Coding Standards for the Un-Standardizable?

Richard Pugh & John James, Mango Solutions

Introduction

As a SAS programmer, developing systems in a pharmaceutical company is a "familiar" task. Typically, systems and documentation are in place to support the use of SAS within the pharmaceutical environment. In particular, SAS "Coding standards" typically exist to guide the design, documentation, implementation and release of your SAS programs. In addition, coding standards documentation is available from resources such as sas.com.

"Coding standards" provide a guide to writing code that is clear, concise, and can be readily understood by other programmers. They provide a framework for the design, development, testing and deploying of code ready for a production environment.

S-PLUS and R

Other software systems have started to become more prevalent in the pharmaceutical industry over the last few years. In particular, the "S-PLUS" and "R" software system have become very popular, particularly in the early phases. These software systems are very powerful, and are often described with words such as "flexible" and "customisable".

While these characteristics are a godsend for users, they can lead to concerns for IT departments and programmers in the pharmaceutical industry. Whereas "SAS" is a known animal, how do we cope with these "new" technologies?

It is clear that utilising a different or new software system, such as S-PLUS or R, might provide more functionality that enables us to produce better outputs and speed development. However, having a more "flexible" solution can lead to issues with validation and standardisation.

Valildation

In the pharmaceutical industry, we need to ensure that we abide by the regulations and guidelines set down by authorities such as the Food and Drug Administration (FDA). There is often a misconception that SAS is the only format in which FDA submissions can be made. In fact, the FDA does not endorse or necessitate the use of any specific software including those softwares discussed in this presentation.

Having said that care must be taken to ensure the functionality used for a submission has been properly validated, whatever the software in question. In

particular, there is the need for proof that validation testing has been carried out, and that we can reproduce the validation test results.

The R and S-PLUS systems are flexible enough to integrate with internal processes, and provide sufficient information to ensure a validation procedure can be carried out.

At the FDA, S-PLUS and R have been officially approved as software systems that can be used internally by it's staff. Currently, these systems are used to perform many tasks internally including product review, research, simulations and analyses to support advisory committees.

A Moving Target

With a familiar software system such as SAS, the validation processes are well known. Also, the timing of SAS versions usually provides sufficient time to follow the process through and ensure the software is install and validated. For example, there could be 2 or 3 years between major releases of the software.

However, S-PLUS and R are more rapidly developed and released. Major releases of S-PLUS tend to occur every year, with major versions of R being released every 3 months. This often leads to conflict between the IT team members wanting to "fix" a version, and the users needing to use the latest features.

Perhaps the greatest challenge when using S-PLUS and R is the modular nature of the system. Both softwares can be extended using "libraries", which are predominantly written by users in the community. These libraries are numeric (about 800 for R) and are frequently updated, and the amount of validation performed can vary greatly. Another complexity is that, when loaded, user libraries can "mask" (or "overwrite") existing functionality.

This means that control measures need to be in place to ensure functionality used is validated and is not masked by unvalidated functionality. There are some straightforward approaches to this, thanks primarily to the formal structures that can be set up to handle library handling and validation testing. Following the R library format (and enforcing S-PLUS to use the same structure) helps to ensure R library components are correctly documented. Components such as "RUnit" also allow automated unit testing for components used. Command handling functionality within the S and R languages also allows clarity in the functionality being called, and allows verbose logging of these calls.

Standard Operating Practices

In order to ensure the quality and validity of the functionality in use, standard operating practices need to be in place. These guideline documents are needed

to outline the process and structure of developing and deploying R or S-PLUS functionality within a company's production environment.

At Mango Solutions, we are often involved in developing applications for customers that include S-PLUS and/or R components. As such, we need to ensure we have the documentation and structure in place to conform with regulatory requirements. To this end, we have create a "Coding Standards" document internally that governs how we develop and deploy these components.

Creating the Coding Standards Document

At Mango Solutions, we have a number of S-PLUS and R programmers both or a full time and part time basis. In order to develop the coding standards themselves, we needed a mechanism for discussing and documenting ideas regarding procedures and coding. In order to achieve this, we used an internal Wiki, which enabled our consultants to contribute and discuss the ideas.

This proved a great success, allowing consultants to debate the best way to structure our projects in a very accessible and easy to use format. It was also easy to extract agreed elements from the Wiki into internal reporting formats for publication within the company's "Software Development Quality Manual" document.

In generating the operating framework and associated documentation, we first looked at existing coding standards and systems. This included a review of SAS coding standards documents and of some R standards discussion (such as those suggested by Patrick Burns in his book "S Poetry"). The team at Mango Solutions include a qualified auditor and a number of developers who use more mainstream technologies (such as Java) so we were also able to look at their practices for guidelines.

Component Development Framework

The first step was to design a formal framework in which S and R development can be done. One of the primary goals was to ensure both S and R framework is carried out in a unified structure. This was vital, since much of our work has to be mirrored in both S and R. It also meant that we could build a single deployment and testing engine for our development components.

