PhUSE 2014

Paper PP01

Reengineering a Standard process from Single to Environment Macro Management

Giuseppe Di Monaco, UCB BioSciences GmbH, Monheim, Germany

ABSTRACT

Statistical programming departments in pharmaceutical companies often have to manage huge macro libraries in an efficient and controlled way. On the one hand, it is important to deploy the macros in a flexible and user friendly way. On the other hand, macros and related documentation require an efficient method of validation and updates. With that standard, organizations in the pharmaceutical industry face the exact same challenges as any other software development house. In my presentation, I will demonstrate the benefits of adopting the Environment Macro Management approach and related reengineering steps that are needed.

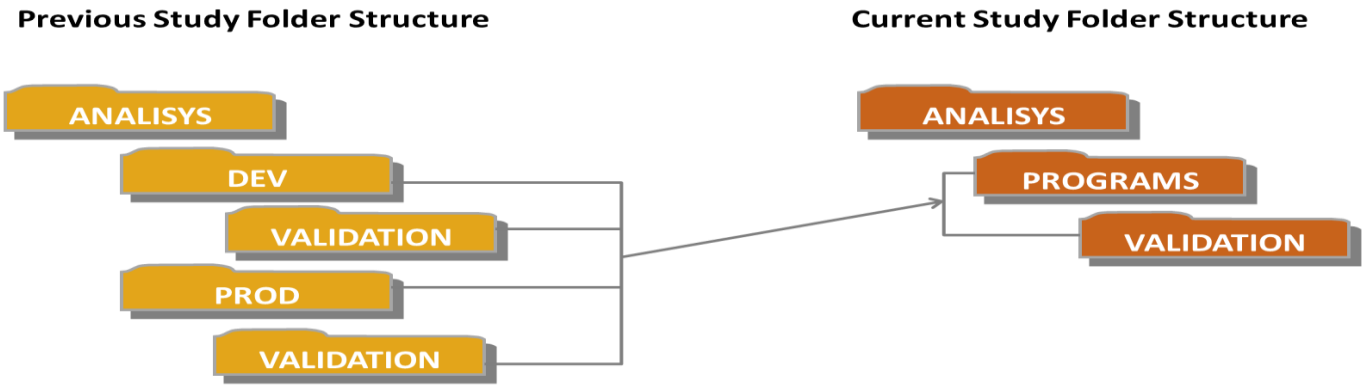
INTRODUCTION

With the implementation of standards in the clinical industry, companies often rush into process standardization without evaluating the overall standard projects. Being concentrate on a micro level rather than macro level can cause issues related to dependability and performance when facing unanticipated changes.

Most clinical companies have standard folder structures and some have also standard macros for clinical studies. Generally each macro of a clinical trial has its own dedicated location within the standard folder structure. A common approach across the industry is to focus on the single standard macro and on the standard folder structure as two divided entities without considering the fact that both could be part of a bigger project which I will call from now on the “Environment Macro Management” (EMM)

The EMM can be seen as a framework able to create sub-frameworks. In our case the intention is to create a package which facilitates the development and validation of study programs and includes standard folder structure, standard macros, non-standard macros and other software tools.

I was involved in implementing a major change to the study folder structure (as shown in the below image) and all standard macros, non-standard macros and programs needed to be re-validated. The below picture shows the change we made to the study folder structure. Under the analysis folder we had two subfolders, DEV and PROD. All study programs were developed and validated in DEV, then moved to PROD for production runs. In the new folder structure there is no DEV and PROD subfolder. All study programs are developed under PROGRAMS and validated within its VALIDATION subfolder. (In the new folder structure there is no physical copy or move anymore to another folder.)



Following the adoption of the new folder structure we had to answer the following questions related to the macros lifecycle:

- What is the impact on the lifecycle of our standard macros when changing our standard folder structure?
- Are these standard macros continuing to work as expected?
- Are these macros still working if we move them from their original locations to other locations?
- How we can retest/revalidate all standard macros in a fast and reliable way?
- How we manage dependencies between macros?
- Will our release process be impacted by this change?

I was chosen as a programmer for this task. As I was very new to the company I started with walk-through sessions with key individuals involved in the standard team. The first step I had to make was to understand what the previous macro management was. From these sessions it was clear that the previous validation process could not be used because the architecture of the macro environment was not reflecting the architecture of the study folder structure.

In fact, the previous macro management was based on a single macro management (SMM). In addition, automations were not used extensively. Major changes were introduced to each phase of the standard macro lifecycle and to its configuration management.

This paper will be focused on the software architecture rather than the coding implementation. It will describe the core activities of the software architecture design:

- Architectural analysis (understanding the current environment)
- Architectural synthesis (creating the new architecture)
- Architecture evaluation (requirements satisfaction)
- Architecture evolution (maintaining and adapting the new architecture)

