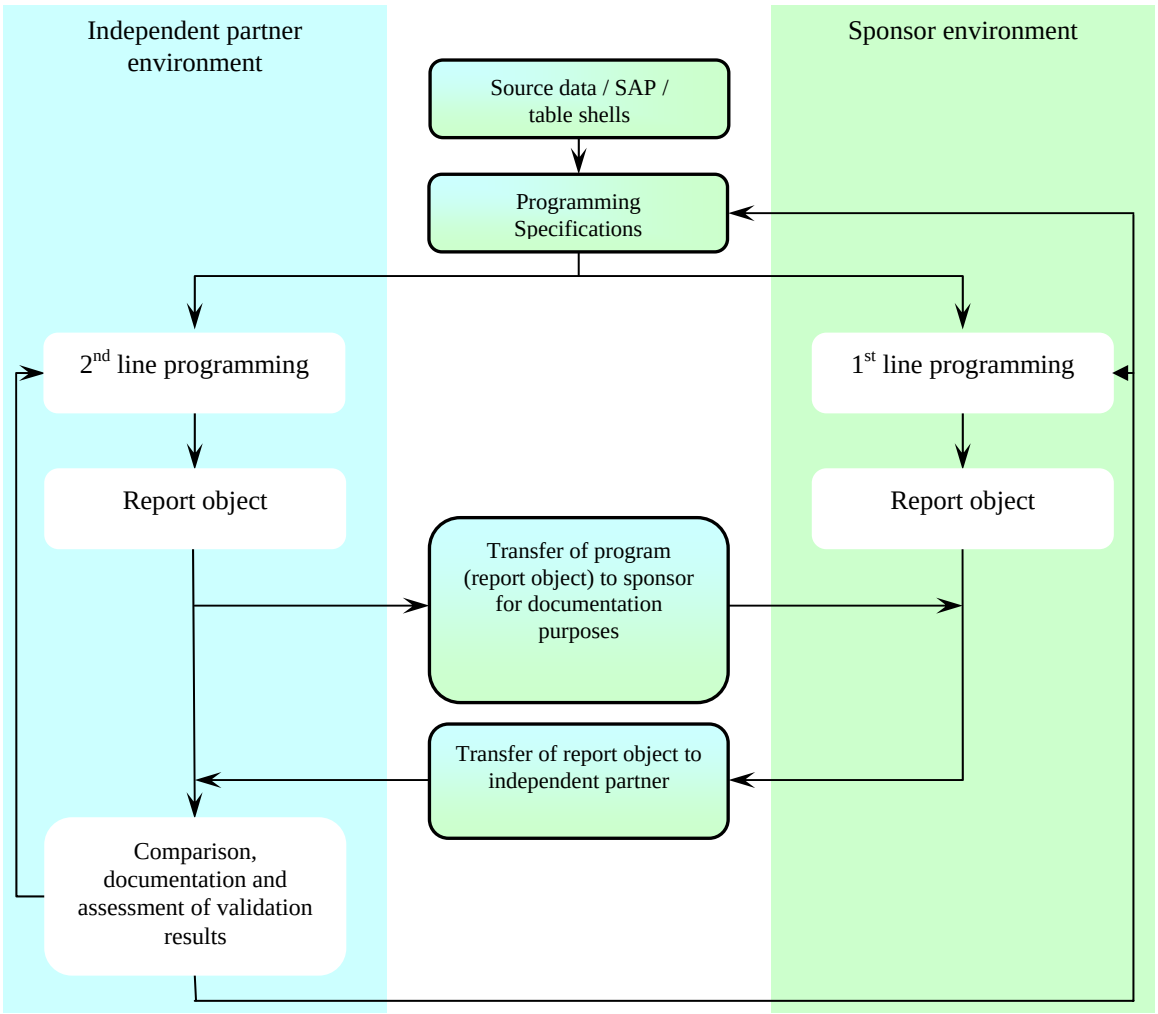
Reprogramming of analysis datasets (ADS) means the 1:1 reproduction of the ADS, i.e. data as well as metadata (variable names, labels and formats) whereas reprogramming of tables usually means the reproduction of the results presented in the table, not the table layout. The latter can easily be compared by eye against the corresponding table shell.

REPORTING ENVIRONMENT AND ACCESS TO 1ST LINE CODE AND RESULTS

In many cases the approach of double coding relies on the qc-programmer's attitude to work independently even if access to the initial programming is technically possible. Therefore approaches involving independent partners facilitate physically independent working environments.

