Paper SM03

Optimizing Workflow and Collaboration in a Pooled Coding Environment for Clinical Development

Louise S. Noergaard, Novo Nordisk, Aalborg, Denmark

ABSTRACT

Managing multiple phase 3 programs in parallel is challenging and requires considerable time and dedicated resources. The conventional approach of working within study "silos" often requires substantial post-conduct effort to align data packages for submissions. We present an innovative approach implemented by Novo Nordisk to manage multiple phase 3 programs by working in a pooled coding environment. We describe how using R and Git allows us to efficiently handle both generic and study-specific coding in one place during development. The code is then later "sliced" into individual study containers, ensuring that technical programming decisions are consistent across studies. We also present our implementation of Azure DevOps to ease collaboration, distribute tasks, track progress and much more. Finally, we describe our implementation of Git for continuous version control through pull-requests and automated code-functionality checks. The goal for our setup is to optimize programming resources, ensure trial alignment, and facilitate faster submissions.

INTRODUCTION

Traditionally, clinical studies are handled individually, meaning that one trial programmer takes the main responsibility for study-related programming tasks in collaboration with support programmers and the trial squad as a whole. This team then develops all the necessary code for all deliverables, including ADaM packages, ADRG, as well as tables, listings and figures (TLF).

This conventional approach has shown to be very time consuming, not only during study conduct, but also when studies are prepared for authority submissions. Preparation of studies for submissions includes pooling data from multiple studies, in order to draw general conclusions regarding efficacy and safety of the drug under investigation. The pooling process requires a big team with dedicated people. The team will work on the pooling strategy to ensure potential differences in data is combined in a meaningful way. The team will also implement a suitable level of alignment in programming rules between data from the different studies. After combining the relevant data into one database, this dataset can be used to program pooled deliverables such as Integrated Summary of Safety (ISS) and Integrated Summary of Efficacy (ISE).

One of the problems with this conventional "after-burner" approach, is that the process is heavily time-consuming, technical, and it is hard if not impossible to frontload any deliverables. Trawling through data decisions made on individual studies can be quite challenging and coding decisions implemented on individual studies may even impact decisions made in the data pooling process, and thus impact the variables available for ISS & ISE deliverables.

In this paper we present a new, unconventional approach to statistical programming in large phase 3 programs. Our approach includes several key components, both in the programming setup, but also innovative solutions to the way the team collaborates, as well as the tools implemented for version control and planning. Notably, planning is handled using one platform, but at multiple levels: both on the day-to-day level, and on the higher submission-related milestone level.

The idea behind working in a pooled coding environment is that code is streamlined between studies, and only differs in sections where trial design or data collection requires differential coding approaches. The pooling of data during trial conduct and the streamlined program development within the pool, allow for early initiation of submission tasks such as ISS, ISE, and country specific submission packages, which requires pooled datasets. This approach also reduces workload on the individual trials by reducing the amount of duplicated base code written and ensures that code is only maintained in one place. When working in the pool, code is only written once, but in such a way that it considers the requirements on individual studies. In this way, all programs are created in one place (e.g., within the pool), and one

programmer creates this program across all studies in the pool, at once. Code is later sliced into individual study containers.

Among many benefit, the major advantages obtained by implementing this unconventional “pooled coding environment” approach, combined with new ways of collaborating, are:

- Ensures alignment in standards used in multiple phase 3 studies going into the submission.
- Quick identification of misalignment between studies.
- Efficient identification of data issues which can be addressed for multiple studies at ones, as required.
- Streamlined code across studies.
- Aligned code by design.
- No duplicated code written in individual trial containers (i.e. fewer lines of code written overall).
- Code maintenance in one place.
- Early pooling of data with the ability to frontload ISS, ISE and country specific submission packages.
- Improved collaboration and tracking of programming and review status.
- Strengthened team spirit.

