Paper AD08

Automated Test Framework for SAS Macros

Sven Prasse, Entimo AG, Berlin, Germany

ABSTRACT

To resolve the conflict between modern programming paradigms like agile software development or DevOps and implementing software for controlled and validated environments, it is helpful to automate software testing.
The presentation shows how a SAS framework supports the development of SAS Standard macros based on automation of

- Test execution,
- Collection and archiving of test results,
- Comparison of test results against reference results, and
- Provision of summaries and error reports.

My experience shows that an automated test framework allows easy and fast evaluation, via regression analysis, of the risk of a software change. Furthermore, an automated test framework enables the software developer to ensure short implementation cycles and also to ensure that the delivered software shows the required behaviour. Automated testing shortens the testing effort and speeds up the development cycle without any decrease of software quality.

INTRODUCTION

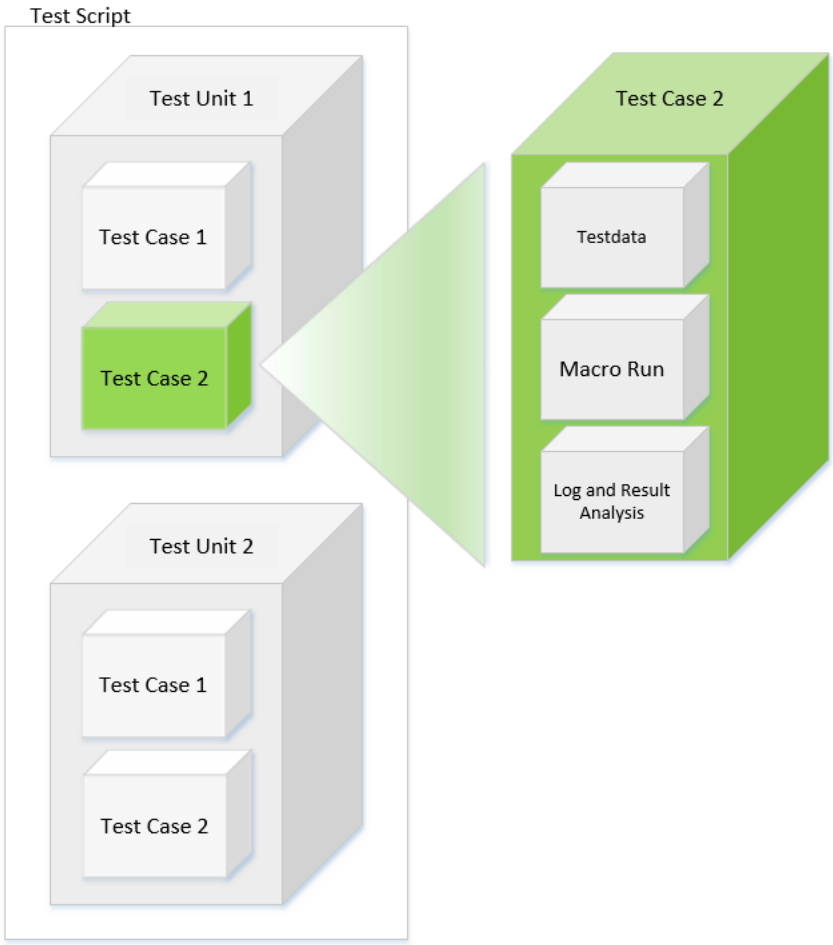
Agile software development and DevOps are modern concepts with the goal of bringing software into production as soon as possible. In nightly (or faster) builds new versions are deployed. Without any automated testing, it is not possible to guarantee a high level of software quality.
We have used and developed a test framework for macros in my company over many years. This paper explains the main principles of automated tests for SAS macros with the background and experience of our test system.
To explain every functionality and the coding of it in detail would exceed the scope of a single paper for PHUSE, but it can give a short overview of the main modules and a brief description of how to build automated tests.

OVERVIEW

To test a SAS macro the test frame has to check the log and all outputs of the test objective. The test run is successful if the outputs meet the expected results, and the log is clean and contains the expected information. To check all required functionality, it is necessary to run the macro with different parameter values and test data constellations. The processing of one call is called a test case, and a bundle of different test cases is called a test unit.