(1) Double programming by an independent partner

Performing double programming in physically different environments and in accordance to a pre-specified process can assure real independency and traceability, as shown below:



(2) Double programming in-house
There are two options for implementing double coding in house, i.e.

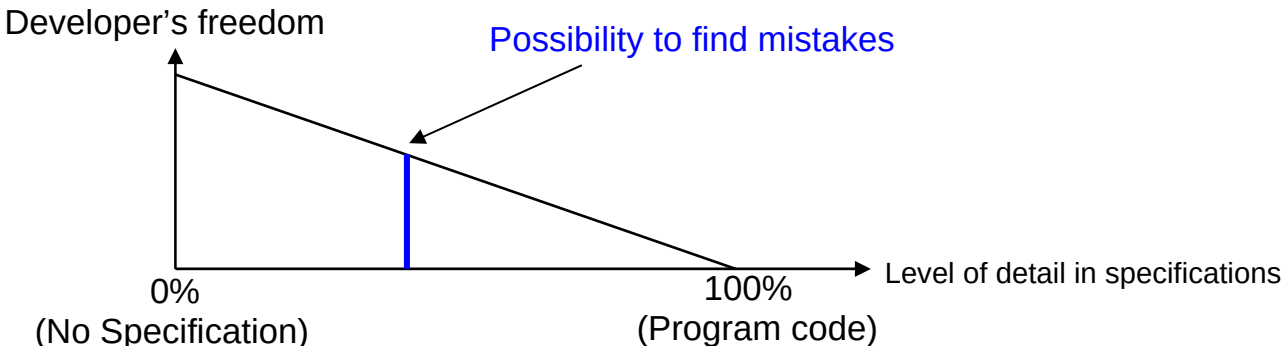
- instruct 1st and 2nd line programmer not to refer to the other programmer's source code
- build a working environment with limited access rights to the source code

Implementing physically independent double coding in-house is usually quite complicated because of access right restrictions in the reporting environment, i.e. two separate programming areas with restricted access rights are required in order to avoid access to the 1st line code and vice versa. Sometimes this is also based on pure trust. However, working in different areas sometimes leads to problems with respect to access to the source data. From both programming environments, access to the same source data should be provided. Ideally there is only one source of data which gets accessed from both reporting areas. This might lead to access conflicts (locked datasets). Report objects generated also need to be accessible by 1st and 2nd line programmer to allow for comparison of the results.

The question is which implementation process of double coding is preferable. Physically independent double coding has the advantage that 1st and 2nd line programmer can neither refer to the other's source code nor the output of the program, i.e. both codes are purely based on the specifications. Therefore issues in the specifications, programming problems and errors can be determined. Two different programmers should finally generate the same result of a report object. But nevertheless, even if access to the other program is theoretically possible, programmers will usually not refer to it but to the respective specifications. Different interpretations of the specifications will be identified, clarified, specified more precisely and the programs get corrected.

DOUBLE CODING AND SPECIFICATIONS

Double coding is usually quite resource- and time-consuming and it mainly relies on the specifications available. Therefore it is necessary that the specifications are in good shape and that they do not change that often. But on what specifications should double coding be based? Technical specifications which are written by the programmer or on the pure statistical analysis plan which gets written by the statistician? For analysis datasets, double coding can only be based on technical specifications. This is because variable names, labels, formats and detailed technical specification rules are usually only described in the technical specifications and not in the statistical analysis plan. It is important that technical specifications get reviewed beforehand and that they get challenged by the 1st and 2nd line programmer, i.e. the programmers should not only rely on the technical specifications but also go back to the statistical analysis plan, the study protocol, the case report form etc. It should not blindly be coded what is written in the technical specifications. Therefore it is recommended to avoid the usage of SAS code for specifications as far as possible. Specifications should constantly be challenged. Also for summary report objects double coding is finally based on technical specifications because all variables and derivations used should be stored in analysis datasets, which are described by technical specifications. To meet the goal of achieving the correct results in the end, it has to be considered whether specifications being too precise and too close to the code itself will not lead to differences in programming and therefore lead to errors; whereas specifications being too vague can result in major differences in the programming results and will be highly time-consuming. Actually, the success is very much dependent on the experience of the programmers. Even double coding can not guarantee 100% correctness.
The diagram below shows dependency of specifications and programming results:



DOUBLE CODING AND UNCLEAN DATA

Double coding usually needs to take place long before final and clean data is available. In many cases time spent for the development of programs is not only dependent on the complexity of the report object but also on the quality and stability of data. Specifications are mainly based on the assumption that data are clean and follow predefined data structures and data ranges. This is often not the case, especially for interim or DMC analyses where results rely on draft data and/or changing data. Often, a lot of time is spent analyzing the data and adopting programs accordingly. In these cases it needs to be considered if double programming is the adequate method to be applied in terms of validation. Double programming on unclean and changing data usually leads to a bunch of issues requiring a significant amount of time and resources wherefore the added value of double coding needs to be put in question.

DOUBLE CODING AND SOURCE CODE REVIEW

In addition to double coding itself the 1st line code should also be checked in terms of programming style, type of code and coding standards in general. Programs need to be changed from time to time and the 1st and 2nd line code should follow the company specific coding standards in order to secure maintenance of the code. Therefore it is important that the 2nd line programmer also checks the source code (code review) of the 1st line program. It is recommended to do this only after successful validation. Besides maintenance issues it has to be kept in mind that programs might need to be delivered, e.g. from CRO to sponsor or from sponsor to the authorities.

AUTOMATED COMPARISON OF REPORT OBJECTS

After successful validation and reconciliation of a report object by double coding, an automated comparison should take place, i.e. the 2nd line program should always be run when new or updated data comes in order to detect possible problems or discrepancies between the 1st and 2nd line code due to new data constellations or unforeseen new data. Ideally the comparison of the report objects should be done at the end of the 2nd line program using PROC COMPARE. It is of advantage to name the 2nd line program as the 1st line program with a suffix (e.g. demo.sas and demo_qc.sas) in order to have both programs next to each other in the company specific version control system.

PROGRAM PROMOTION ON BEHALF OF SOMEBODY

When working in different areas where the 2nd line programmer does not have access to the 1st line programmer's code, program promotion "on behalf of" is necessary. This means: In a program version control system, a 3rd person has to promote the 1st line code on behalf of the 2nd line programmer after successful validation and reconciliation because he or she does not have access to the 1st line code. This needs to be documented with a reference to the validation and reconciliation documentation or log files.

MODULAR ANALYSIS DATASETS

Another aspect of double coding is that specifications (technical and statistical analysis plan) sometimes do not come in all at once and that specifications change from time to time. Analysis datasets contain a certain amount of variables and its specifications might change or do not come in all at once. Therefore a modular analysis dataset structure is recommended, i.e. each variable of an analysis dataset should ideally have its own program or macro. This makes it easier to maintain the 1st and 2nd line code and avoids long "spaghetti" code. When adding a new variable or when changing an existing variable it is ensured that only the affected variable gets touched and that other variables remain untouched. The example below shows how a demography analysis dataset could be build up.

The calling program demo_ads.sas calls the following separate macros (CRTNPT & AGE60) and brings the result variables together, i.e. adds them to the DEMO dataset and creates DEMO_ADS (analysis dataset) according to the key variables protocol, patient number and center number (PROTO, CRTN, PT). Each of the macros creates exactly one variable (CRTNPT & AGE60). It is recommended to have naming conventions for the individual macros for each variable, e.g. demo_ads_crtnpt.sas & demo_ads_age60.sas. The formats, the lengths and the variable labels should be controlled in the calling program demo_ads.sas.

```
%demo_ads(DEMO | DEMO_ADS | PROTO CRTN PT

        | VARIABLE       | LABEL                                  | FORMAT     | LENGTH
        | CRTNPT         | CRTN/Patient Number                    | $11.       | $11
        | AGE60          | Age > 60 (1=YES, 0=NO)                 | 8.         | 8  );
```

PROC REPORT DATASETS

The decision on what to double code (summary report objects, analysis datasets, listing report objects etc) should be based on risk assessment at the planning stage of a reporting event. Analysis datasets are usually of high risk because the included database and derived variables are often used for more than one summary or listing report object. Analysis datasets are the basis of reporting and therefore of high importance. If summary report objects get

double coded, it is recommended that only the content (the numbers) get reproduced, not the entire layout. This is because reproducing the entire layout is resource intensive and has no added value. It is recommended to use so-called PROC REPORT datasets with a pre-defined structure (example see below).