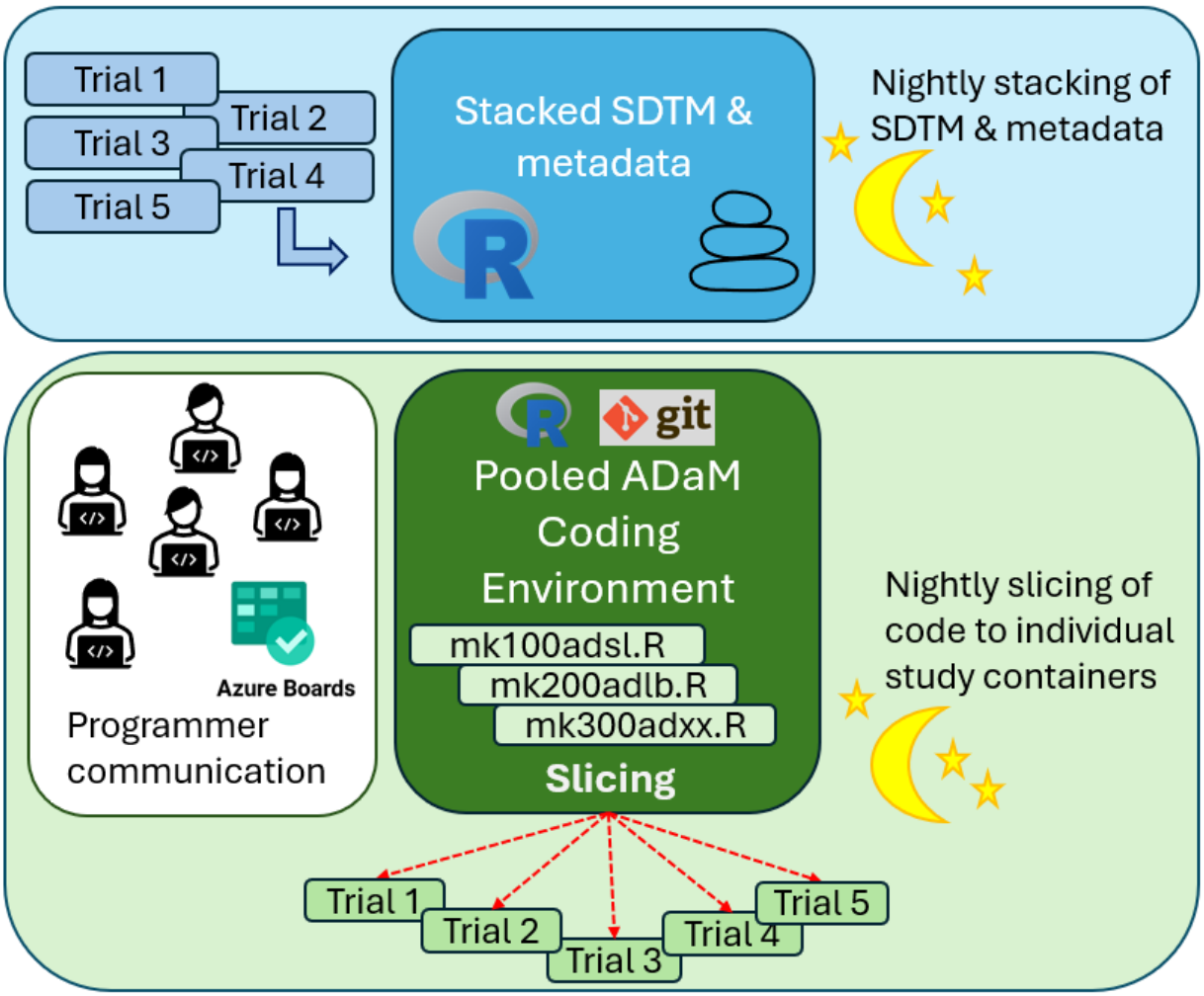

Figure 1 An overview of the workflow when working within a pooled coding environment. Blue colors represent SDTM data and metadata on individual trials and stacking of SDTM & metadata. The stacking procedure is performed every night, to ensure fresh data. Green colors represent the ADaM programming. ADaM programs are developed based upon the stacked SDTM data, and programs are then sliced into individual studies on a nightly basis. Furthermore, Azure DevOps Boards is implemented for communication, planning and overview of programming progress and resource allocation. Git is implemented for version control of all programs, and R is used both for developing code, stacking of SDTM, as well as slicing of ADaM code from the pool into individual study containers.

