Paper AD11

Automated review of metadata standards, data checks, and trend visualization.

Giuseppe Di Monaco, UCB BioSciences GmbH, Monheim, Germany

ABSTRACT

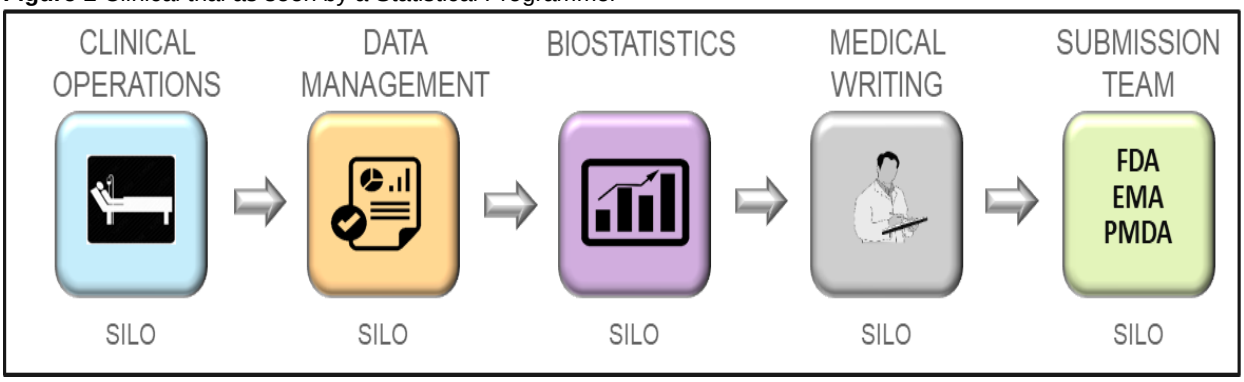
Statistical programmers are rarely involved in a large automated software project where multiple groups, departments, and various technologies are involved. This paper presents the steps taken toward creating a large, automated project using SAS®, Qlik Sense® and Microsoft® SharePoint together with managing the project version releases. The objective is to support the creation of metadata standards, data checks and visualization of data quality trends. The paper also describes the initial modeling, requirements and automated validation of the generalized modules which are at the core of the project. The design eases maintenance and facilitates future development while allowing for flexibility in study-specific data and quality checks. Data Problems are surfaced directly to the consumer using targeted, time-saving, automated reporting, and these reports are loaded into Qlik Sense® for trend detection visualization. Finally, all communications are managed centrally by a component that identifies and notifies consumers that data reports and visualization are available.

INTRODUCTION

INNOVATION IN CLINICAL TRIALS

Many processes in clinical trials are resource-intensive, relying heavily on highly trained professionals who are frequently engaged in repetitive manual tasks for generating of protocols, statistical analysis and data management plans, metadata documents, programs for datasets, programs for TFLs generation, data review, safety data processing, quality control of regulatory submissions and many other documents and tasks. On top of the resource-intensive processes comes the metadata management, which cannot be automatically incorporated because of the differences in technologies and processes. This facilitates the creation of silos, which in turn increases the number of inefficiencies and risks of duplicating the work across departments. **Figure 1** shows silos across groups and areas of a clinical trial seen by Statistical Programmers.

Figure 1 Clinical trial as seen by a Statistical Programmer



Because of resource-intensive process and silos, the overall cost of bringing a single drug to the market is very high and continues to rise over time. To tackle this problem, UCB launched an initiative which promotes Process Automation in the areas of data standards, data access, data analytics & visualization. Within this scenario, our team is working on a large project on which different technologies and components are designed and integrated into a tailored automated process for data review, metadata collection and standardization, standard data checks, trend visualization and notifications.

BACKGROUND

The initial modeling of this project was already implemented and presented to previous PhUSE conference[1]. So, some parts of this project were already developed before this new project was initiated. Naturally, some parts were in a "proof-of-concept" state, so they were either fully developed or improved by the current team. Some former features were deprecated due to changes in the process design.

Of course, a significant challenge is associated with the size of this project and with in-depth knowledge required in many areas. To overcome this, to each team member were assigned the tasks that would best fit their strengths. Unfortunately, I also had to accept the fact that this opportunity did not come without a personal cost. I soon realized that managing such a large project would require my focus on the design and management of the project rather than its implementation, which is a part I also enjoy doing.

