PhUSE EU Connect 2018

Paper AD02

Access Rights Management and Version Control in an Independent Programming Environment

Christian Sachara, ICON GmbH, Germany
Anke Grohl, ICON GmbH, Germany
Elena Berdichevskaya, ICON GmbH, Germany

ABSTRACT

This paper illuminates various situations and related options in the context of independent double programming, starting from access rights management to programming environment to source code version control. Independent double programming is the gold standard for the statistical programming industry to ensure the quality of the outputs produced. Is how we work truly independent? In a group of experienced programmers that have worked together for years, problem solving approaches tend to be based on the experience they have made together. Technical solutions are required to control but also help programmers ensure the validity and true independence of their programming. This paper will look at the problem statement of true independence, the requirements within a technical solution deployed by ICON, and the cost associated with implementation.

INTRODUCTION

We all take Independent double programming as a given, however there is more to this than one might see at first glance. In this paper, we will take a look at the aspects and measures that influence independence. These measures will in one direction increase the reliability of the results produced and in the other direction influence the costs of producing the results. The focus of this paper is on factors that may influence independence. This leads us to consider requirements for the environment used for independent double programming and the management of access rights in that environment. In addition, we will take a look at the option to add version control in this setting.

Most likely most of the aspects were in one or the other way already thought through by those who had to design a process and environment for statistical programming. The text is not meant to give the final ultima ratio for all the aspects, but will try to bring a lot of the different ideas, dos and don'ts, and solutions together and encourage those involved to draw their own conclusions which apply to their own situation.

When talking about programs in this paper, one may think of SAS® programs as it is a well-known situation, although the approach holds true for other programming languages too.

PROBLEM STATEMENT

Independent double programming is one in a number of quality control methods to ensure the validity of statistical analyses. It is seen as the gold standard in statistical programming. "The most rigorous and thorough approach to programming validation is double programming in which two distinct programmers independently program an output from the same set of specifications and input data. … to compare output from tables, listings, and figures."[1]

The main feature of the method is the independence of two programs for the same output which is commonly understood as two programmers, a production programmer and a validation programmer, writing their own source code without any direct interaction, e.g. in the invention of algorithms or the interpretation of specifications.

But are we really clear what independence means here?

The best way to assess what independence means is to look at the factors which jeopardize it. The worst case scenario is a real breach of independence where one of the involved programmers copies code from the other one. This is the most serious violation of the method and will result in disciplinary actions in all companies but still occurs due to inappropriate time or cost pressure, lack of knowledge or even the human tendency to make the own life easier. Although this fraud sounds very obvious, it is time consuming and cost-intensive to walk through the numerous programs used in a statistical analysis to identify affected programs and even this is not always reliable as similarity of programs does not only occur if code was copied. Other factors which will make the code look very similar or even identical include the use of the same guidance, exposure to the same training, review of each other one's programs or even talking to each other outside the current setting of a study. A solution widely propagated can also lead to independent implementation of almost identical code and programs written by programmers working together will over time look very similar and may even include the same code snippets and solutions.

The other central part of the independent double programming process is the processing of the detected discrepancies between the production programmer's output and the validation programmer's output, i.e. the resolution of potential faults detected by the validation programmer, is also heavily affected by soft human factors. Communication of mistakes, even potential ones, always creates some reaction in the involved individuals depending on individual behavior and personal relationship, should they be the sender or the recipient of the message or even a third party involved like a programming lead. So, here the approach is again dependent on individuals influenced by the soft factors especially in a setting which is open for direct communication, face-to-face or via phone or email. A typical example of such issue is observed when findings get discussed in multiple bilateral discussions, developer with validator, developer with lead programmer, and so on. This leads to an enormous time consumption and may even create faults in analyses due to different interpretations or specifications must again be consolidated with additional efforts.

With this look at the influence of human behavior and human relationship, the formation of study teams is a factor that should never be underestimated. Unfortunately, it is hard to give recommendations or even rules but it is of value to think through the options like a combination of the old hand and the well-educated novice in a team to overcome issue with well-worn paths and the encouragement of each team member to speak for any concern or new idea. Having said this, whilst the human factor can never be ruled out completely, we have to generate a setting that minimizes the risks and the burden to individuals using clearly defined processes and establishing a technical environment that makes it easy for programmers to follow the processes.