POOL PROGRAMMING IN A NUTSHELL

The pooled coding environment is built by first stacking relevant metadata, RAW and SDTM data from all studies. This initial stacking of SDTM data, is of course essential for any downstream programming on pooled data sources. Thus, the decision about working in a pooled coding environment must be established early in the planning process. It is, for example, a requirement that all studies run on the same SDTM IG version and use the majority of the same variables. Studies can of course differ in the types of data collected, as some studies might even be collecting special types of data such as Continuous Glucose monitoring for T2D patients, or Dexa scans for particular endpoints. Such differences can easily be accounted for in the pool setup, especially when considered well in advance during the planning process.

When studies intended for pooling use different variables in SDTM, then the pooling becomes increasingly more complex and requires additional hardcoding. It is therefore essential to determine at an early stage, which studies should be part of the pool, and ensure the best possible alignment between studies. In the same way, should it be decided for which studies the pooling does not make sense, potentially because studies use different SDTM versions, or simply because data collection, objectives, and trial design, do not allow for logical pooling with other phase 3 trials in the portfolio.

The stacking of SDTM data is performed using a custom build R program placed within the pool instance. This program is (like most other programs, see below), set up to automatically run every night, to ensure that all programmers are working on the latest SDTM data at any point in time. This nightly run is set up on a RStudio Connect server, which is a publishing platform where users can share and publish tools created in R (or Phyton). If any execution errors occur in the nightly SDTM stacking process, a notification will be available, and the programmers can further investigate and fix the bug.

Among the many advantages of the initial stacking of SDTM data, lies the opportunity to address data issues for all studies, at once. In this way, pooling forces alignment in standards and misalignment between individual studies are quickly detected when working within the pool. In our setup, close collaboration between SDTM programmers allocated to individual studies were pivotal, and their coordination of data resolution updates was essential to the functionality of the stacked SDTM data.

The following sections of this paper dives into the downstream workflow and highlight how each component of our ‘way of working’ contributes to more efficient and seamless programming on multiple large phase 3 trials.

ADAM PROGRAMMING

The development of ADaM programs within a pooled coding environment offers several advantages. Firstly, it enhances efficiency by allowing programmers to ‘reuse’ code across multiple studies, removing the need for duplicated efforts. This not only saves time, but also ensures consistency in the implementation of analysis datasets. Additionally, project functions is established to ensure alignment in implementation rules, further streamlining the process and maintaining uniformity across different studies. By leveraging a pooled coding environment, our team can focus on refining and optimizing code, leading to more robust and reliable analysis results. This approach contributes to a more efficient and cohesive workflow, benefiting the overall project and its outcomes.

Using our approach does, however also allow for implementation of different coding approaches and rules when required on an individual study. In such scenario, code chunks that are only relevant to one or more specific studies can be included in the pool program, by adding a specific study ‘tag’ in the code (see below, tagging and slicing).

TAGGING AND SLICING

Moving code from the pooled coding environment to the individual study containers is facilitated using a custom-built R program that segments / slices codes from the pool into the individual study containers. Code chunks that are only required in one or more studies can be “tagged” within the code, and thus only included in the specified study. All other parts of the code will be transferred to trial containers for all studies included in the pool. Figure 2 demonstrates the use of tagging of specific code chunks that should only be included in specified studies (here, study 1234 requires inclusion of DIABTYP and DIABDUR, whereas study 4567 requires HBA1CBL).