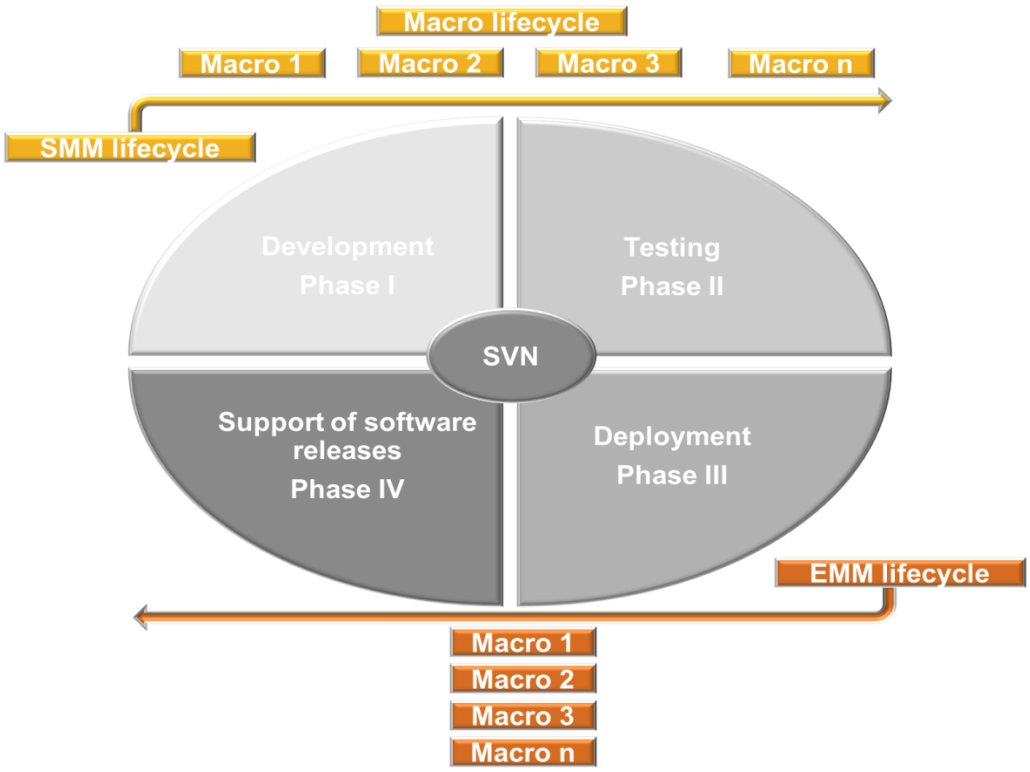
MACRO LIFECYCLE

SMM ENVIRONMENTS-PROBLEM

The Single Macro Management (SMM) was focused on a single macro. So, each macro was going through the 4 phases of the macro lifecycle. As a consequence the environment was replicated at each versions and so the number of files for programs and documents.

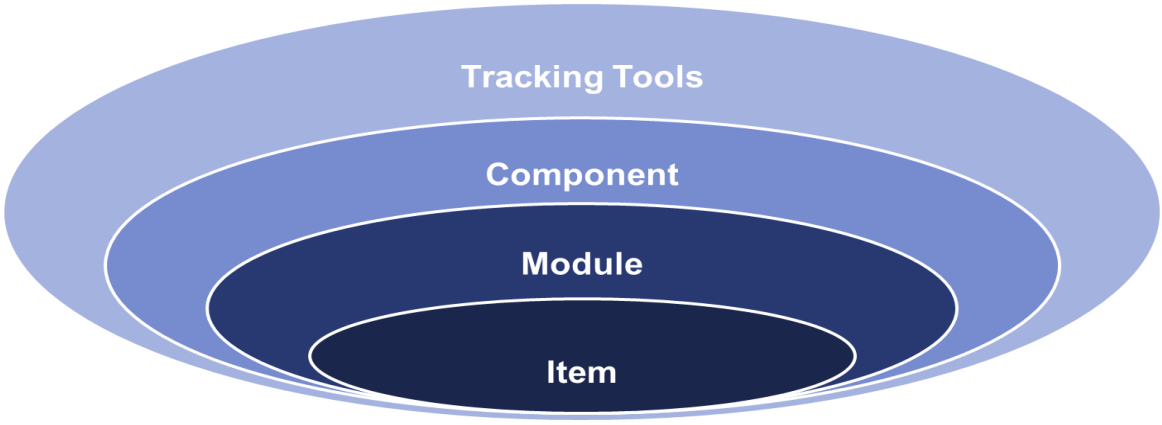
EMM ENVIRONMENTS-SOLUTION

Unlike the SMM, the EMM goes only once around the cycle for each version. This new environment takes the entire package only once around the macro lifecycle, therefore there isn't any replication of the environment. The below image show both SMM and EMM lifecycles. The EMM is able to create a release of a full validated package (with all macros, programs included) **with just one click.** This was possible by making the most of SVN subversion (freeware application used to control versioning of files), process automation and creation of configuration Management tools.



The EMM is composed by environments that are nested in an onion-like layered architecture. Each level uses the functionalities offered by its immediate inner layer and is used by its outer layer so that external levels are more abstract and easier to use.
Tracking tools offer a visual configuration management map for all our products. It includes (but not limited to) bug reporting,