All in all, the gold standard remains a gold standard but only if carefully implemented with the appropriate technical and process setting.

ACCESS RIGHTS MANAGEMENT

The risks to the approach of independent double programming and burden to individuals can be reduced by the introduction of access controls along with clear rules that one can refer to, i.e. a framework with clear regulated processes described in standard operating procedures (SOPs).

PROCESS CONTROL

The basic and common technical setting for an environment in clinical research is a company-wide intranet to which (only) the employees of a company have access. Obviously, global access to all the files in such environment is a too wide and inappropriate setting as not everyone in a company is usually in charge of tasks directly related to a study. Traditionally the very first restriction of the access to the study-specific part of the environment is the limitation of the access to the group of people who are in charge of the processing of the study. We will use this here as the starting point.

Looking at this setting from the perspective of the programmers in charge of the implementation of the statistical analysis, if access is controlled and managed at the study level, each production programmer ("developer") and each validation programmer ("validator") has read/write (r/w) access to each program and output location for his and for the opposite role. Without further regulation this would allow programmers to view and copy code from the opposite role and it goes without saying that this is not the setting for independent double programming. Hence, the very first requirement to introduce independence is a process that prohibits the copying or even the visual inspection of the other roles code (R1).

Having in mind the softer human factors like the personal relationship, a second measure would be to prohibit the direct contact between developer and validator during the implementation and only allowing the documentation of findings and responses in a central repository in writing (R2). To bring the whole situation to a head one could even think of an anonymization approach, so that the programmers would not even know each other when reporting findings (R2+).

In a third step, one could allow an individual programmer to only take the role of a production or a validation programmer in a study across all outputs (R3). This relates to the fact that the programmer is not tempted to engage in the solution development of the opposite role.

These steps lead to a good portion of independence but are also associated with costs. The costs associated with R1 are not subject to discussion as this is a minimum requirement to avoid the damage of a company's reputation. Under R1, an organization could invest in a simple check utility that provides a similarity index when comparing the production and validation source code.

The costs for R2 are related to the strength of the requirements introduced. Using an open system like an Excel® workbook accessible by any programmer leads to low costs but is also only a soft measure as it open for manipulation, mistakenly or fraudulently. A closed system like a database with an audit trail to log any actions regarding the clarification creates much higher costs but is also a more reliable and audit-stable tool. Establishing a system however, does not limit interactions outside of the system, so a further enhancement might be to implement anonymization within the database system (R2+). However, this may require a physical separation of the involved programmers and impacts the sense of team work – not a recommendable solution.

The costs of R3 are related to the number of programmers needed to ensure that in each role at least one individual is available at any time of the analysis, flexibility in assignments of programmers, plus the additional administrative cost just to ensure adherence. The latter arise from the fact that each absence needs to be compensated without violation of the rule and with availability of programmers with the appropriate skill set which may become difficult for unplanned absences.

If no technical measures are introduced, the regulations are dependent on the good conduct of the individuals working on the study and there is lack of evidence of independence for inspections and audits. This in the end means that costs may be lower for the technical solution but higher from the need of controlling the process and creation of evidence of the validity. So, the translation of the process regulation into a technical implementation would be more efficient.

INDEPENDENT ENVIRONMENT

The roles and rules in the process, lead to the segmentation of the environment and a graduation of the access rights.

The element of an appropriate environment would be the

- program location for the developers (DEV-prog),
- output location for the developers (DEV-out),
- program location for the validators (VAL-prog) and
- output location for the validators (VAL-out).

The program location for each of the roles contains all the programs and program related files that are used to generate outputs for an analysis, i.e. essentially the programs themselves including any macros, and also the log files and any files that programs are technically dependent on. The output location of the developer contains the outputs as defined in the specifications plus any supportive files that have analog contents, e.g. output datasets including the contents of associated table. The output location of the validator contains the results of the validation process, i.e. any report of differences when comparing the production output to a validation output, and may also include outputs identical or similar to the production output for visual inspection. For the developer the output location must not include any program or program related files, and similarly nor for the validator.

Further parts of the environment are the input location which contains all common input files including the input data that are not subject to validation, i.e. used by developer and validator as they are, as well as any standard tools also used as they are. These are not subject to any further discussion here but read-only access to both locations is assumed for both roles.