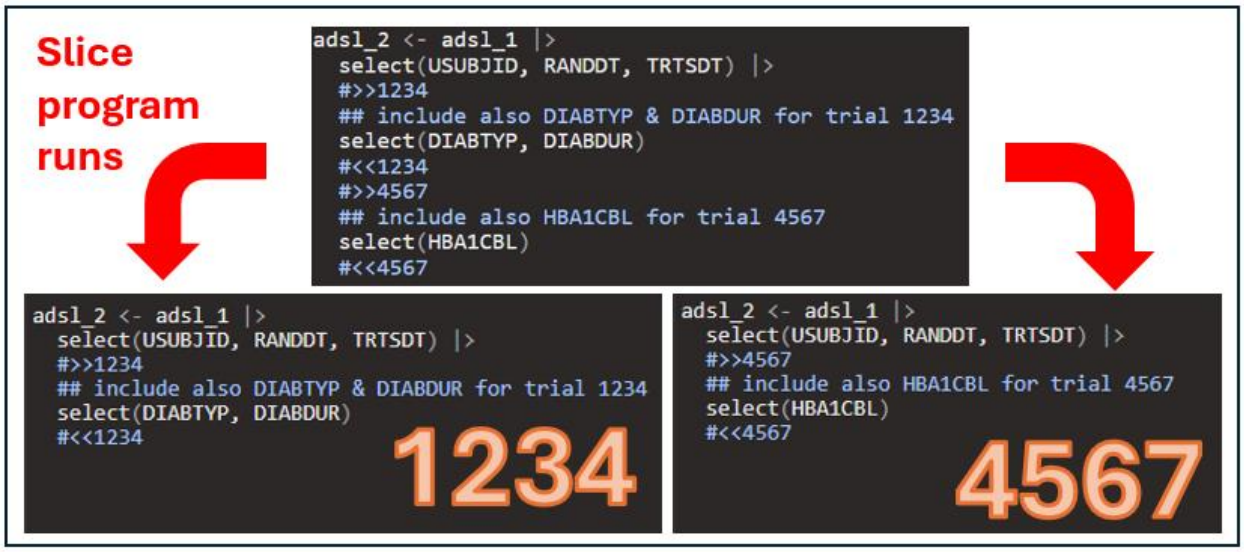

Figure 2 The use of code tagging in the pooled coding environment, allowing for inclusion of study specific code chunks at the trial level.

GIT VERSION CONTROL

In order for our pooled coding approach to fully become a successful way of working, then great collaboration, program history, and time management are all key ingredients. To maintain a seamless and collaborative code development environment, we utilize Git for version control for all programs in the pool. This allows for efficient tracking of changes, easy collaboration among team members, and the ability to easily revert to previous versions of a program, as necessary. Git version control was implemented in the pool (as well as on individual studies), to ensure that no work is lost, damaged or otherwise erroneous. Every time an update to a program is made, then the programmer is prompted to pull code from the repository to ensure that the local working environment is "up to date". Hereafter the programmer can "commit" i.e. locally save the changes made to his/her program in the local environment. Next, when using the Git command "push", then these changes are submitted to the project repository and will appear in the Azure DevOps Repository shortly after. Later, these changes will be pushed from the online repository to the company drive.

In case another programmer happens to have made changes to the same program in the meantime, a conflict will appear in the system. Using the Git functionality, programmers can easily detect the changes made by colleagues, and in collaboration determine how to solve the code conflicts (i.e. determine which part of the code should be kept in the main program) (see Figure 3).

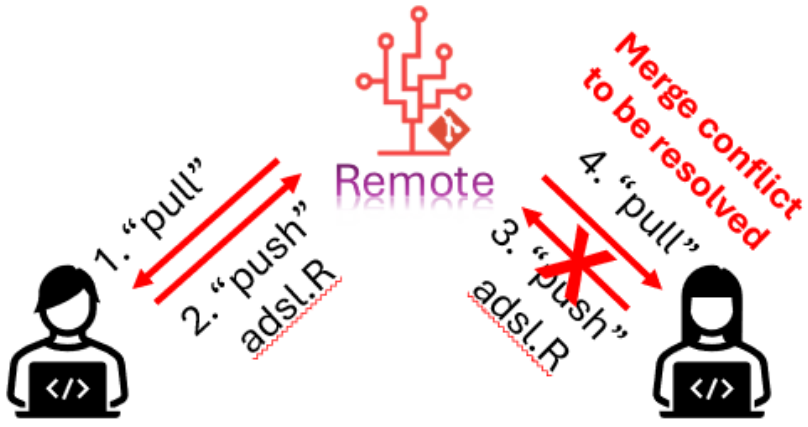

Figure 3 The Git workflow. By using "pull" and "push" commands, programmers can submit changes made to a program into the project repository. In case two or more programmers by mistake worked in the same program, at the same time, then a merge conflict will appear. Upon a merge conflict, the programmers must resolve the conflict in collaboration. Conflict resolution is made easy by using the git functionality to quickly point out the differences between versions.