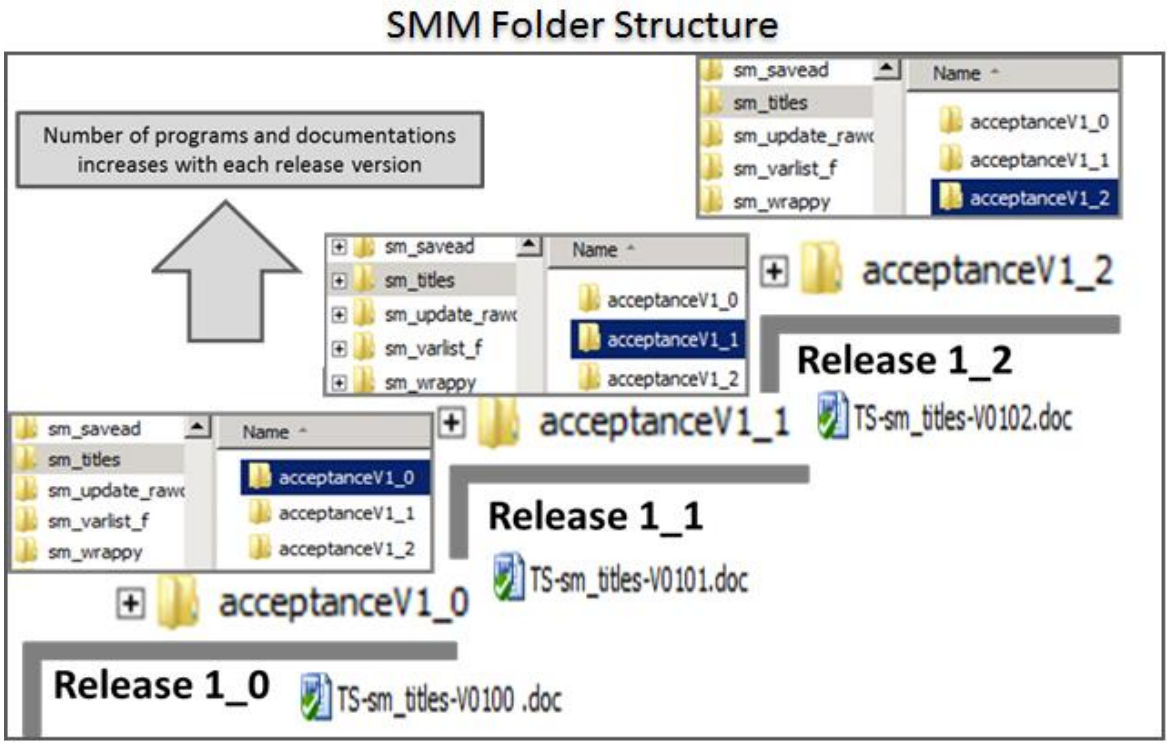
features/changes requests, test cases information such as (programs, inputs & outputs), dependencies, documents etc…
A Component is a set of modules and items designed to act together or in sequence. A single module is a smallest possible component. (e.g.: set of macros and programs to create Define.xml). A Module is an element to be part of delivery that can be tested as an entity. (e.g.: standard macro for checking iso dates.).
An Item is an element than cannot be tested separately and it is meant to be tested and delivered with a module or a component. (e.g.: any text or excel file which serve as an input for modules and components.



MACRO ENVIRONMENT FOLDER STRUCTURE

SMM FOLDER STRUCTURE-PROBLEM

The structure of the SMM affects all phases of the macro lifecycle, and it is the cause of the replication of the environment. Each standard macro had its own folder structure. The image below shows the folder structure under the macro name. The name of the macro is the root folder and it contains as many branches as the number of the macro version released.



EMM FOLDER STRUCTURE-SOLUTION

The structure of the EMM does not cause the replication of the environment at each version. At one point in time there is only one file for each macro and for each document. All previous versions of these files are kept in SVN server. Also, the EMM has a hierarchical structure which allows easy navigation of the environment. In order to avoid working with two different environments, all content of the SMM environments (folder structure, macros, test cases and documents) were revised and then copied in to EMM.

The EMM, consist of two main roots:
One of them is called source_package and its subfolder trunk contains our deliveries plus all other tools needed for their creation and

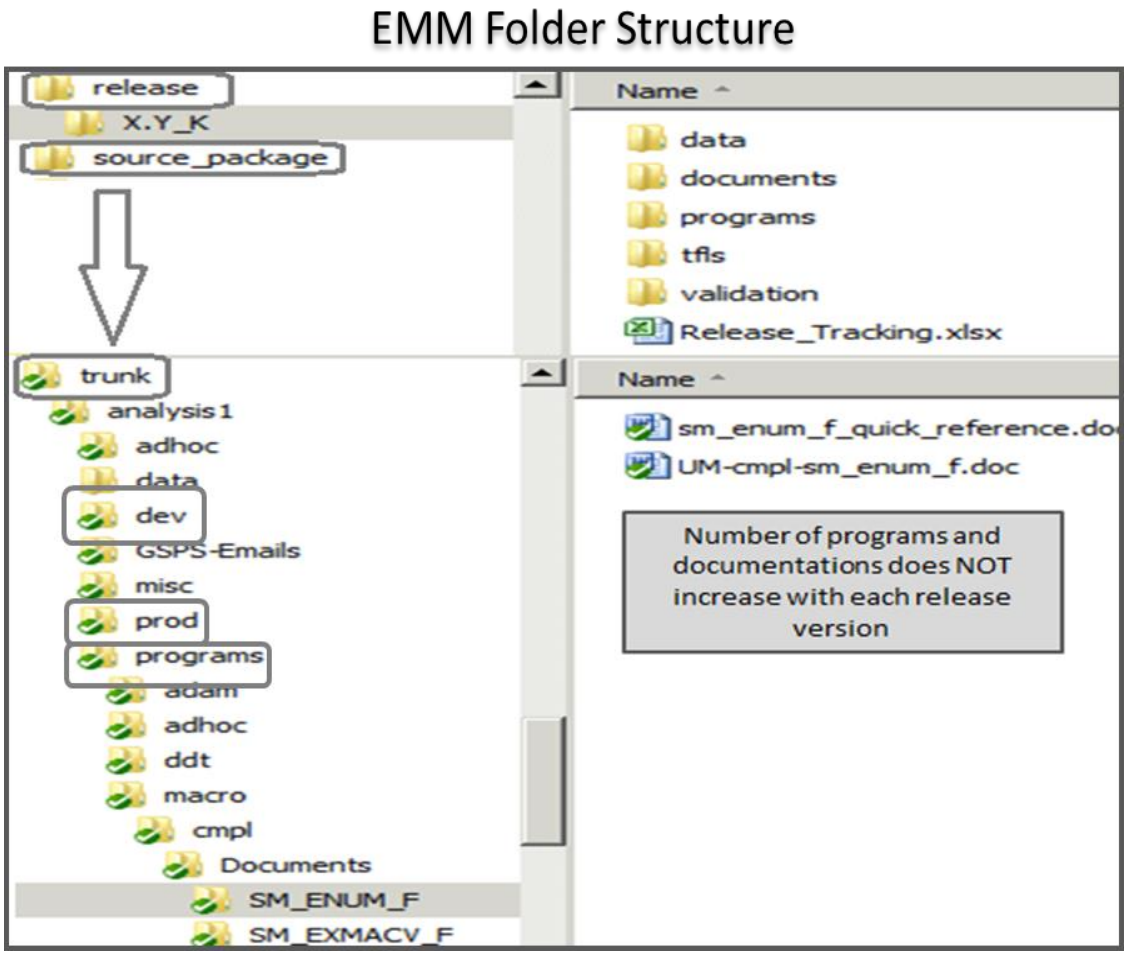
maintenance. In addition, in order for our macros to have backwards compatibility, the EMM contains both, old and new folder structures (see dev, prod and programs folder under analysis1). Hence, each macro is validated against both folder structures.

