# Aspects of software development


## Katja Glaß

Bayer Pharma AG, Berlin, Germany



A lot of aspects have to be considered for the software development and maintenance. This article will give an overview of some of them when dealing with the creation or maintenance of standard systems especially within the clinical environment. These aspects are estimations regarding effort, required group sizes, programming principles, ways to motivate the end users, and other aspects.



**Keywords:** Standards, Systems, Programming


## Introduction

More and more companies are developing or updating SAS macro systems and similar tools to enhance productivity and product quality. If these tools are to fulfill their purpose, developers need to consider, and make sure not to forget, many aspects of the process. In order to help the reader to understand the main principles and find issues before they arise, this paper will outline the most important of these aspects.

Modern business practices require increasing productivity from daily work, and in the pharmaceutical industry, quality requirements, already high, are continuing to grow. The need for efficiency, in particular, has led to increased standardization: in order to produce study results in ever-shorter periods of time with extremely high validation requirements, software systems such as SAS macros can perform commonly used routines. The value of these systems increases with added flexibility so that even small companies with limited funds, such as CROs, may enjoy the benefits of standardization. Besides economic efficiency and a reduced validation effort, CROs can utilize macro systems as source generators or as commercial software. This paper will outline ways of supporting efficient and successful software creation.

## Macro System Paradigms

Several aspects need to be considered when planning a system. What will the system do, or in other words: what is its main purpose? What kind of application will the resulting tool be? Who will develop it? And which development model should be applied?

### *Systems by purpose*

Systems can be grouped by purpose, as, for example, a toolbox system or a more complex application, but categories like these need not to be restrictive; the programs for a single system will fulfill many different purposes. In the case of SAS, a toolbox macro system can be very powerful, providing small tools needed on a daily basis that can be standardized, validated, and stored. The users of these tools do not need to revalidate their functionality.

Especially for CDISC, it is worth considering the option of a 'closed' macro system application. 'Closed' means that much of the system's functionality is invisible to users and the macros have powerful effects despite having only a few parameters. When datasets, variables, labels, and formats have uniform definitions throughout all studies, the tables created by the system will presumably also be uniform. The system may then generate special kinds of tables such as, for example, an adverse event table. As a rule, so long as the requirements are stable, the user need only call one simple macro to create the table. Maybe a few options should be available as a parameter set for special purposes such as, for example, to include a total column and support subgroup analyses.

In contrast to the closed system, an 'open' system is a flexible application with many options. 'Open' means that all functionality is visible through parameters and the system permits modification of many results. An 'open' system may perform the same task as a 'closed' one, but with much more flexibility at the cost of increased complexity. For example, a macro to create an adverse event table might have parameters for the dataset, the variables, labels, and necessary formats in addition to other options such as column widths, additional selections, layout options, and many more. Users need sufficient training to learn all options and understand critical but unobvious details.

All three forms of systems are important and could be used together. Every company should have at least a toolbox SAS macro system as this is easy to create,


Correspondence to: Katja Glaß, Bayer Pharma AG, Sellerstr. 32, 13353 Berlin, Germany. Email: Katja.Glass@Bayer.com




Published by Maney
DOI 10.1179/175709311X13166801334235

to maintain, and to benefit from, but when sufficient resources are available, a complex application is worth creating in addition. When customers are highly creative, an 'open' system would be the right solution, whereas a 'closed' system is better adapted to standard deliverables.

### *Application types*

Apart from content, also the application type provides many options to the developers of macro and other systems. Developers can provide SAS-based systems as pure SAS macros, as web-pages, with a graphical user interface or even as part of other applications such as Excel. When the users are themselves SAS programmers, SAS macros are well adapted to their expertise and working environment.

### *Developer*

Who should develop the software? The best place to find area expertise and company knowledge is within the companies themselves: their employees understand their workflows and processes better than anybody. They know the specialized cases which set the company apart from others. Typically, a pharmaceutical company will already have SAS programmers who create macros for themselves and use them in several studies. If the user is also the developer, motivation and expertise will increase, system acceptance will be higher, and it will be easier to train up new users.