OBJECTIVES

The primary objective of this paper is to provide a guide for developing a large automated software system, whose functions are: reviewing of data and metadata standards, support of metadata standards creation, and trend visualization. Its secondary objective is to help prevent such a large software system becoming unmaintainable and eventually decommissioned. To build a successful, maintainable and intuitive large software system, the focus was set on project management, system configuration management, design of the system architecture and finally easy access to documentation.

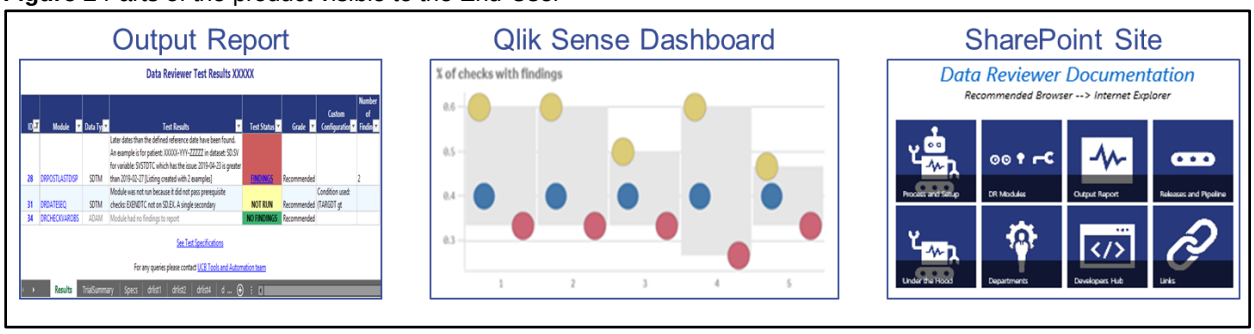
METHOD

The methodology used is called incremental model[2]: it is a software development method, where the model is designed, implemented and tested incrementally, as the name suggests. The product is built little by little until it is finished. It follows the philosophy of prototyping, where the product is decomposed into components[3] and modules[4], each of which are designed and built separately. The product is finished when it satisfies its requirements. Each component can be utilized independently. This allows partial utilization and user testing of product and therefore avoids long development time. This methodology also facilitates the code refactoring[5], which is needed to reduce complexity and improve software architecture and code readability.

OVERVIEW

The final product is a large, automated and maintainable system for reviewing of metadata standards, data checking, and trend visualization. It runs over each clinical study completing an enormous number of repetitive tasks more quickly and accurately than a programmer can when running standalone programs. The process output consists of an Output Report, which is sent automatically to the users for review together with a link to the Qlik Sense visualization dashboard. In addition, instead of creating traditional word or excel documents, a SharePoint site for knowledge distribution was created (**Figure 2**). The site contains all the information related to this project, from technical documentation aimed at expert users to user documentation needed by regular users. Stakeholders and management are also kept up to date through announcements on this site. The SharePoint site (and its content), Output Report and Qlik Sense visualization dashboard are the only parts visible to the final users. Everything else is visible only to system developers.

Figure 2 Parts of the product visible to the End-User



1 https://www.lexjansen.com/phuse/2015/pp/PP01.pdf
2 https://en.wikipedia.org/wiki/Incremental_build_model
3 https://en.wikipedia.org/wiki/Component-based_software_engineering
4 https://en.wikipedia.org/wiki/Modular_programming
5 https://en.wikipedia.org/wiki/Code_refactoring

Before delving into the core of the automated process, let's share some thoughts regarding the project management and the release pipeline.