By design, R has a more sophisticated directory and build structure. Our design took this structure and extended it to ensure it would be compatible with S development, and to ensure that it captured all development needs. A (very) high level view of this directory could look like this:

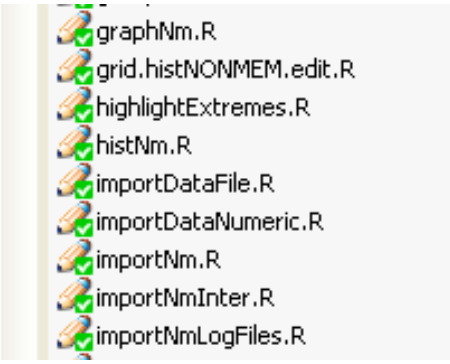
```
<CVS Project>      # CVS Project Root
|
├───Settings       # Project settings
├───Build          # Directory for building the library components
├───<Library1>     # Library component folder
├───<Library2>     # Library component folder
├───<Library3>     # Library component folder
├───Documentation  # Project documentation
├───WorkDir        # Working directory with setup scripts
└───Test           # Test directory
```

Each section of this has it's own formal structure that is governed by the CVS repository. For example, each "library component" folder contains a number of directory elements. Some of these elements are shown here:

Folder	Description
src	Source code to be used in the component build
man	Component docs and help files
inst	Files to be copied to the root component on finalisation
exec	Additional executables to be linked with this component
data	Data structures to be associated with the library
test	Component unit tests and test files
demo	Example scripts that exercise functionality

Function Meta Data

Each piece of functionality within the system has, at its core, a text file containing an S or R function. An example of this can be seen here:



By carefully defining metadata associated with each text file, we were able to develop an infrastructure where unit tests and help systems could be linked to each function.

Typically, this involves the dynamic generation of standard code headers. Using the ability to extract function information within S and R, we were also able to dynamically generate help templates. An example of a header for a standard R function can be seen here:

```
##############################################################################
# © Mango Solutions, Chippenham SN14 0SQ 2006
# Title:        mean.ssc
# Date:         06MAR07
# Author:       Rich
# Version:      1.0
# Email:        support@mango-solutions.com
# Description:  Calculates a mean of some data
# Requires:
# Keyword:      stats, stats:descriptive
##############################################################################
# <INTERNAL>
#   <TODO>
#     <DO Category="General" Priority="Low">
#          Consider changing default na.rm to "TRUE"?
#     </DO>
#   </TODO>
#   <ARGLIST>
#     <ARG>
#       <NAME>x</NAME>
#       <DEFAULT/>
#       <DESCRIPTION>The data to summarise</DESCRIPTION>
#     </ARG>
#     <ARG>
#       <NAME>trim</NAME>
#       <DEFAULT>0</DEFAULT>
#       <DESCRIPTION>Proportion to remove from ends of data</DESCRIPTION>
#     </ARG>
#     <ARG>
#       <NAME>na.rm</NAME>
#       <DEFAULT>FALSE</DEFAULT>
#       <DESCRIPTION>Should missings be removed?</DESCRIPTION>
#     </ARG>
#   </ARGLIST>
# </INTERNAL>
##############################################################################
```

This framework for our code allows systems to be built in the correct manner, and allows functionality to be linked directly to associated information such as help files and test suites.

Coding Guidelines

One major advantage of using S and R is the flexibility of the language. This results in a number of ways of coding in order to achieve the same goal. This can lead to issues, especially when more than 1 programmer is involved in a project, or if projects are continued at a later date. As such, a need to define style

guidelines was highlighted – the style guidelines were built using the Wiki as mentioned above.

This process started with a light-hearted discussion entitled "The S/R Commandments". This was used to promote the scheme and to allow people to initiate debates in an informal environment. An example of a set of "commandments" can be found here:

- Thou shalt not use any other assignment character but mine own ("←") ...
- Thou shalt not use poor object names ...
- Thou shalt not mask other objects ...
- Thou shalt not use global assignments ... ever …
- Thou shalt only put a single return statement in each function ..
- Thou shalt not take the logical names in vain (use "TRUE" instead of "T")
- Thou shalt always refer to arguments using the full name (eg. "na.rm" as opposed to "na")
- The first two letters of an argument name shalt distinguish it from all of the other argument names.
- Thou shalt document every argument inline with the code
- Thou shalt find out why a call takes more than 5 seconds, not guess
- Thou shalt try to use existing code as much as possible and not reinvent the wheel
- A function should only do one thing, if it does two things, then make 2 functions
- If you write 3 lines of code more than once, make a function

From this point, a number of style categories were derived, and Wiki discussions were set up for each category. The resulting discussion were very useful, and often heated! The development of these guidelines were aided by regular meetings to discuss each category. Eventually, the categories were "locked" and the quality documentation written. Following a number of reviews by both internal and independent parties, the standards were agreed and are in use for internal projects.

Development Reviews

During a project life cycle, code reviews are used to ensure the structure and style procedures are adhered to. Thanks to the formal structure, much of this process is automated using review scripts. This can be used to check basic measures against expected results, producing test review documentation and flagging any potential issues.

Testing Code

In order to test the code, a formal testing suite called "RUnit" was employed. This works much in the same way as standard unit testing software systems, and

allows the mapping of functionality to tests and the automated regression testing of component functionality. Careful checking versus functional requirements ensures test coverage is used to ensure the test suite is sufficient.

Improvements in this area (such as functional construction of traceability matrices) are currently under consideration, but the current system exceeds the guidelines set by regulatory agencies.

Ant Builder

The formal structure in place, together with the function meta information allows us to build project components in a controlled manner. We use an "ant" system to build system-ready library components for either S-PLUS or R. The "ant" process performs a number of tasks, including the following:

- Check project structure vs CVS tags
- Checking that the structure conforms to the required structure specified
- Use settings defined for configuration
- Dynamically build and compile the help systems by removing the help documentation from each function element
- Building external components such as dlls
- Automated regression testing
- Logging of build and component version numbers
- Copying additional file elements to the final component
- Zipping and labelling the final component

Conclusion

The use of S-PLUS and R in a pharmaceutical company provides new functionality for users. However, care must be taken to ensure these systems are correctly integrated with internal business practices.

Using the experience of internal and external S-PLUS and R consultants, we have been able to implement a framework for the development of S-PLUS and R components, which meets (and often exceeds) the guidelines set by the FDA and other governing bodies.