Finally, we also version control the R packages included in the pooled coding environment. Specifically, a renv.lock was created defining all R packages needed for program execution. If a programmer, during program development, need to use a package not previously included in the renv.lock, then an update to the renv.lock should be performed. After the update, all fellow programmers should be notified, and update their local working environment to align with the new version of the renv.lock in the pool.

RSTUDIO CONNECT JOBS FOR FRESH DATA EVERY SINGLE DAY

To ensure that all colleagues are working on the latest data every day (and to ensure that programs in the pool run without errors), we set up a so called “night run” on a RStudio Connect server. Specifically, we set up the R slicing program to run every night after normal work hours (as much as possible due to different time zones within the team). Allowing the slice program to run every night ensures that any updates made to programs during a working day, are distributed into individual study containers on a daily basis. The presence of updated code in the pool, but particularly also in individual study containers, ensures that testing of ADaM programs as well as output program functionality can be performed at the study level on an ongoing basis. Every morning, an appointed programmer will check the night run logs for any errors, and if any errors or unexpected warnings has been logged during the night run, then these will be addressed as the first task of the day.

Next, for the pool environment (and for all individual studies), we also set up a night run job which executed all programs including metadata programs, ADaM programs, and output programs. This meant that each morning, when colleagues returned to work, the latest data was also present both in the pooled coding environment, but also in the individual studies. In this way, team members are always working in an “up to date” environment, where any changes made – for example to an ADaM program - is reflected in the data available in the study folder and can be used for TLF programming and analysis purposes.

Importantly, any changes that may be required at the study level (such as a change to a variable, addition of a parameter etc.), should be performed in the pooled coding environment, so that changes made are not overwritten at the study level during the nightly slice job. Figure 4 depicts this workflow, where an update is required at the study level. The update must be performed and implemented in the pool environment, and later sliced from pool into the individual study folder.

Figure 4 The workflow when updates are required at the study level. Updates are performed in the pooled coding environment and later sliced into the study folder during the night slice, performed on the Rstudio Connect server.

TOP CLASS COLLABORATION WHEN WORKING IN A POOL

Whilst considering how to best work in a pooled coding environment while also maintaining trial-level activities, it is essential to explore the best suited tools for internal communication and not the least for planning of all study related tasks.

After exploring a number of different solutions, our team found that the use of Azure DevOps for planning and communication was a brilliant fit. In our studies, a master document that included a list of all deliverables to be programmed (both in the pool and for each individual study) was set up. The content from these master documents was uploaded to a an Azure board, which was accessible to all team members.

All tasks on the board had a main programmer and a reviewer associated with it. Figure 5 demonstrates a typical setup of an Azure board, where each program is set up as a task, with information about the task, the responsible programmer's initials, the reviewer's initials and the study to which the task belongs.

In this way, all team members can easily get an overview of all tasks and track the progress for each individual task. For example, in Figure 5 the program mk400adeg.sas is located in the column "in progress". This task should then be shifted to the column "Draft Ready" or "Ready for review" as applicable during code development. This makes it easy for the whole team to get an overview of the status of individual tasks, but also allows the reviewer to know when he or she can start the review process of a given program.