PROJECT MANAGEMENT

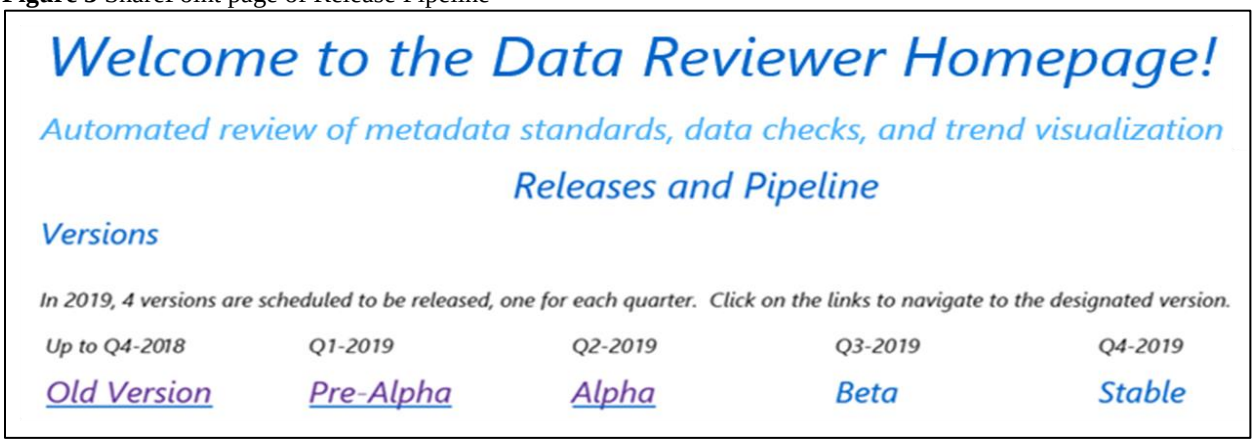
A large software project can be risky, complex, and it needs a significant budget. It requires a lot of structure and effective management to be successfully completed because the software system must be able to adapt and change quickly to accommodate new requests from stakeholders. Therefore, these changes require a considerable amount of management effort for making sure they do not affect maintainability. Software maintainability is a long-term aspect describing how easily a software system can evolve and change. So, the only way to create a successful software system is to organize it by having a clear vision of the end-product. What's more, special attention needs to be given to the roadmap for project development, milestones, timelines, tasks definition, tasks assignment and finally to concisely communicate the state of the deliveries. It is also very important to ensure that communication between team members, stakeholders and management is honest, concise, clear, and very frequent.

To facilitate the project management, communication and tracking of the progress of task implementation, Microsoft Planner and Microsoft Team were used. These were very useful and yet very easy-to-use tools. Frequent team meetings were held to discuss about important topics regarding the project. These meetings were also very useful to get the team to work consistently and to share tricks and techniques. Finally, a selection of highly professional team members was a key factor to the success of this project. Consultants with an extensive experience in standardization and automation were selected.

RELEASE PIPELINE

All process documentation is kept in SharePoint, including the release pipeline. **Figure 3** shows the SharePoint webpage containing the release plan**.** Four development phases were planned, with deliveries at the end of each quarter of the year. Pre-alpha, Alpha, Beta and Stable are the planned releases respectively for Q1, Q2, Q3 and Q4. (At the time of writing this paper, Beta version is about to be released.)

Figure 3 SharePoint page of Release Pipeline



The purpose of the Pre-Alpha roll-out was mainly to test and gain the confidence of both management and stakeholders. For this delivery activities including analysis requirements, software design, software development, and unit testing were performed. For the Alpha version, new modules were developed, and testing was done mainly by using white-box[6] techniques. Additional testing was then performed by clinical study leads. For the Beta phase the focus was on two major tasks: development of new components of the system and fixing known or unknown bugs. This version did not contain all the features that were planned for the final version, for example the visualization dashboard. The Stable release candidate is the final product. At this stage of our product stabilization, the expectation is that all product features are designed, coded and tested. After this release no major changes are planned unless there are new requirements from stakeholders. Of course, some code changes to fix defects can still occur.

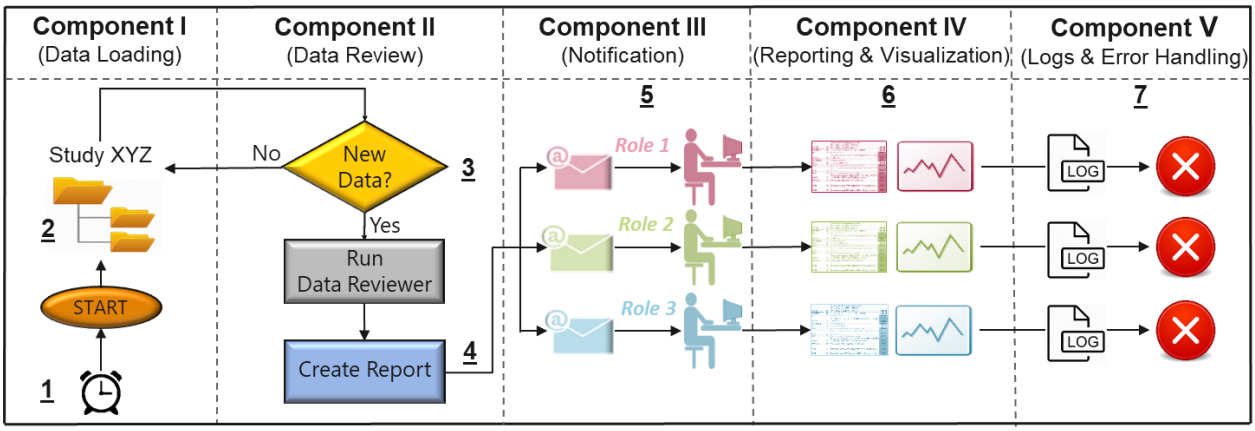
Two weeks before each version was released, mock releases were made to make sure everything was running as planned. A high level-view of the process flow follows in the next section.

