PhUSE EU Connect 2025

Paper AD12

The Key Features Every Statistical Computing Environment Should Have and Often Lacks. A Presentation About Almost 20 Years of Experiences with SCEs

Michael Weiss, P-300, Berlin, Germany

Abstract

While working for a big pharma company for around 20 years now in Statistical Programming & Analysis, I have been part of multiple projects related to Statistical Computing Environments (SCEs), Clinical Information Environments (CIEs) and how else they are called. Having seen environments from vendors in different status and with different focus. This paper tries to look into the SCE from different perspectives and shows up some often-missed features.

This paper avoids names of vendors and solutions and simply talks about SCEs.

Introduction

Some years back there was a presentation at the PhUSE about how to exchange SAS programs between a "custom" environment and the official SCE solution from an external vendor. The presentation was explaining in detail a lot of work that was done to be able to have an automatic transfer of programs and other required files between the two environments. Additionally, the presentation stated that this is done, because the development will be continued outside of the vendor solution and only the final run and final storage will be done inside.

This paper does not want to go into details about this presentation (or any similar one) but tries to bring some light into the question what features are possibly missing in the common vendor solutions, so that it is "required" to build up a custom solution next to an "official" SCE as there are other companies doing similar things.

To reach this goal, the paper will explain the core components of an SCE and describe some features that help to "Live inside and not next to the SCE". Additionally, this paper names some examples how these requirements should not be implemented, but have been this way in the past.

The core components of an SCE

A basic SCE can be broken into the following components:

- A Programming (or coding) Environment to supports the development of programs
- A Data Viewer to support understanding the data
- A Backend to store all the inputs, programs and outputs and implement all the GxP / GCP related functionality like access management, audit trails, lifecycle management and so on.
- A Navigator (or Explorer) Component to navigate between different compounds, studies files and so on.

Next to these components of an SCE, the SCE is often only a component within a bigger solution that e.g. supports all the clinical data flow "from EDC to submission".

When it comes to implementation, the available SCEs can be separated into two groups. The first that implements all the parts in a fully integrated way as one own solution and the other,

that puts together different independent tools (from different vendors) to create a toolset that builds the SCE.

Which one of these ways is preferred highly depends on personal preferences of the users and the technical skills of the vendor. When putting together external tools, users must get familiar with all these tools to fully understand them. The tools are mostly built for a different set of requirements and often fit just “80%” to the requirements of an SCE, so that the users have to jump between different tools and ways of working to get the job done.

When creating a fully integrated peace of software, the vendor has to build up a complete set of tools and will be measured at other available tools. If this is done right, users can leverage from a fully integrated working environment with a straight UI experience.

Features an SCE should have

In the following sections some really useful features are explained that should be stated on every SECs requirement list. Some might look like “not so important” but they can really make a difference in day-to-day work. It is proven that people who like to do the work are faster and do less errors.

Program Editor

The program editor is still the primary component, a Study Analyst is working with. If for the task of study program creation or when working with the data, most of the Analysts are writing a lot of program code. And especially when doing validation, a good program editor helps a lot in understanding of other people’s code.

It should be clear, that correct **syntax coloring** is a minimum requirement for any program editor and even this is not available in all SCEs. Next to this there are other must have features for a good programming editor.