A PROC REPORT dataset is defined as the dataset going into the SAS procedure PROC REPORT in order to display results (e.g. summary statistics) and other variable content. This dataset includes all variables, values and other information which is necessary to produce the actual report object. In case the "OUT=" option is used within PROC REPORT, it is defined as the dataset coming out of PROC REPORT. The advantage of using the "OUT=" option is that actual display errors can be spotted (e.g. if the maximum length of a variable content is 20 characters but the respective column width is only defined with a length equals to 10 (without the FLOW-option)). It is recommended that no calculations should be done within PROC REPORT.

A typical PROC REPORT dataset for summary report objects should consist of the following elements:

a) Order variables: It is necessary to define a certain number of (numeric) order variables. It is recommended to name them **_order1** – **_order<x>**. Order1 is responsible for the so-called "block number" within a summary table. Each summary table consists of 1 – x blocks. Order2 is e.g. responsible for the ordering of the elements within a block. When looking at the example below, "Observation Time (Days)" is block #1 and "Observation Time (Months)" is block #2. In case a third level (or even more) would be necessary, **_order3 (_order<x>)** should be introduced.

Observation Time (Days)			
Mean	628	697	662
SD	309	288	300
Median	596	665	618
Min	1	4	1
Max	1343	1372	1372
n	407	403	810
Observation Time (Months)			
Mean	20.65	22.93	21.78
SD	10.15	9.46	9.87
Median	19.60	21.87	20.32
Min	0.03	0.13	0.03
Max	44.16	45.12	45.12
n	407	403	810

b) Labels: In almost all report objects, there is always one or more "label column" describing the column / block / row content. It is recommended to name this variable as **_obs1**. This variable (character, $200) contains all the labels of the displayed summaries (variables). In case more than one label column is used, it is recommended to name the variables **_obs1 - _obs<x>**. An example (label of a block shown in a separate column / **_obs2** also used as order variable in PROC REPORT – see output below):

Obs	_obs2	_obs1	_NPCT1	_NPCT2	_ORDER1	_ORDER2
1	LIVER*	CLINICAL LYMPHADENOPATHY	130 (32%)	117 (29%)	1	1
2	SPLEEN*	CLINICAL LYMPHADENOPATHY	189 (46%)	198 (49%)	1	2
3	CERVICAL	CLINICAL LYMPHADENOPATHY	321 (79%)	335 (83%)	1	3
4	INGUINAL	CLINICAL LYMPHADENOPATHY	253 (62%)	249 (62%)	1	4
5	AXILLARY	CLINICAL LYMPHADENOPATHY	280 (69%)	294 (73%)	1	5
6	OTHER MANIFESTATIONS	CLINICAL LYMPHADENOPATHY	20 (5%)	17 (4%)	1	6
7	LIVER SIZE**	RADIOLOGICAL LYMPHADENOPATHY	86 (21%)	82 (20%)	2	1
8	SPLEEN SIZE**	RADIOLOGICAL LYMPHADENOPATHY	285 (70%)	292 (72%)	2	2
9	CERVICAL	RADIOLOGICAL LYMPHADENOPATHY	195 (48%)	201 (50%)	2	3
10	AXILLARY	RADIOLOGICAL LYMPHADENOPATHY	224 (55%)	219 (54%)	2	4
11	THORACIC	RADIOLOGICAL LYMPHADENOPATHY	163 (40%)	136 (34%)	2	5
12	ABDOMINAL	RADIOLOGICAL LYMPHADENOPATHY	268 (66%)	258 (64%)	2	6
13	INGUINAL	RADIOLOGICAL LYMPHADENOPATHY	170 (42%)	182 (45%)	2	7
14	OTHER MANIFESTATIONS	RADIOLOGICAL LYMPHADENOPATHY	57 (14%)	52 (13%)	2	8

Below is an example how this dataset would look on the actual output.