There are also additional locations under the root location trunk. These locations are needed for the implementation of all 4 phases of the macro lifecycle.

The entire trunk folder is backed up in the svn server; therefore, all previous versions of the trunk can be recreated when needed.

The other main folder of the EMM is the Release folder. It is a point in time on the trunk we wish to deliver to our study programmers. It is a sub-environment of the EMM and it is created by tagging (in SVN server) a subset of the trunk folder. (We deploy a subset of trunk because we are only interested in delivering macros and documents needed for our study programmers and not all other internal programs and tools needed for their creation and maintenance.)

NOTE: Tagging is a lightweight process; nothing gets physically copied in the svn server. Tags are just markers saying "these files and folders were copied at this revision".



REGRESSION TESTING OF TEST CASES

Test cases are test scripts, usually short SAS® Software programs designed to ensure that the functionality of the standard SAS program meets the specifications. Regression testing determines whether a change in one part of the software affects other parts of the software.

SMM REGRESSION TESTING-PROBLEM

In the SMM, regression testing was not automated. The dependency between macros, creation and instruction on how to run test cases, evaluation on re-running previous test cases were done manually by standard programmers. Therefore, it would have taken months to manually execute all previous test cases and make sure all outputs were as expected without automation of regression testing. In addition, some test cases were testing more than one specification and this had a performance issue on debugging activity.

EMM REGRESSION TESTING-SOLUTION

EMM Regression Testing is fully automatic; each test case does not usually test more than one specification. For this reason some test cases which were developed in SMM were modified and sliced up to test only one specification. This facilitates the implementation of the automation of the regression testing and ensures an easy maintenance. Here there is no evaluation on how to run previous test

case because once a test case has been created it goes in to the queue of the automation and runs automatically. The dependency between macros is not an issue anymore since the entire package gets tested and released at once. In addition, the automation of the regression testing allows both interactive and batch running testing.

The first run of the automation of the regression testing is the reference start point from which the subsequent run is compared to. The entire concept of the automation is based on comparing outputs such as PDF and parsing listings and logs for expected results.
Regression testing runs all previous test cases in batch. When all test cases have produced their listings and logs, these are loaded back in to SAS and parsed for expected results. If the content of the listing and logs is not as expected, an output is produced pointing to the exact test case where the mismatch has been found.

EMM REGRESSION TESTING: IMPLEMENTATION FOR LOGS AND LISTINGS (EXPECTED RESULTS)

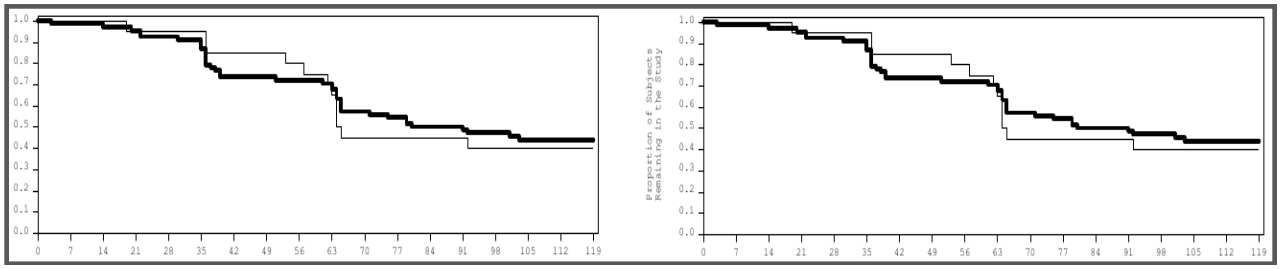
The parsing logic for listings and logs consists of 4 parts (see next image). The first part is the test case number, the second part tell us in which mode the test case should produce the message (batch or interactive mode for P or D respectively). The third part is the message itself which is expected to be found or not in the output. Finally, the forth part is a flag which confirms the presence of the expected message.
Unlike the SMM where there are multiple versions of the same macro, in the EMM there is only one file at the time for each macro. This is the reason why the flag for expected message was introduced. (Having one version of the macro at the time makes it impossible to replicate old bugs and changes in functionality without this flag.)