[6] https://en.wikipedia.org/wiki/White-box_testing

PROCESS FLOW

The process starts when UCB receives SDTM (Study Data Tabulation Model) and ADAM (Analysis Data Model) data from our partners. Data is automatically unzipped into the study specific standard locations. The automation process starts through a scheduler which kicks off the process at a specific time under a service account. (See point 1 in component I of the **Figure 4**). The process first loads the entire study list and then recursively checks if new data is available for the current study (point 2). If new data is available, then the first component of the process extracts SAS dataset from XPT files (point 3). Subsequently, the component for the metadata and data checks which builds the output report (point 4) is launched. When this is completed, the notification component is launched and sends an automated email (point 5) to the various stakeholders (Role1, Role2, Role3) containing reports and links to trends visualization (point 6). Finally, process logs are sent (point 7).

Figure 4 Process flow



Note that the process flow in **Figure 4** only shows a very high-level view of the process. The size of the process may rise questions on maintainability, however, thanks to the architecture design that eases maintenance and its structure made of several blocks of components and modules, the process remains fully maintainable regardless of how it expands over time. However, maintainability will be discussed later, now let's have a quick view of the architecture design based on components and modules.

ARCHITECTURE DESIGN

HIERARCHICAL AND SEQUENTIAL COMPONENT-BASED

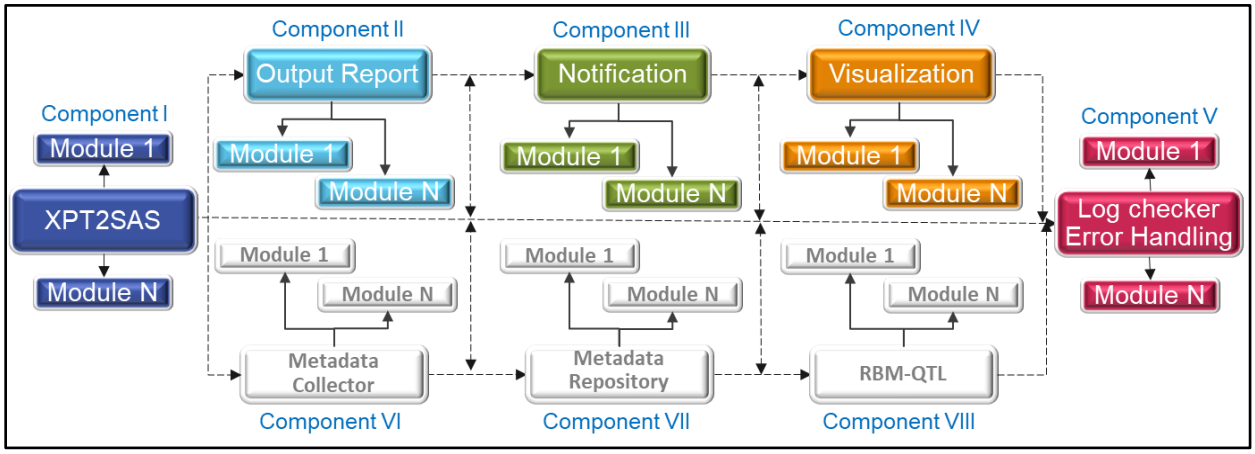
The hierarchical composition is the architecture where one component provides data to many other components. For instance, Component I provides data to all other components, see **Figure 5**. Sequential composition is the architecture where components are launched in sequence (e.g.: Components I to Component V). This was achieved through continuous code refactoring and architecture re-design. Code refactoring prevented redundancy, increased code readability, reduced complexity and improved maintainability. Without the regression testing[7], the code would not have been able to be refactored so safely, and the continuous improvement that brought this project to where it is now would have not been possible.