As company resource issues are usually limited, external companies may support the software creation process. They can perform validation or, if user resources are especially tight, they may create the system themselves from a user specification. If the external companies are especially effective, they may be more efficient than the sponsor companies with respect to documentation, implementation, and validation. Unfortunately, these undoubted advantages come at the cost of user friendliness, as there is typically no direct communication between users and developers. Additionally, the barriers to enhancements are typically higher, as there is no internal expertise and even small change requests may be a great effort. My advice is to find a user as developer, if possible.

### *Software development models*

Before programming starts, it is important to identify a software development model. There are different models available for programming.[1] The Waterfall model is no longer in fashion as every development step is only checked against the previous one and so the final result may not necessarily meet the initial requirements.[2,3]

The most common software development model is the V-Model, in which all major development phases have corresponding verification and validation phases. The system starts life with a concept, which is typically an estimation from users or the future system owner. The specifications subsequently become more detailed, both technically and with respect to users, as the developer creates implementation requirements and decides the layout of the system. As a last step, a very detailed design emerges, after which system implementation can begin. Once the system is ready, various verification and validation steps begin; if there is a detailed design specification, the system is tested against the specification using integration and unit tests. The technical requirements are verified against the system verification and validation results. Verification of the general user requirements typically takes place in form of a user validation in the context of general operations.[2,4]

Owing to the high validation requirements within the clinical field, system and user validation typically takes place at the end of the process. It is often a problem to have the user requirements at the beginning, as many ideas and specifications emerge during development. Other implementation models help in dealing with this problem, of which the most efficient is prototyping.[5,6] Using this method, the developers collect specifications from an initial study as it is evaluated and validate the macros for the study. As the macros are reused for more studies, new specifications are added. Study validation for these macros is still required at this point. Finally, both specifications and macros reach production state and a system validation, with checking against user and technical requirements, is possible. The main advantage of this process is that the prototypes used to generate the requirements have already been in use on a daily basis with real data. Accordingly, many special cases and real-world problems will already have been solved.

To reduce study validation effort during the specification phase, another design model is useful. One significant aspect of Extreme Programming is that there are several release phases. First of all, the developers set priorities in the specifications so that they decide which specifications to implement for each release phase. Thus, the first version can be released to users in validated form quite quickly after implementation, and the initial validated form provides a basis for all subsequent development and validation. After the first release phase, the developers have acquired additional experience and so, for the next release, they can review and modify the specifications according to what they have learned. This model supports changing customer priorities and requirements.[7,8]

## Specifications

The basis of software is the specification: this document specifies content, error handling, inputs, outputs, and other things. Most problems are based on insufficient or incomprehensible specifications. These documents should be created by people having content related and technical expertise. The smaller the group responsible for specifications, the faster it can finalize the specification document. Nothing is faster than a one-person project, which is the preferred way, but larger systems may need to take many points of view into consideration. Larger groups are comprehensive, whereas small groups are more efficient, but efficiency and breadth depend most of all on the team itself. The team must consider how to balance individual requirements against the interests of the group and make best use of the available skills, the roles of people involved in the group, and the presence or absence of people with special responsibilities.

## Time Estimations for System Creation

The final implementation is typically just a minor part of the software development process. When people are experienced, the implementation part typically takes much less than 20% of the total time. The most time-consuming part of the work is the validation, which can take at least 60% of the time if not more. Special tools can be used to support validation, of course, but the total effort is the same as excessive validation is required to create a trustworthy system to validate any kind of input and which will meet regulatory requirements. Documentation and training are also essential to the creation of a system and these too consume large amounts of time.

## Rollout

System rollout is a very sensitive topic which must not be underestimated. User acceptance is of the highest importance, as the users will work with the system daily, and their schedules are often already overloaded. Additional training sessions, which are a requirement for new systems, should preferably take place shortly before first use. If the learning curve is steep, additional training time may be needed, but afterwards the software will presumably make the users more efficient.