An example of expected test result mechanism is shown in the next image.
The first line means that the message is expected (Y) to be present in the log of the test case (TC01) when runs in interactive mode (D). The second line means that the message is not expected to be present (N) in the output of the test case (TC08) when runs in interactive mode (D).
The third line means that the message is expected (Y) to be present in the test case (TC08) when runs in batch mode (P).

```
TC01D|WARNING: The variable AVAL in the DROP, KEEP, or RENAME list has never been referenced.|Y
TC08D|ERROR 71-63: The value specified for the LABEL option is too long.|N
TC08P|ERROR: Macro SPATH must be either in SVN1 or SVN2 structure|Y
```

EMM REGRESSION TESTING: IMPLEMENTATION FOR PDF (EXPECTED RESULTS)

Checking consistency of PDF files is different than checking consistency of logs and listings; these files can't be loaded in to SAS and therefore the comparing between current and previous outputs is done by an external application.



EMM REGRESSION TESTING: IMPLEMENTATION FOR LOGS (UNEXPECTED RESULTS)

Another important automation is the check over the unexpected results. The parsing of expected results does not cover scenarios where we can have unexpected errors, warning etc. For this reason a module called "Auto Bulk Comparison of the unexpected result" was developed. This is a SAS tool that compares the distinct count of the current messages (errors, warnings etc.) against the distinct count of messages of the previous run. If the count is not equal then a message is output highlighting the exact test case where the mismatch has been found.

TRANSITION FROM SMM TO EMM FOLDER STRUCTURE

EMM TEST CASE GROUP

As we have seen previously, unlike the SMM where we have a root folder for each version of a standard macro, the EMM contains only one unique root folder (trunk) containing the latest version of all macros and documents. Having only one root location has the downside of having also unique inputs and outputs location for test cases. In order to overcome the problem of overwriting inputs and outputs, a concept of test case group was introduced.

A test case group consists of:

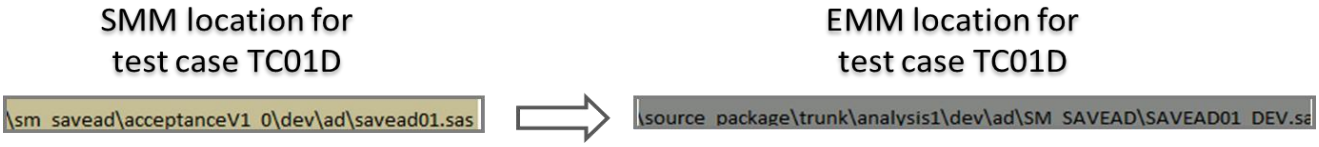
- Test case program
- Test case log & reference log
- Test case listing & reference listing
- Test case inputs
- Test case outputs & reference outputs

Each Test Case Group has its unique code. The code has the following structure: TCXXY where XX is the two digit number which uniquely identify the test case, Y is a letter (D or P) indicating whether the test is meant for interactive or production run mode. Each test case group has its own unique folder structure.

EMM TEST CASE GROUP-TEST CASE PROGRAM

In the EMM, naming conventions were chosen to promote consistency and avoid ambiguity. Also, having naming conventions facilitates the automations. For this reason all test cases and relative documentations were renamed according to the convention definition. For test case names the following convention was chosen: <macro name><test case number>_<running mode>.sas where running mode could be either DEV or PROD for interactive or batch mode respectively. For example, savead01.sas from SMM in EMM would become SAVEAD01_DEV.sas for interactive running mode and SAVEAD01_PROD.sas for batch running mode. (NOTE: logs and listings names are the same as the program with the exception of extension.)

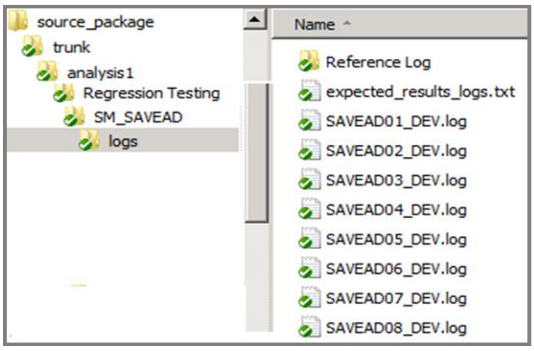
Also, since all macros, test cases and documents were copied in to the EMM, a mapping table was created in EMM to keep track of the original locations of these files. This mapping table could also be useful for audit purposes.