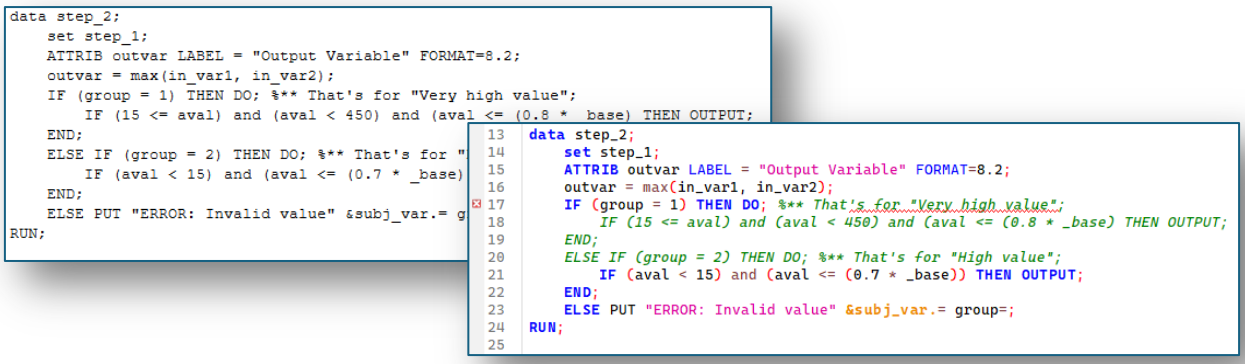

Figure 1: A Program editor with and without syntax highlighting

Good Program Editors have Code Completion features like **templates** or **syntax completion** build in. This provides some template code like for function arguments or variable names. This not only increases typing speed, but (especially for variable names) reduces typing errors.

To support the understanding of code, **ToolTip’s** can help and give some extra explanations like actual variable values or the parameter names for a function.

Especially while doing validation of adapted template programs, an overview of what code was changed by whom, can provide great help. An example for this is the “blame” view in github, that shows who last changed which line of code when.

If programs are copied from a global standard location and adapted afterwards, or if multiple programmers are working on the same program, it can really be helpful to see which parts are from the global standard template and which parts are changed / added by whom on study level. If now **validation comments can be done within the program editor**, no study team needs to keep validation lists in Excel or OneNote anymore.

Another feature, the Program Editor of every SCE should have, is the possibility to **execute code interactively** "step by step". This especially means, that the status of the program can be investigated between each of these steps and e.g. the generated data can be browsed or actual values of variables can be seen or even a fully functional single step debugger.

As a next feature, the program editor can integrate the execution log into the program in a way, that on errors or warnings, the code line that caused the log message is highlighted and the log message is "attached" to this line. This is not only usefully for interactive execution, also in background execution it can get tedious to manually align the log messages with the related program code.

Keeping all this in mind, the biggest "fail" of a program editor, that exists in real live, is a Multi-Line text box in a web page. No colors, no highlighting, no extra support. Only a Save button to save the program.

Data Browser

The Data Browser is the second core component of the SCE for Analysts and might even be the primary component for Statisticians. If the Data Browser is not able to fit the needs of the users, it still seems to be a common way to copy over data into Excel or similar tools. This can lead to a lot of extra work and cause issues in reproducibility or similar things.

The first feature of a good Data Browser is definitely **filtering and sorting** capability. In most cases the users do not only want to brows the data, but also understand the data. Especially when working with vertically structured data, like most SDTM or ADaM data is, filtering is really important to find the relevant parts. When designing the filtering easy usage should be number 1 priority. Providing a text box, the user can write some SQL WHERE conditions in, will allow to define all possible filters, but is far away from being comfortable.

What is often missed as well, is the same possibility on the horizontal axis, meaning to be able to **re arrange and filter variables** in the data browser. Especially if in ADaM, variables from ADSL are merged to other domains, the number of variables can easily become more than 100. If now horizontal scrolling is needed to view all related variables, the overview is lost quite soon.

While all this can be done by program code as well, having a fast UI helps a lot especially in meetings when discussing the data with other parties.

In real life there still exist data browsers without any of these features, but with a paged display that splits up the data into chunks of 20 records and the requirement to click the "Next" button to see the next 20 records.

Another "fail" of real-life Data Browsers is one, that uses a paged display as well, supports filtering, but applies the filter only to the actual page. This means that page 2 can be empty (because of the filter), but page 3 will again contain values.

The Backend

The backend is the component that allows the most differences in design and features. Implementations reach from simple file systems with basic security models to complex data base backed systems where files are stored as binary objects in data base tables.