CLINICAL LYMPHADENOPATHY	LIVER*	130 (32%)	117 (29%)
	SPLEEN*	189 (46%)	198 (49%)
	CERVICAL	321 (79%)	335 (83%)
	INGUINAL	253 (62%)	249 (62%)
	AXILLARY	280 (69%)	294 (73%)
	OTHER MANIFESTATIONS	20 (5%)	17 (4%)

RADIOLOGICAL LYMPHADENOPATHY	LIVER SIZE**	86 (21%)	82 (20%)
	SPLEEN SIZE**	285 (70%)	292 (72%)
	CERVICAL	195 (48%)	201 (50%)
	AXILLARY	224 (55%)	219 (54%)
	THORACIC	163 (40%)	136 (34%)
	ABDOMINAL	268 (66%)	258 (64%)
	INGUINAL	170 (42%)	182 (45%)
	OTHER MANIFESTATIONS	57 (14%)	52 (13%)

c) (Treatment) columns including the statistics: There are always 1 – x columns showing the numbers, statistics, percentages etc. In a lot of cases, those columns are reflecting the treatments. It is also possible that those columns show other information like cycles or lab parameters etc. It is recommended that those columns are always in character format (in order to be able to do alignment to the decimal point / to add percentages in brackets / to display Min/Max in one row (separated by "/") etc.) and that those variables are named **_npct1 - _npct<x>**. For an example, please see below

A typical PROC REPORT dataset could look as follows:

_obs1	_obs2		_npct1 - x					_order1	_order2
RO0503821 1*/2 weeks	1	Week	190	38.71	14.62	25.00	50.00	90.00	1	1
RO0503821 1*/2 weeks	2	Weeks	190	38.99	14.77	25.00	50.00	90.00	1	2
RO0503821 1*/2 weeks	4	Weeks	190	38.28	14.46	0.00	50.00	90.00	1	3
RO0503821 1*/2 weeks	8	Weeks	187	36.92	16.62	0.00	50.00	112.50	1	4
RO0503821 1*/4 weeks	1	Week	190	38.42	15.00	30.00	50.00	90.00	2	1
RO0503821 1*/4 weeks	2	Weeks	190	38.65	15.19	30.00	50.00	90.00	2	2
RO0503821 1*/4 weeks	4	Weeks	190	39.47	16.13	30.00	50.00	100.00	2	3
RO0503821 1*/4 weeks	8	Weeks	189	38.32	19.67	0.00	50.00	180.00	2	4

The corresponding output looks as follows:

Treatment [ug/Week]			N	Mean	Std	Minimum	Median	Maximum
RO0503821 1*/2 weeks	1	Week	190	38.71	14.62	25.00	50.00	90.00
	2	Weeks	190	38.99	14.77	25.00	50.00	90.00
	4	Weeks	190	38.28	14.46	0.00	50.00	90.00
	8	Weeks	187	36.92	16.62	0.00	50.00	112.50
RO0503821 1*/4 weeks	1	Week	190	38.42	15.00	30.00	50.00	90.00
	2	Weeks	190	38.65	15.19	30.00	50.00	90.00
	4	Weeks	190	39.47	16.13	30.00	50.00	100.00
	8	Weeks	189	38.32	19.67	0.00	50.00	180.00

An automated process, to check the report objects produced by the double coding process, is advantageous and the method of choice. The concept of comparing the datasets feeding into the PROC REPORT syntax is more adept than comparing e.g. *.out or *.lst files.

OTHER CONSIDERATIONS AND THOUGTHS

CHANGE OF REPORT OBJECT RISK – HOW TO CONTROL THIS?

During the conduct of a project, the risk of a report object or a reporting program can change, e.g. a summary report object which had a low risk initially and where standard code review was performed can suddenly become of high risk due to safety issues in a clinical trial. Double coding may now be required. Questions related to the process arise:

- Is there a process in place triggering the review and revision of the risk associated with the report object?
- Is the company specific reporting environment and program version control system able to track changes like this?

Review of the risk assessments at particular time-points should be included in the company specific standard operating procedure and should at least take place before delivery of the final results. It has to be taken into account that higher risk is associated with additional effort, especially as report objects are usually produced using multiple other programs and macros. Therefore the review of the risk assessment needs to take place in a timely manner conformant to the overall schedule.

HOW TO MEASURE QC-METHODS?

Which validation method is the best and most effective one? In general, it is quite difficult to measure the effectiveness and the success of the different qc-methods for report objects and programs. In order to learn for the future and in order to see whether the correct qc-methods were applied a system or a process needs to be in place which checks the amount of errors and which compares this to the applied qc-method. The problem with this approach is that it is not reliable. Double coding (and the other validation methods as well) mainly depends on the quality of the specifications. The success and the quality of Code Review and Extended Code Review are highly dependent on the experience and the level of the qc-programmer, e.g. a very experienced qc-programmer will probably do a more efficient code review than a junior qc-programmer. QC methods can finally only be measured by the number and importance of errors identified in final results. Issues in report events are usually determined by chance only and therefore no reliable assessment of QC methods is available.