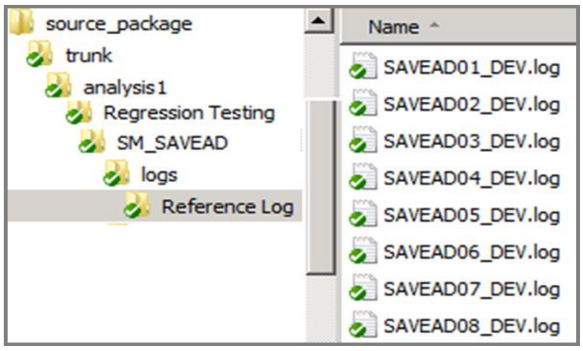
EMM TEST CASE GROUP-TEST CASE LOGS AND REFERENCE LOGS

Logs and reference logs names are the same as the test case programs with the exception of the extension. This facilitates automations. Two version of logs are kept locally, (all other versions are kept in SVN server) logs and reference logs. Logs are always compared against reference logs to ensure consistency across consecutive runs. If after running regression testing all results is as expected, then all logs are copied in to reference logs before next run happens.

Current Logs



Reference Logs



EMM TEST CASE GROUP-TEST CASE LISTING AND REFERENCE LISTING

Listing and reference listings follow exactly the same structure and naming conventions choose for logs and reference logs.

EMM TEST CASE GROUP-TEST CASE PDF AND REFERENCE PDF

Pdf and reference PDF follow exactly the same structure as the logs and reference logs.

EMM TEST CASE GROUP- OUTPUTS AND REFERENCE OUTPUTS

Below image shows the difference between SMM and EMM output folder structure for the test case TC07D of the standard macro SM_SAVEAD. In SMM the test case creates datasets adsl.sas7bdat and adpe.sas7bdat under analysis1/data/adam.
In EMM the same test case creates the same datasets but this time under gsp/analysis1/data/adam/SM_SAVEAD/07D

EMM TEST CASE GROUP-TEST CASE INPUTS
Test case inputs follow the same structure as the test case outputs.

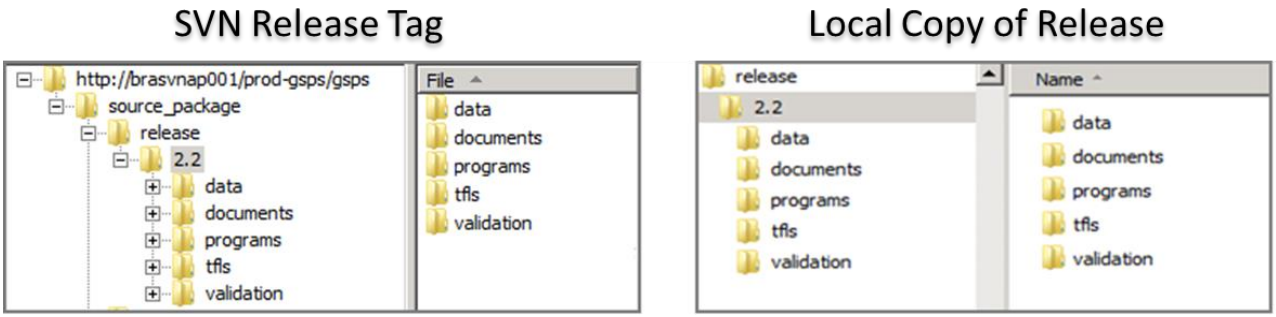
SOFTWARE RELEASE MANAGEMENT PROCESS

SMM RELEASE MANAGEMENT-PROBLEM
Release management in the SMM did not offer an adequate level of automation and relationships between macros were difficult to maintain since each macro was released separately.

EMM RELEASE MANAGEMENT-SOLUTION
Release management in EMM is fully automated. The EMM package can be seen as the container to ensure that changes are packaged, released and tested in a repeatable and controlled manner. Hence, releasing a package instead of a single macro also offers a good container for managing relationships across macros.
Once all tests (like the regression testing) have passed, release documents are created together with a backup of the release on the in SVN server. Note: only the package meant to be released is backed up and not the entire environment. However, both package and environments can be retrieved at any time since both are under SVN server.

SMM RELEASE TO LOCAL SERVER-PRODBLEM
This process was done automatically, but the process was not simple and easy to maintain.

EMM RELEASE TO LOCAL SERVER-SOLUTION
The creation of the package release to the local server is also fully automated, and the entire process consists of a small batch file running SVN commands. Once the release folder is created, we have a full configuration of a study set up. As previously mentioned, the release folder is a subset of the trunk folder. Before creating the package locally, a tag of this package is created in SVN server as the below picture shows.



SMM RELEASE TRACKING-PROBLEM
Release tracking was not cumulative across versions.

EMM RELEASE TRACKING-SOLUTION
The release tracking is the main source of information for release deployments. It is a cumulative file which keeps track of release across versions; therefore, it contains information about current and previous releases. This document is distributed together with our package release. For each release of the package a new sheet is created and populated with information related to each released file. It contains info such as location of residing file within the package, last changed author, last changed revision, last changed date etc.., and in this way we are able to identify from the deployment all related software that was required in that release.