Following are the major components of the system:

- Component I: Automatic unzip and extract of study data. Ensures data are available for each study.
- Component II: Output structured report allows easy navigation of data/metadata issues and documents.
- Component III: Automated and targeted selection of email notification.
- Component IV: Trends visualization of data and metadata issues in Qlik Sense.
- Component V: Log checker and error handling.
- Component VI: Support creation and maintenance of metadata standards.
- Component VII: Support bulk load of metadata standards into metadata repository
- Component VIII: Risk Base Monitoring (RBM), Quality tolerance limits (QTL)

7 https://www.phusewiki.org/docs/Conference%202014%20PP%20Papers/PP01.pdf

Figure 5 Hierarchical and sequential component-based architecture design



NOTE: Components VI, VII and VIII will not be part of this paper. They have only been included in **Figure 5** to show the scalability and maintainability of the system architecture. Hence, from here on, these parts will be excluded from the discussion while the focus remains on the other components.

COMPONENT I: XPTTOSAS

This is the very first component of the system. Its main function is to parse each study folder and extract SAS datasets from standard SAS transport files received from our partners or created internally at UCB. The component runs once per day and it is triggered by a scheduled task running under a service account provided by UCB IT. Note that the service account is needed because without it the scheduled task cannot run when the user is not logged in UCB internal network. Extracted datasets are accessed by multiple departments and for that reason this component must be as reliable as possible.

COMPONENT II: OUTPUT REPORT

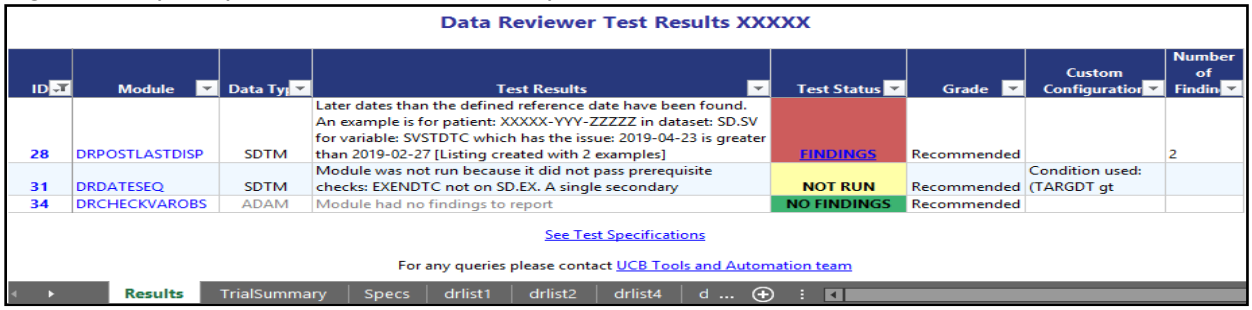
The system's output report is simple and easy to understand, data problems are reported directly to the consumer using targeted, time-saving, automated checks grouped into the Output Report. The current version of the output report increases speed of data review, supports the creation of metadata standards, and provides source files for the visualization of data quality trends. The report contains various tabs: a Result tab, a Trial Summary tab, a Specs tab and tabs for modules' specific findings.

RESULTS TAB

This tab contains a summary of the check results for each module. The table consists of an identification code for each check, the type of data on which these checks run, a summary message, a status for each of these tests (red for findings, green for no findings and yellow for not able to run), a grade for the importance of the check, a column for customized checks and finally the total number of issues per check.

For simplicity the image shows only 3 checks, but the output can literally have any number of checks. Also, the text is grayed out in the case of a check which did not find any issue. This helps focusing on significant findings.

Figure 6 Output report of Data Reviewer Component



Data Reviewer Test Results XXXXX

ID	Module	Data Ty	Test Results	Test Status	Grade	Custom Configuratior	Number of Findin
28	DRPOSTLASTDISP	SDTM	Later dates than the defined reference date have been found. An example is for patient: XXXXX-YYY-ZZZZZ in dataset: SD.SV for variable: SVSTDTC which has the issue: 2019-04-23 is greater than 2019-02-27 [Listing created with 2 examples]	FINDINGS	Recommended		2
31	DRDATESEQ	SDTM	Module was not run because it did not pass prerequisite checks: EXENDTC not on SD.EX. A single secondary	NOT RUN	Recommended	Condition used: (TARGDT gt	
34	DRCHECKVAROBS	ADAM	Module had no findings to report	NO FINDINGS	Recommended		

See Test Specifications

For any queries please contact UCB Tools and Automation team

Results | TrialSummary | Specs | drlist1 | drlist2 | drlist4 | d ...