From a user perspective and from features, the implementation of the backend is completely irrelevant. It is often the case, that file system backends are faster than backends with databases, especially when the programs are not running inside the database, but require an export of the data for evaluation.

Next to standard features of the backend, like enforcement of access restrictions and data governance, most of the features are visible to the users from within the Navigator / Explorer component of the SCE, like lifecycle management. Still there are other possible features that are really helpful, but not always available.

The first of these features is an **automatic tracing between input and output per program**. If at all available, this is mostly implemented by a manual linking in the UI or within program code. This means that users must link the required input files and the expected output files per program. For multiple hundred programs within a study, dozens of input data sets, metadata, standard programs or macros and so on, this is a high effort that often leads to workarounds like "include all everywhere". Having a correct automatic tracing between all input and output elements of a program run can help a lot in understanding the study flow for validation or later to identify all the inputs used to generate an output.

Another feature that is often missed is a **smart separation of access** between program lifecycle stages. While in development it is often useful to have read access to elements in production lifecycle, but when working in the production lifecycle, it should not be possible to access elements from development or validation.

In real life it is either not possible to access elements from different lifecycle stages at all, or it is possible to access all elements from each lifecycle.

And last but not least, an often-missed feature is the **easy access to elements stored inside** the SCE. This covers two different features. On one hand the access to files like programs or results. These files should be accessible by the users Explorer to support access by programs outside the SCE for example with Office Applications.

On the other hand, in case the SCE is designed to be the place where ADaM and possibly SDTM data is stored, data access should be granted in a common way for external tools. In 2025 the standard method would be a HTTPS based API like REST or GraphQL. This access allows to do scientific data analysis in external "state of the art" tools, without manually storing the data in other systems.

The biggest "fail", when it comes to access of elements inside the SCE, is a web based system, that required to download every result file separately and "hides" these files behind a lot of clicks. Even the log file, created by program execution, was "hidden" behind more than 5 clicks.

The Navigator / Explorer

The navigator can be seen as "the part between the other parts". It provides access to the (otherwise invisible) backend features and is used to open programs into the program editor or data into the Data Browser. A feature rich and good designed Navigator can provide a fast and good overview about the status of the study programs and all related study information.

Often the navigator is a basic File Explorer, **sorting the available file by date or size** should be a standard feature, but some SCEs do not even provide this information in the navigator. **Additional information** like the user who did the last change or lifecycle status of the file **should also be displayed** directly in the navigator.

A really helpful feature, the navigator can provide, is an **integration with other systems**. For example, if the study data is stored in an **external database**, the navigator can still display the available data tables and allow to open these in the data browser. Another example is the integration of **external document storage systems** like network drives, SharePoint or Documentum. And if the results, created in the SCE should finally be stored in another system like an eTMF, an integration to automatically upload results can also be triggered from the navigator.

Another useful feature is a **linkage between the navigator and the program editor or the data browser**. Especially when working between multiple studies, it is a great help if the navigator jumps to the directory, a program or a data table is stored in, just by the "press of a button" or if the data browser selects the variable that gets selected in the navigator.

AI, LLMs and how these influence the SCEs

Initially this paper was intended to not cover AI. On one hand, this is something that is everywhere announced, so it's definitely not a "missed feature" and on the other hand this is nothing that will make other features irrelevant.

The LLM can be seen as a good co-worker that helps writing code and can explain the data very fast, if it is correctly integrated into the SCE. Understanding and validating the generated code, even more requires a good SCE that supports the analyst. Assuming that the LLM will generate correct code in 90% of the time still requires to find the last 10% of the code that need to be adapted manually.

Conclusion

The Statistical Computing Environment for clinical development still seems to be a piece of software that is not followed up in feature development in a way that is comparable to other programming environments.

Acknowledgments

I would like to acknowledge Elena Glathe who always pushed me to write all this down and publish it after we both have been discussing all these requirements and what would be useful for clinical development workflows.

Contact Information

Michael Weiss

michael.weiss[at]code2be.de