EMM Release Tracking

Path	URL	Last_Changed_Author	Last_Changed_Rev	Last_Changed_Date	Name	type	RELEASE_FOLDER	RELEASE_REPOPATH
macro\cmpl\spath.sas	asvnap001/prod-gsps/gsps/source package	U049851	5417	2014-01-27 09:11:39 +0100 (Mon, 27 Jan 2014)	spath.sas	VALIDATED	M:\Sites\Braine\Biocdm\CMPL\SVNReleaseLibrary\source_package\release\2.2.6288	http://brasvnap001/prod-gsps/gsps/source_package/release/2.2.6288

SMM and EMM PACKAGE DEPLOYMENT TO STUDY LOCATIONS

In EMM this step has been kept as it was in SMM. The deployment is mostly manual at this time because we want to give us as much flexibility as possible to study programmers. In this way they are able to choose to upgrade their version of the macros to a newer version or to keep legacy versions. In other words, when a package has been released, a single macro can still be copied from the release area down to the study area. (This may be necessary especially if a study is near to closure and there is a need for keeping all the other macros)

SMM RELEASE MANAGEMENT-RELEASE VERSIONING-PROBLEM

Since each macro had its own lifecycle in SMM, each version of each macro had its own hardcoded statements shown in the below image. In addition, each macro had its own version, so at one point in time we could have had two macros with two different versions. This can cause compatibility issues if macros are dependent on each other. Also, each time there was a new version of a macro, the hardcoded part which was printing the version info had to be updated.

```
%put STANDARDS GROUP DEVELOPMENT - sm_savead (v01.00);
```

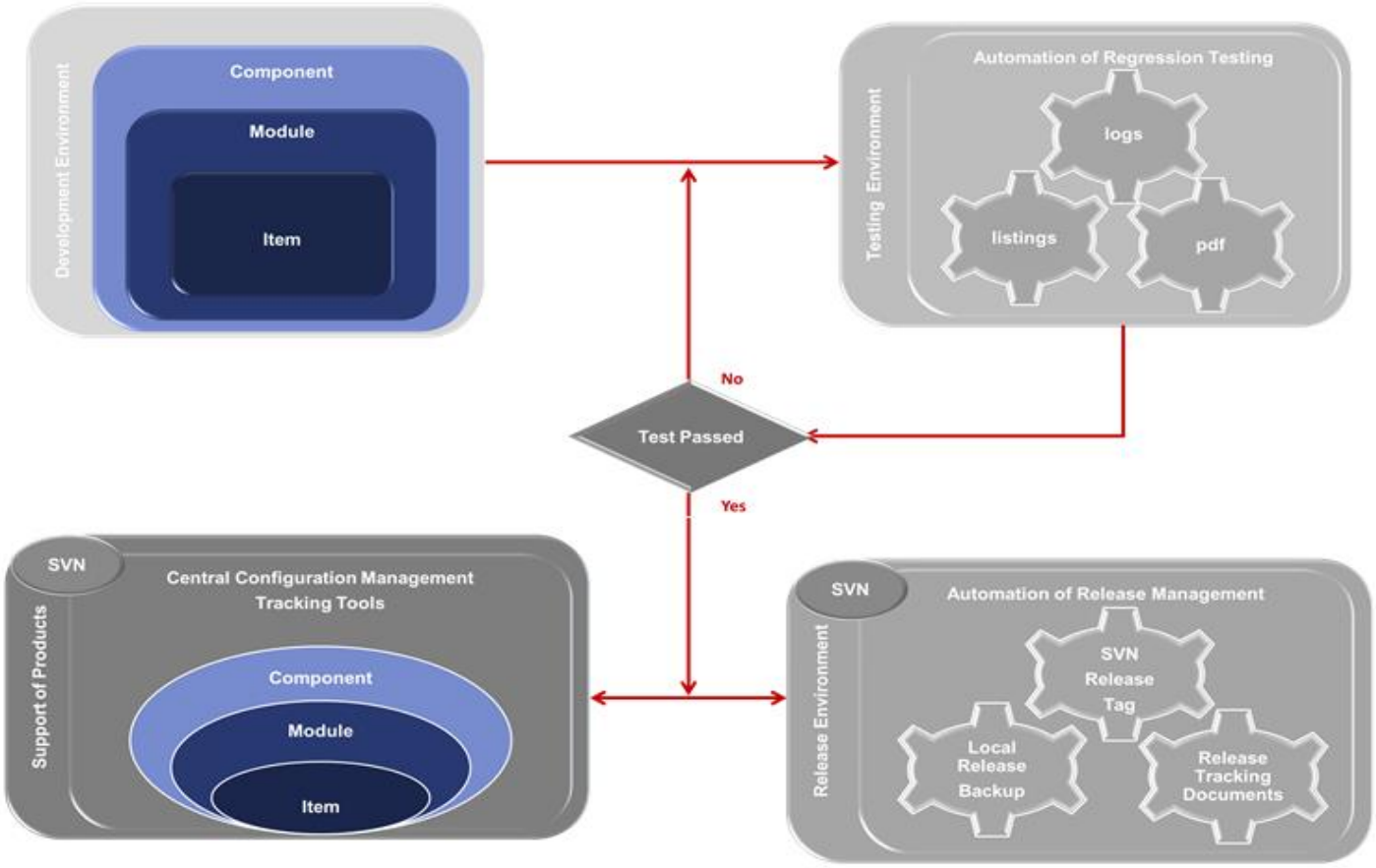
EMM RELEASE MANAGEMENT- RELEASE VERSIONING-SOLUTION

In EMM the versioning is not on the single macro anymore but it is on the entire package. Therefore, compatibly between dependent macros is not an issue anymore. Also, there are no hardcoded statements within the macro code; all information about current release version is taken directly from the Release Tracking file and printed to the log when the main macro “spath” runs.(“spath” define the search path to “autocall” macro libraries including both standard libraries and study-specific libraries)

```
-------------------------------------------------------
-------------- Macro package release ------------------
-------------------------------------------------------
NAME                RELEASE_VERSION    SVN_REV_NUMBER
sm_savead.sas             2_2                6014
```