TRIAL SUMMARY TAB

The Summary Tab displays the SDTM trial summary domain (TS). It is useful for the reviewer to have trial information. It is also useful for visualizing information in Qlik Sense. For example, it is possible to group by type of studies and analyze output reports by indication or by study design.

SPECS

Next tab is the specification setting file. It contains all parametrized checks that run for each module. Amongst this information are the module definitions and their parameters. These modules are designed as general-purpose and they can be used to check consistency and basic derivation of data by modifying input parameters. This is done via a configuration file which will be not described in this paper. There is no restriction to the number of modules and related parameters. A check is defined as a module with a specified set of parameters. Hence a generalized module can perform many different checks.

PLANNED IMPROVEMENT

There are often ad-hoc requests by study team for running the process on specific study data or data pooling. This is intensive labor cost because it requires a change in process configuration file. The plan is to further improve this so that that the study team can make ad-hoc requests for running the process via a SharePoint form.

DRLIST

Through these tabs it is possible to drill down into the findings in the form of listings customized to each check. To avoid excessive processing time and creating unnecessarily large files, only the top 500 results are shown in each tab. However, the total number of issues per check are still present in the result tab. The structure of the "drlist" tabs is not fixed allowing outputs tables to be clearly tailored to the checks.

COMPONENT III: NOTIFICATION

At the time of writing this paper, this component is made of two main parts. A study list spreadsheet used as a setting file, and a SAS module. The spreadsheet holds information about studies, team members, source data location, scheduled time of system running, email receiver, email subject, email text, output location etc. System developers have also a shared mailbox in Outlook where the project development team receives the emails from this component containing process logs and output reports for each study. Depending on the study and nature of the issues/concerns, an email is sent to every person responsible for that study, with the output report attached.

PLANNED IMPROVEMENT

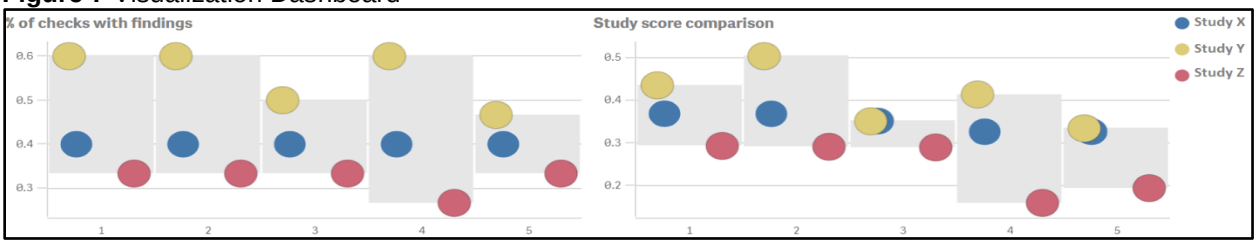
The study list spreadsheet is a temporary solution and in our development pipeline there is already a plan for replacing it with another system or with a connection to our Clinical Trial Management System (CTMS).

COMPONENT IV: VISUALIZATION

This component visualizes the data issues collected in the output reports to help UCB run efficient and successful clinical trials. A visual dashboard helps identify significant trends and deviations, hidden patterns and detect important exceptions so that they receive the necessary attention and real-time reaction. A very simple example of the visualization dashboard is shown in **Figure 7**, where a plot displays the distribution and range of data values over a scale.

The dashboard visualizes the content of a cumulative report generated from the reports of different studies at different time points and summarizes the scores of each output check. Note that due to the big difference in issue occurrence-count in the individual output checks, a score was assigned to each check for each study. To do that, a denominator was used in each output check and then the scores were normalized (between 0-1) against all other scores for the same check for all studies. This allowed us to see how the study Y performed in relation to study X and Z within specific time (X axis). The figure below shows that study Y performed worse than study X and study Z.