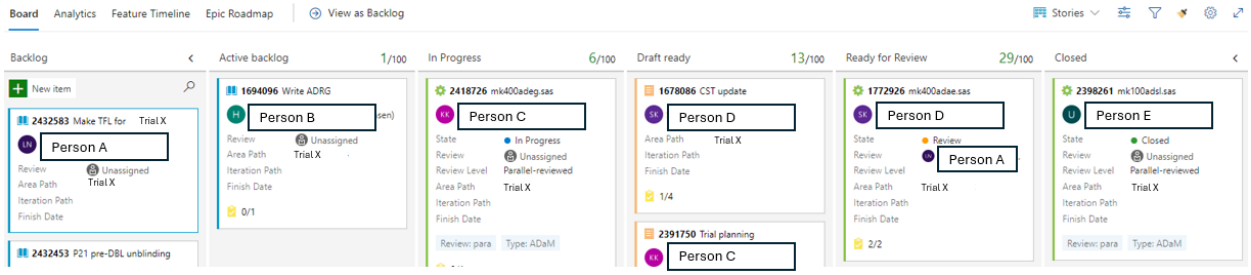

Figure 5 A typical view of programming tasks on a study. Each task contains information about who is the responsible programmer and reviewer (when applicable).

The Azure DevOps board is also used to keep track of review comments both from parallel and peer-review processes. Within a task, the programmer and reviewer can add comments, bugs etc. to allow for transparency on everything that has been discussed regarding a given program throughout the study (see Figure 6). This also ensures that all issues raised are resolved prior to shifting the task to the "Closed" column.

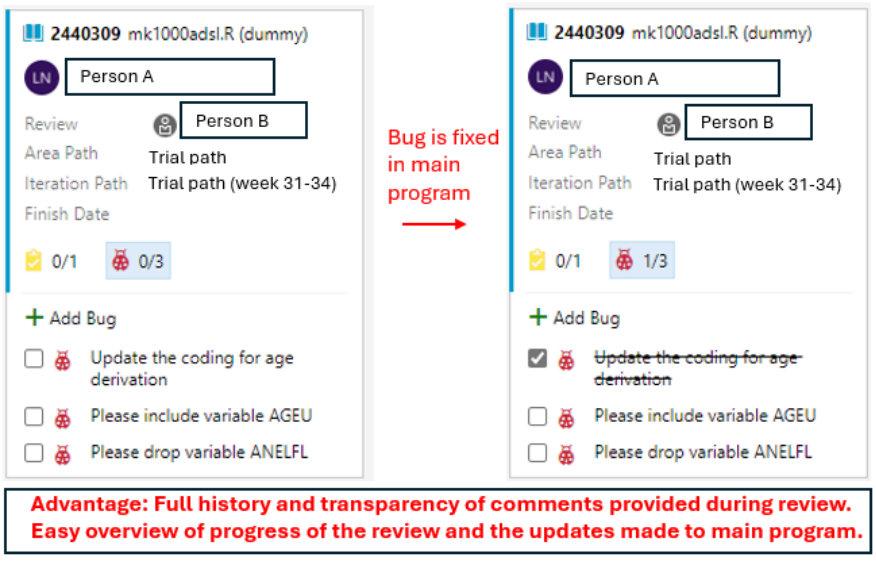

Figure 6 Easy overview of review comments, review progress and any discussions that has taken place during program development.

To maintain close collaboration among team members, the team also inherited parts of the “sprint” workflow, originally developed in the software development industry. To fit its purpose in clinical programming, we tested a number of different durations of sprints. In our group, we found that a 4-week sprint was suitable during the main period of study conduct. This allowed enough time for programmers to get a substantial amount of work performed within each sprint. Furthermore, the team did not spend more time than considered suitable for planning purposes for the duration of a sprint.

For each sprint, a programmer must plan the work they intend (and find suitable) to perform during the sprint. This involves identifying tasks, estimating the time required for each task, and prioritizing them based on their importance and dependencies. By doing so, the programmer ensures that they have a clear roadmap for each sprint, allowing them to stay focused and organized while meeting the sprint goals and deadlines. Practically, the tasks in scope for the upcoming sprint, is allocated to the specific sprint cycle within the Azure DevOps board. When doing so, the tasks appear in the sprint view. In the sprint view, individual sub-tasks (under a main task) can be shifted relative to the progress of that task during the sprint (see Figure 7).