EMM PROCESS FLOW

In EMM all products are developed exclusively under the root trunk, and its history is always ensured by Subversion (SVN), hence this location always contains the most up to date version of our products but may also be the most unstable version! Another choice would be splitting trunk in branches, implement changes within these branches then merge the those changes back into the trunk when branches have proven to be stable and working. In EMM all previous outputs created by the test cases (logs, listings, pdf etc...) are automatically checked and compared against expected results. The automation will check if the expected results match the actual results. Testing Environment is also under Subversion, and the working version always contains the latest run of the regression testing automation. All Information related to the support of products is centralized within the tracking tool. Its main functions are to provide information regarding the status of each macro, the relationships with other macros and all related information which allows for things like the reconstruction of such macros, at any point in their existence. The EMM package release can be seen as the container to ensure that changes are packaged, released, and tested in a repeatable and controlled manner. In EMM all products are released at once as a package. SVN server release back up, local back up and related release documents are created automatically.



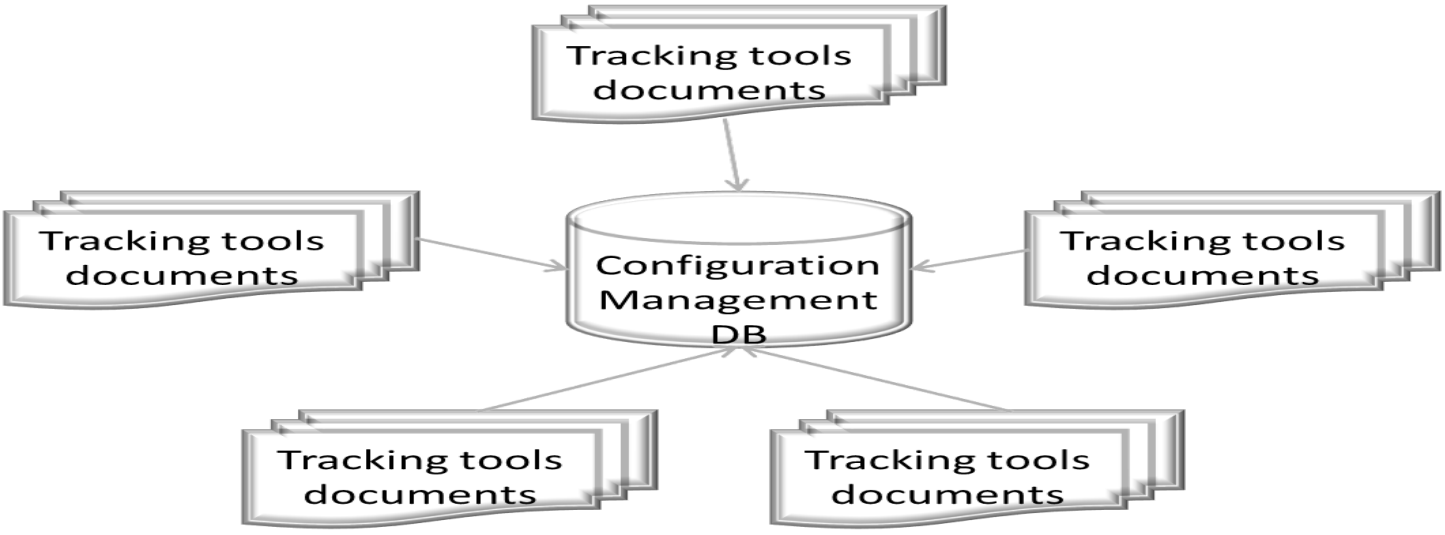
EMM FURTHER IMPROVEMENT

CONFIGURATION MANAGEMENT DB

The implementation of this component is still under construction. It will merge information from different sources such as release notes, test case documents, tracking tools, specification documents, release tracking etc…

The configuration management database should identify for each product that is built and released required software and configuration. Therefore, the release package can either be configured by all the documentation as I have done or by having a configuration management database.

Its main functions are to provide information regarding the status of each macro, the relationships with other macros and all related information which allows for things like the reconstruction of such macros, at any point in their existence. Another positive aspect would be the possibility of performing impact analysis in the case of change management or root cause analysis.



CONCLUSION

This paper has briefly highlighted the weakest parts of the SMM and offered solutions through the implementation of the EMM. This was achieved by putting together software engineering concepts and free tools such as SVN subversion. The implementation of the EMM has drastically reduced the time for testing, validation and release. It has also improved robustness and increased consistency of output, productivity and quality.

REFERENCES

A Guide to Deploying Subversion for Version Control of SAS Programs in the Pharmaceutical Industry by Tim Williams, UCB BioSciences Inc, Raleigh, USA
http://www.phusewiki.org/docs/Conference%202013%20AD%20Papers/AD01.pdf

Subversion Best Practices
https://svn.apache.org/repos/asf/subversion/trunk/doc/user/svn-best-practices.html

Strategies for Repository Deployment, Prev Chapter 5. Repository Administration by Ben Collins-Sussman, Brian W. Fitzpatrick, and C. Michael Pilato
http://svnbook.red-bean.com/nightly/en/svn.reposadmin.planning.html#svn.reposadmin.projects.chooselayout

ACKNOWLEDGMENTS

This paper (and the implementation of the EMM) could not be written to its fullest without the collaboration of my colleagues. Without their continued efforts and support, I would have not been able to bring my work to a successful completion.

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