HOW TO MEASURE THE QUALITY OF CODE?

When performing Code Review or Extended Code Review the question is how to measure the quality of the code which is reviewed.
Quality of code could be judged by

- the amount of comments in the code?
- the amount of feedback loops with the 1st line programmer?
- the correct results?
- the ability of the code to handle different data scenarios?
- the programming style?
- the time spent for maintenance by another programmer?

It is very difficult to measure the quality of code. Sophisticated instruments would be needed to perform this task. However, this would be quite time consuming and resource intensive.
Another question which comes when performing code review is when is it time to say that a program is ok. In most of the cases this is purely based on the gut feeling of the 2nd line programmer. An experienced qc-programmer might confirm that a program works as expected earlier than an inexperienced qc-programmer who is performing the code review several times in order to be sure that all is fine.

AUTOMATED CHECK OF METADATA

Besides checking the programming results, the 1st and 2nd line programmer should both verify that the metadata also complies with the respective specifications. This can be quite time-consuming and error-prone if length, format, label for dozens of variables need to be checked manually. The implementation of an automated check procedure including consistency checks across panels can facilitate programming as well as the validation process.

THE ROLE OF THE 1ST AND 2ND LINE PROGRAMMER

Though having validation methods in place, the main responsibility to ensure correctness and quality should rest with the 1st line programmer. The role of the 2nd line programmer is to provide added insurance of quality to programs and report objects, regardless of the validation method which is applied. This means, the 1st line programmer should never rely on the 2nd line programmer. It is actually in the nature of statistical 1st line programmers to produce high quality programs and report objects, i.e. not to rely on the 2nd line programmer.

SELF-VALIDATION?

One problem when validating reporting programs in a version control system is so-called "self-validation". This situation might happen, when a program gets promoted by programmer A from the development area to the qc area. Then programmer B should ideally validate the code and promote the code to production. However, when programmer B does, e.g. a further small update of the code he / she checks the program back out to development, updates the code and brings it back to the validation area. If then programmer A promotes the program code to production level, it is so-called self-validation which is not acceptable.

VALIDATION OF REPORT OBJECTS CONSISTING OF MULTIPLE PROGRAMS

Quite often, report objects are produced using multiple (standard) macros and other programs. When double coding a report object which is created using multiple programs, it is recommended to create a stand-alone 2nd line program which is not using other macros and programs. There are two reasons why this is of advantage:

- all programs and macros which are used in the 1st line program to create the report object get (partly) validated as well
- when submitting programs to the FDA, stand-alone programs are requested. The 2nd line code can be used for this purpose once validation is completed successfully.

DATA, PROGRAMS AND OUTPUTS – A TRINITY TO VALIDATE

When validating report objects by Extended Code Review or Double Coding, it is important to always look at the data, the program and the actual output (report object) when performing the validation (besides the underlying specifications). Usually 1st line programs are developed on non-final data during the course of a clinical trial for a certain reporting event. This is also when validation takes place. It is important to re-check report objects on the final data for a reporting event again. In case Double Coding is applied, the 2nd line program should be re-run and the comparison should take place on the final data. In case Extended Code Review takes place, re-checking of the code is recommended as well.

VALIDATION OF SPECIFICATIONS

Programs and the resulting report objects mainly rely on different kinds of specifications, e.g. Statistical Analysis Plan, dataset specifications, table shells, etc. Currently specifications are challenged during the programming and validation period, but formal validation of specifications is still not common practice. During the process of double coding issues in the specifications are usually identified and corrected accordingly. But with the approach of source code review this is often not the case. Therefore it should be considered whether review and approval of specifications is necessary. At least when providing specifications to a third party, e.g. a CRO providing specifications to the sponsor for review and acceptance testing, the implementation of an internal review cycle is strongly recommended.

EVIDENCE OF VALIDATION – NOT DOCUMENTED = NOT DONE?

Validation activities need to be documented in order to have proof that validation was performed.
Generally the following documentation is recommended and should be completed on an ongoing basis

- Tracking of validation steps performed including dates and responsibilities
- Validation findings and actions taken
- Final approval

This information can be collected either manually or within a version tracking system. It has to be kept in mind that validation documentation needs to be available for internal or external audits and might be required by sponsors for archival. The documentation process and the associated document templates are usually described in the company specific SOPs together with checklists that need to be followed during the validation process.