Figure 7 Visualization Dashboard

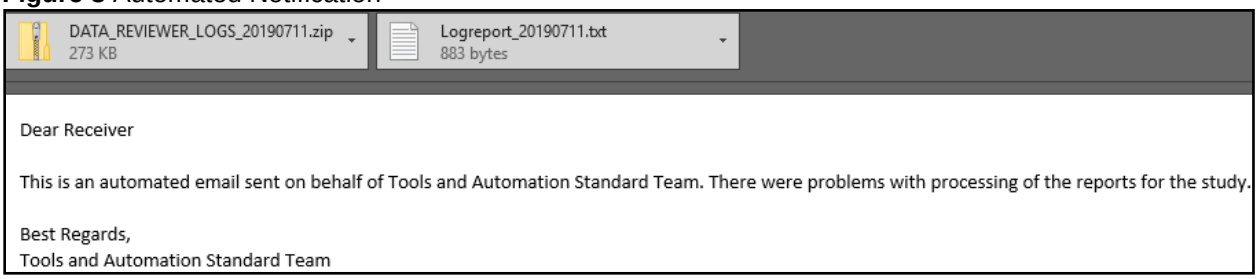


Following the same method, another distribution plot could be created to see how a specific score check performed in comparison to all scores for the same check across all studies and all times, or within the same study. At the time this paper was written; the visualization dashboard was not yet released.

COMPONENT V: LOGS AND ERROR HANDLING

As stated many times in this paper, this is a large software process consisting of several components and modules generating hundreds of log files. The component handling logs and errors is the last one to run and parses each log generated by all modules of each component. It extracts only the most useful information and creates a summary of all logs. It finally zips all logs under the project and study folders with a time stamp in the file name. If any log contains an error, warning, or anything else which needs the team's attention, this component calls the notification component (Component III, Figure 5) and sends an email to our outlook shared mailbox with the attached file which was previously zipped plus a file containing a report of all logs. In other words, operational tasks are not performed by the development team members unless a notification of a system issue is received.

Figure 8 Automated Notification



Having described the process flow, the architecture and the various components of the system, the last part of this paper focuses on maintainability.

MAINTAINABILITY

Software programs become more complex and difficult to maintain as more developers join the development team or when more components or more modules are added. Even if you find the best specialists in the field, you may still have a mess or heterogenic code - it's sometimes unavoidable, but it can be constrained by a constant refactoring and architecture redesigning. Software refactoring and redesign can increase enormously the maintainability, and these are perhaps amongst the most expensive and resource-requiring phases of the software development lifecycle. Therefore, assigning the right team members to the right types of refactoring and redesign is a critical aspect of the project management. Though the refactoring and the architecture redesign may not change the external behavior of the code, it improves its internal structure so that not only the current development team is benefitting but also future maintenance done at any level by regular users or system developers. Let's see now how the types of users and levels of maintenance are defined.

TWO TYPES OF USERS

In this system there are two types of users: regular users and system developers. In the current release, regular users can only review the output reports. After the stable release, regular users will be able to setup the system for scheduled runs and/or ad-hoc requests. System developers can develop new components, modules or interfaces between components and/or modules. Other types of users may be created in the future if necessary.

LEVELS OF MAINTENANCE

This software system is used globally for many UCB clinical studies; thus, due to the large amount of data input the system must process, maintenance is inevitable and will be done at the following levels: process run, process configuration, component and module.

PROCESS RUN

This level requires a very general knowledge; after the stable release it can be performed by regular users. It is the only level on which regular users can interact with the system and it is the also the most frequent level used. It is mainly related to study specific settings and selection of data checks to run. As stated previously, for each study there is a setting file configured at the study set up. It contains mainly information about location of study documents and their versions, such as study Protocol, SAP (Statistical Analysis Plan), Define.xml etc.

From these documents the system extracts some information and compares it against data. For example, if SAP contains “Kaplan Meier” or “Kaplan-Meier” or “Time to Event Analysis”, then ADTTEx dataset should be available amongst the data transfer. It also contains more detailed information such as name of treatment emergent flag variables, treatment start and end-date/datetime variables, AE (Adverse Event) start date/datetime imputed variable, number of cut-off days etc. At the time of writing this paper, this level of maintenance is covered by system developers. However, a further improvement of the system (planned after the stable release) can remove this level to make the system even more automated.

PROCESS CONFIGURATION

A thorough level of testing is always imperative in a multi-component software system. Maintenance is defined by how quick and cost-effective modifications to the process configuration can be done by system developers. By changing the configuration, the testability is affected because testability is also defined as the capability of the configuration to enable modified software to be validated. Testing is therefore highly affected by the process configuration.