The required access rights in this setting are

- r/w access to DEV-prog for the developers,
- r/w access to DEV-out for the developers,
- r/w access to VAL-prog for the validators,
- r/w access to VAL-out for the validators,
- read-only access to DEV-out for the validators and
- read-only access to VAL-out for the developers.

Figure 1 shows the schema for this setting.

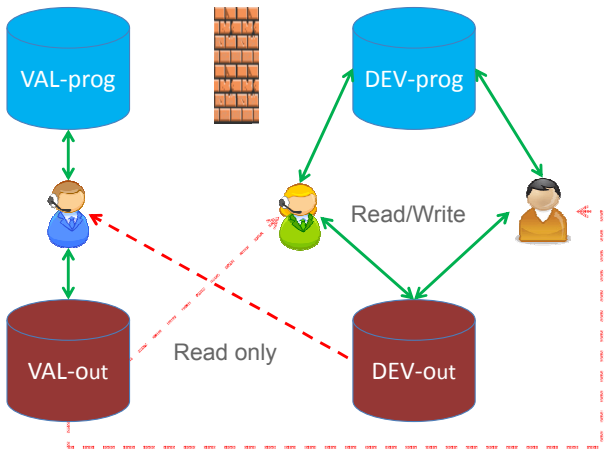

Figure 1: Environment without version control

The read-only access for the developers to the validators' outputs is not required if other ways of communication of findings are in place. This access is of value in the day-to-day work as the validation outputs can be used as clear references and it should not harm any of the requirements. However, one may think about the question if it could create pressure on the developer to accommodate the output of the validator.
Combined with the process rule that an individual programmer can either be developer or validator in a study across all outputs (which even could be implemented technically), this ensures that source code cannot be copied based on an individual's decision, i.e. a validator will never be able to access the source code of the developer or vice versa without the support of the other role.

VERSION CONTROL

The term version control introduced here refers more precisely to source code version control in the context of statistical programming. It means the tracking of changes of program source code and related files and can easily be introduced with an extension of the independent environment.

The version control approach here follows a Copy-Modify-Merge model and the program location described above is changed to two instances.

One instance is a private file location, the working copy, exclusively accessible for the individual programmer (Name-prog). In this location programs are developed, executed and stored together with their log-files and any other files they are dependent on. The other instance, the repository (DEV-repo or VAL-repo), includes a tracked copy of these files upon committing it to the repository. The repository being under version control, logs each and every change of the committed file with date, time and the programmer identification. In such a setting a work cycle for programmer A consists of the creation of a working copy in A-prog from the repository associated to his role, modification of existing programs, development of new ones and execution of programs and concluding the work cycle with a commit back to the appropriate repository.

This does not only strengthen the independence up to the point where two individuals are needed to bypass the technical control but also allows to track the steps of program implementation, a fall back to previous program versions and traceability of the programs source code and log-files. Figure 2 shows the schema for this extended setting.