Typically, there are always three types of users within a company. There are opportunistic users, users who accept changes, and very conservative users. If a system reaches conservative people at rollout, it is likely to be successful.

## User Motivation

There are several ways to motivate users when introducing a new system. First of all, the system must be user-friendly. The user should have been the focus in every development phase, but the user will need help in any case, for example, with clear support, trainings that are easy to understand, names that are meaningful, intuitive graphical user interfaces, and parameters.

When colleagues are system developers or deeply involved in the specification process, users will accept the system more easily, especially if the developer is convinced by the system and able to demonstrate its advantages to others. Colleagues will trust each other more than they will trust a trainer from outside.

Another important point is to include the user within the update cycle. If it is easy to address change requests by writing a simple mail or adding a list entry, the user will understand the potential for enhancements of the system and will take part in the process. Finally, if user changes are implemented according to user needs, the system will grow more and more user-friendly. Nothing kills ideas faster than bureaucratic methods of addressing change requests.

The system needs to provide users with sufficient confidence to be sure that the system will do what they expect. To this end, users need plenty of documentation and engaged users need the source available besides the validation documentation. Even if users do not have the time to understand complex macro source, they will feel more confident when finding a PROC MEANS or similar. Black Box[9] systems always result in uncertainty, especially when tasks previously performed by the user are now done by the system, even if it is validated.

Finally, the management needs to motivate the user too. Managers need to give users sufficient time to learn the new system along with their daily work. If some conservative users cannot be reached otherwise, management must force them to use the system, but with any luck even, the most conservative users will be convinced in the end.

## Additional Tips

For a successful system development, it is essential to collect ideas throughout the system's life cycle. Visions can only be created and implemented when the right ideas are available. Collecting new ideas from a stable group of users is sometimes not easy, so it makes sense to look for ideas elsewhere. An excellent way to get new ideas is to include students and work with them on work experience programs and bachelor or master theses to get them into groups and collect any resulting ideas. Some ideas may be unrealistic, but often they result in significant improvements. Additionally, this could help for later recruitments. And finally, whatever way is gone for

system development and enhancement, the user needs to be in the focus!

## Summary

Developers need to consider many different aspects when creating or updating applications. They should think about the options available to them and the special situation of the company. Each option has its advantages and disadvantages. If possible, users should create the system. Start with a simple SAS macro toolbox system in which the macros are prototyped and tested on various studies before a system validation. Use an external company for documentation and validation when resource limitations arise. Include the users in the maintenance process. Collect any ideas and use them for enhancements.

Standard macro systems can be very powerful and save resources. The main point is to align the system to the user: when the end user is the focus, system rollout will succeed. If we set standards, all standard tasks can be simplified so that, finally, the user has more resources to concentrate on the 'few' exceptions.

## References

1 http://en.wikipedia.org/wiki/Software_development_process#Software_development_models [cited 2011 Aug 30]; http://en.wikipedia.org/wiki/Software_Methodologies [cited 2011 Aug 30].
2 Thaller GE. Design und Implementierung – Kerntätigkeiten der Software-Entwicklung. München: Verlag Technikk; 2000.
3 http://en.wikipedia.org/wiki/Waterfall_model [cited 2011 Aug 30].
4 http://en.wikipedia.org/wiki/V-Model_(software_development) [cited 2011 Aug 30].
5 Rechenberg P. Informatik-Handbuch. München: Hanser; 2002.
6 http://en.wikipedia.org/wiki/Software_prototyping [cited 2011 Aug 30].
7 Jeffries R, Anderson A, Hendrickson C. Extreme programming installed. Boston, MA: Addison-Wesley; 2002.
8 http://en.wikipedia.org/wiki/Extreme_Programming [cited 2011 Aug 22].
9 http://en.wikipedia.org/wiki/Black_box [cited 2011 Sep 1].