Three levels of testing were performed: developer testing (the white-box method described in the release pipeline section of this paper), full quality control (QC) performed by study leads and automated regression testing[8]. In addition, whenever possible, the configuration has been designed not only to be easily modifiable for changes that can be predicted, but also for the modifications which are not predicted. Three main principles were followed: simplicity, consistency, and easy navigation across development environment and releases. Process configuration maintenance includes the process architecture and SharePoint site.

COMPONENT

A component is a software unit whose functionality and dependencies are completely defined by its interfaces. At this level the maintainability varies depending on design, size, and interfaces of the components. Components for reuse may be specially built by generalizing existing components. The focus was on creating components as independent as possible to increase testability and integrability within the rest of the process. Future extensions to the architecture may also include integration of third-party software components. Therefore, integrability concerning third-party components is also supported. Of course, by doing this there is an extra cost for enhancing the reusability, but this should be considered as an organization rather than a project cost. Clearly, components can only be maintained by system developers.

MODULE

There are two types of modules in this process. Functional module and utility module. A functional module has been defined as the smallest block of code able to produce a valuable output for the reviewers. In fact, these modules are designed for returning output data and/or metadata checks. It is usually focused on implementing specific types of functionality. Its implementation is a module with a specific set of parameters as shown in the example below:

```
%macro functional_module(param1=Val1, paramN=ValN)
 %drutstart();        %*initial state workspace: datasets, catalogs, options*;
 %drutvaldsvars();    %* Prerequisite checks *;
   ……                 %* Body of module *;
 %drutend();          %* Cleans up any datasets created or options changed *;
%mend;
```

Utility modules produce values which are used by the functional modules. The previous version of this tool did not include the utility modules `drutstart`, `drutvaldsvars` and `drutend`. However, the code was already separated into blocks ready for code refactoring. Clearly, having utility code separated from functional code makes the latter ones much more stable and less prone to changes.

8 https://www.phusewiki.org/docs/Conference%202014%20PP%20Papers/PP01.pdf

CONCLUSION

Data review can be time-consuming and costly. Running ad-hoc programs at each review cycle does not bring a significant cost reduction because of expensive repeated manual checks and high level of maintenance required. In addition, the risk of missing or ignoring significant trends could potentially put the studies at risk.

This paper presented a software system that facilitates greatly the activities of reviewers who are asked to review data within a short timeframe. This is achieved through a well-structured output report and a visual dashboard which helps identifying significant trends and deviations. It also supports the collection, creation and storing of metadata standards into a repository.

Furthermore, the design and the implementation of the system allows end-to-end automation while reducing maintenance costs. The building blocks of the system are small components and modules which ensure flexibility and scalability, so that if some parts become obsolete, they can simply be unplugged or rebuilt independently with no consequences on the other parts of the process.

Finally, the project can be considered as a breakthrough in the field of data review and trends visualization. It is an unattended Bot, which runs automatically on a server and sends notifications with output report to each study team member. Thus, no statistical programming activity such as development, maintenance or execution of programs is needed for any study. Once fully operational, the system can be maintained by just one person with minimum effort.

Wouldn't this be an example for reducing costs and increase quality and efficiency?

ACKNOWLEDGMENTS

I would like to thank Jim Elder, Jeremy Teoh and Przemysław Sosna. Without their continued efforts and support we would not have been able to bring this automated process to a successful completion. I would also like to thank my colleagues at UCB Biosciences for their support and comments.

RECOMMENDED READING

1. A Data Science tool for Data Reviewing
   https://www.lexjansen.com/phuse/2015/pp/PP01.pdf
2. Reengineering a Standard process from Single to Environment Macro Management by Giuseppe Di Monaco
   https://www.lexjansen.com/phuse/2014/pp/PP01.pdf
3. Boosting the efficiency of data review by using SAS Integration Technologies and VBA by Giuseppe Di Monaco
   https://www.lexjansen.com/phuse/2016/cs/CS01.pdf

CONTACT INFORMATION

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

Giuseppe Di Monaco
UCB BIOSCIENCES GmbH
Alfred-Nobel-Straße 10
40789 Monheim, Germany
Tel: +49.2173.48.2091
Fax: +49.2173.48.1947
Email: Giuseppe.DiMonaco@ucb.com
Web: www.ucb.com