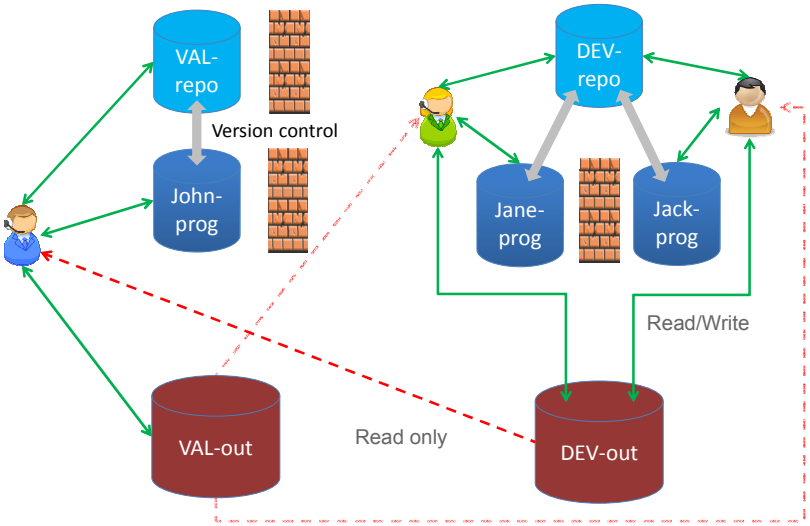

Figure 2: Environment with version control

For the reader who is familiar with version control, we have not chosen the Lock-Modify-Unlock approach due to the risk of locked files that need to be unlocked by a third party if programmers forget to unlock after modification and become unavailable. The associated risk and costs were estimated higher than those associated with the loss of modifications if programmers forget to commit and become unavailable or the efforts for the resolution of discrepant updates which may occur if different programmers in the same role update the same file concurrently. The latter may at first glance look like the most likely case but in the usual setting of exactly one programmer being responsible for a program it rarely occurs.

PRACTICAL IMPLEMENTATION AND EXPERIENCE

Our own experience at ICON, led to the development of the so called 'Data Governance Portal' (DGP) which is a self-service application using a database to manage individuals, their roles and their membership to access groups for individual studies.

In our initial version of the DGP, upon request and approval by a manager or initiated by a manager, created in the setup step the folder structure for a study on the file server and the different access groups (see Figure 1) without read-only access restriction to the output areas. In the next generation DGP, it creates the folder structure on the file server, the repositories and the different access groups (see Figure 2). In both scenarios the system sends an email to the assigned data owner informing him about the assignment so that he is aware of his responsibility. The only right granted in this step for the data owner is the right to approve or reject access requests, i.e. assign access group membership for programmers upon request.

Upon assignment to a study team a programmer requests access to the standard folder containing all relevant study documentation like statistical analysis plan, output templates and specifications. This request triggers an email to the data owner for approval and with this notifies him about the new team member. Before granting access to the environment for production or validation, a written confirmation of the appropriate knowledge on the study is required from the programmers. Upon this confirmation they can place an access request for their respective role in the DGP. This request triggers an email to the data owner who will approve the request if all prerequisites are met or rejects if they are not met. The approval grants the respective access rights to requesting programmer and again triggers an email that confirms the approval to the requester. The rejection of the request triggers an email that notifies the requester about the rejection. This concludes the process of access controls.

The role of the data owner here shows that this process, as with all processes, needs a controller. The controller in this instance is the lead programmer and hence shows the limitation of the approach.

Beyond all the optimization regarding independence, other requirements in statistical programming need to be addressed as well, and may even be hampered by the setting. These requirements include but are not limited to the confirmation of error free execution of programs (clean logs is a typically a validation check), the well-formation of the source code and appropriate commenting of the source (also a validation check within some organizations). Primarily, the individual programmers are responsible for checking these requirements but mainly due to the same human factors as described for the need of the independence a further quality control step is needed. At ICON, this is hence performed by the lead programmer checks as an independent confirmation of the requirements mentioned above are common and showed as required in the past. For our final setting the lead programmer role can be achieved by allowing an individual in this role to have the rights of a developer and a validator but one may also think about an evolution of the approach creating this a separate role with read-only access to both parts of the environment. Even the very tricky situation of assigning the role to a pair of programmers, one developer and one validator, may be thought through.

CONCLUSION

Whilst independent double programming is seen as the gold standard for statistical programming it still needs an appropriate environment that allows programmers to be as independent as possible to achieve the ultimate goal of 100% error free statistical analysis. The concept of a segmented independent environment allowing all required access but prohibiting unnecessary access in combination with individual working copies and a version control system as well as a tracking and communication tool for findings leads to a sophisticated approach to reduce the risk of mistakes and even fraud.

Beyond all technical solutions, the formation of a programming team considering the combination of the old hand and the well-educated novice, the time needed for implementation and the personal relationship established in a department is indeed an important success factor.

REFERENCES

[1] "Quality Control Programming: A Lost Art?", A. Randall and W. Coar, SAS Global Forum 2017

RECOMMENDED READING

"Version Control with Subversion", B. Fitzpatrick, C. Pilato and B. Collins-Sussman, http://svnbook.red-bean.com

CONTACT INFORMATION

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

Christian Sachara
ICON GmbH, Germany
Robert-Perthel-Str. 77a
50739 Köln, Germany
Work Phone: +49 221 5999 210
Email: Christian.Sachara@iconplc.com
Web: www.ICONplc.com