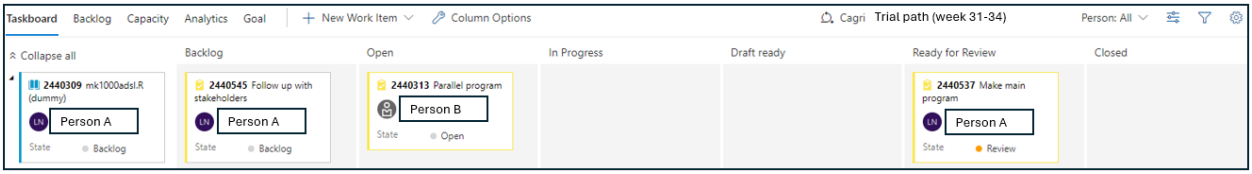

Figure 7 Sprint view for three sub-tasks within a main program. The tasks presented in sprint view were determined to be in scope for the given sprint by the individual programmer, and can be moved to the appropriate state as programming progresses throughout the sprint.

During a sprint, the team would have two weekly “stand up meetings” which consist of a 15 minutes catch-up between all team members. On these meetings, any issues encountered could be discussed. Topics relevant to other team members was also to be raised, and these meetings were a fantastic opportunity for quick knowledge sharing and efficient updates delivered to the whole team at once. This implementation of weekly sprints was a great way to ensure that all team members were “up to date” without taking too much of all colleagues’ time. The stand ups also allowed the team to get to know each other even better, whilst some team members were working remote and others on site.

Finally, the Azure boards also served an additional important purpose during milestone planning. For this purpose, we set up an additional Azure board used specifically to determine internal deadlines, relative to official deadlines within the project. This planning included all studies within the project; i.e. deadlines for Phase 1, 2 and 3, as well as overall submission deadlines for the project. This board thus served as a great source for project planning, and overall priority management within the project team.

CHALLENGES ENCOUNTERED AND OVERCOME

During the establishment and setup of the pooled coding environment, several challenges were of course encountered, but also successfully addressed. First, the team consisted of colleagues who were all eager to make the approach work, and willing to explore the benefits, potentially to be archived. The team spirit and innovative mindset of all team members was pivotal for the implementation and execution of such a big pooling project.

The team had to adapt to a new way of working, which involved building skills within all the technical tools, including connect jobs, slicing program and building confidentiality with the Azure DevOps board. All team members had to build an understanding of how to efficiently use the tools available, for review and communication. The implementation of Git for version control played a crucial role, but likewise required all team members to understand this workflow. Initially, some instances of merge conflicts did appear, but solving these merge conflicts together made all team members more familiar with the workflow. As programmers became more comfortable with the process, they were able to handle these conflicts more efficiently. The team also focused on leveraging the tools to streamline their workflow and ensure alignment with implementation rules, leading to a more cohesive and productive working environment.

CONCLUSION

Working in a pooled coding environment and leveraging suitable software solutions to facilitate efficient programming and collaboration has truly increased performance and resulted in reduced redundancy and more efficient allocation of valuable time and resources. The use of R software for slicing code, combined with RStudio Connect night jobs for code execution, enables all team members to work on fresh data every day.

This approach facilitates seamless programming of all study activities, including efficient development of summary documents such as ISS and ISE. With the success of this experiment, many projects are now adopting the pool programming approach, prioritizing pooled environments over the conventional approach of coding in individual study containers.

DISCLAIMER

The views and opinions expressed in this presentation belong solely to the presenter and do not necessarily reflect those of the presenter's employer, organization, committee, or any other group or individual. The purpose of this presentation is to highlight the experience gained in one of our projects. No data or confidential information belonging to Novo Nordisk has been utilized in this presentation.

ACKNOWLEDGMENTS

I would like to thank all the great colleagues in my team, and in Novo Nordisk, for their contribution to making this project possible, and supporting the implementation of innovative solutions.

CONTACT INFORMATION

Your comments and questions are valued and encouraged.

Contact the author at:

Author Name: Louise Solveig Nørgaard
Company: Novo Nordisk
Address: Alfred Nobels Vej 27, 9220 Aalborg Øst, Denmark
Work Phone: +45 34 48 18 06
Email: lnqg@novo.nordisk.com
Web: https://www.novonordisk.dk/