In addition to the documentation mentioned above different forms of evidence are required by different validation methods.

- **Double Coding**: It is recommended to make a reference to the 2nd line qc log and output files when promoting the 1st line program to production within the company specific version control system. Those log and output files should permanently be stored and should not be overwritten. Also a reference to the 2nd line code is strongly recommended.
- **Extended/Standard Code Review**: It is difficult and time-consuming to document what is done during an Extended or Standard Code Review. It is recommended to set up a standard check-/test-list. Additional tests and checks can be documented, if needed. By promoting program code to production in the version control system, the qc-programmer confirms that the standard and pre-defined check-/test-list were followed.

IS VALIDATION BY A QC PROGRAMMER ALWAYS NEEDED?

One common standard approach within the statistical programming departments in the pharmaceutical industry is to apply Standard Code Review for low risk report objects, Extended Code Review for medium report objects and Double Coding for high risk report objects. As mentioned in the chapter 'The role of the 1st and 2nd line programmer', the main responsibility for ensuring correctness and quality of a program should rest with the 1st line programmer. For low risk report objects it is sometimes acceptable to skip validation by a qc programmer. This saves time and resources and puts more responsibility on the 1st line programmer.

FINAL EXECUTION

Validation usually takes place before final data is available, and validation activities can neither be postponed until then, nor repeated at that point in time. Therefore a formal process should be in place for the final execution of the programs.

For final execution all programs need to be final, i.e. production. For this final run it needs at least to be checked, that

- Report objects are generated from correct version of data
- SAS logs document proper run of the programs
- All outputs are complete and created from the final program run
- For programs where a qc program is in place, the results of the respective programs are compared against the results of the initial programs
- For outputs where a validated version on the same data status is available, no changes to the original results appear

TRAINING THROUGH VALIDATION?

The aspect of training associated with any kind of code review should not be underestimated. By reviewing other programmers' code, even experienced programmers identify coding solutions they were not aware of. On the one hand code review should be done be experienced programmers in order to identify possible problems within the code on the other hand it is a good opportunity for younger colleagues to expand their knowledge. If less experienced SAS-programmers do code review for validation purposes the code review should always be accompanied by other validation means, e.g. partial double programming.

CONCLUSION

Validation is crucial with respect to quality of report objects. Therefore validation is one of the most important tasks within each programming department. Efficient and effective validation procedures are required to meet quality expectations while adhering to time and resource constraints at the same time. Due to the high complexity of the analyses being performed and the huge amount of data to be handled error-free reporting events can hardly be achieved, even though processes and standards are constantly improving. There is no validation process in place ensuring 100% correctness of a reporting event. The challenge of identifying the ideal validation method for each single report object remains. Besides the process itself success is mainly dependent on the experience of the personnel involved. Writing of programming specifications, generating of reporting programs as well as quality control mainly relies on the know-how and experience of the programmers involved.

REFERENCES

- FDA Guidance for Industry, Computerized Systems Used in Clinical Trials, Food and Drug Administration, May 2007
- FDA Guidance for Industry Part 11, Electronic records; Electronic signatures – Scope and Application, August 2003
- "A risk based approach applied to the validation of reporting objects" (Christoph Ziegler, PhUSE 2008)
- "Spot the Difference" (Anja Feuerbacher, Beate Hientzsch, Gabi Lückel, PhUSE 2008)
- "Validation of Programs developed Using SAS" (Nikos Tsokanas, PhUSE 2007)
- "Risk based approach to SAS® program validation" (Keith. C. Benze, PharmaSUG 2005)

ACKNOWLEDGMENTS

We would like to thank Juha-Pekka Perttola and Ivan Robinson from Roche, Gabi Lückel and Nicola Tambascia from Accovion for their valuable help and input to this paper and Dr. Christian Müller from Roche and Helmut Sayn from Accovion for supporting this publication.

CONTACT INFORMATION

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

Christoph Ziegler, Dipl.-Dok, Senior Lead Statistical Programmer, QC Process Site SME Basel
F. Hoffmann – La Roche AG
Malzgasse 30, Bldg. 670
4070 Basel
+41 61 68 89200
christoph.ziegler@roche.com
www.roche.com

Beate Hientzsch, Director, Statistical Programming
Accovion GmbH
Helfmann-Park 10
65760 Eschborn, Germany
+49-6196-7709-406
beate.hientzsch@accovion.com
www.accovion.com