For the automatic test framework, a test case needs to pass the following steps:

- Prepare the test data
- Run the macro (test objective)
- Catch the log and results
- Search the log for triggers
- Compare the results



TEST ENVIRONMENT

To be as flexible as possible, the main design principle for our test macros is to minimize the necessary preconditions and the requirements for the environment. We have reduced the technical requirements for the environment down to:

- Catalogue with the compiled test macros
- Few configuration metadata (delivered as SAS datasteps)
- A start script to load the catalogues and prepare the metadata
- The test script (as a SAS program) with the coded test cases

This lean design avoids problems with side effects and incompatibilities with the test objectives and requirements of the environment that the test objectives have to run in.

MACRO RUN

The test framework redirects the log output directly before the test objective run and saves the log to analyze it later. Furthermore, the different outputs of the test objective have to be stored and compared. The different output types of a macro can be:

- Datasets
- Text files (e.g., listings and tables)
- RTF Files
- Graphics (e.g., png Files)

The test framework should implement a logic (naming convention, directory structure) to store the evidence of a specific test case run. This is needed for validation proposes and for the tester to analyze any noticeable problems in a specific test case.

LOG ANALYSIS

To automatically decide if a test of a SAS program runs successfully it is necessary to analyze the log. On the one hand, the log has to be clean in terms of SAS-specific technical driven messages, and on the other hand the required specific messages have to be displayed or should be not displayed for the test. To satisfy these needs, the log parser analyses the logs for:

- Overall unexpected strings (e.g., "ERROR:")
- Test-case-specific expected strings (e.g., "Mandatory check failed: Age of Subject 123 is missing")
- Test-case-specific unwanted strings (e.g., "Value of variable XYZ truncated")

RESULT ANAYSIS

The test framework has to provide different comparison methods for each output type. For a SAS dataset it is easy, because PROC COMPARE can be called and compares the meta data (format, length and types of columns) with the content of the dataset. But for png files it is much harder, because two png files showing the same graph may differ in the binary representation. This may happen if the two graphs are created with different graphic drivers on different machines. Providing solutions for these more complex types is the main challenge for the implementation of the test framework. This will be considered later in detail.

TEST CASE IN DETAIL

The automatic test case framework implements macros for each step within a test case. They could be considered much more for very specific purposes. For a standard test, our test script looks like the following.

The first macro initializes the test case and prepares all internal datasets, initializes the needed internal variables, and prints the test case title in the log

```
%NEWTESTCASE('##A','Missing or partial start dates ');
```

The NEWTESTCASE macro initializes the test case and prints a message to the log. Directly after the message, SAS code can be provided to manipulate test data for this particular test case.

```
%TESTDATA();
OPTIONS SOURCE;
DATA inae_testcase_1;
  SET TestData.ae_tnc;
  WHERE USUBJID in ('TEST_0001_0000000001','TEST_0001_0000000002');
  IF substr(AESTDTC, 1, 10) = "2002-06-24" AND AESEQ = 2 THEN DO;
    AESTDTC = "";
  END;
RUN;
OPTIONS NOSOURCE;
```

The MACCALL macro just prints massage in in the log. A global variable MACROCALL has to be filled with the test objective call included with all parameters. The macro cannot be called directly (e.g., %MACRO2TEST) because the framework has to redirect the log immediately before the test objective runs. That means the test framework has to call the test objective (e.g., %&MACROCALL;) later on

```
%MACCALL;
%LET MACROCALL = MACRO2TEST (INAE = inae_testcase_1
          ,OutAdamAE = outref.ADAE_TESTCASE_1);
```

With the message of the expected results, the framework marks the start of the section where the test-specific messages are stored, and parses this log part later on to identify the strings to search for in the log analysis algorithm.

%***EXPECTRESULT***;
PUT A clear log is expected;

Now the system starts with the run of the test objective by calling the macro TESTRESULT.

%***TESTRESULT***;

The COMPARE_SAVE_REFERENCE macro system compares the results form the test run provided in the library baselib with the references in the library complib.

%***COMPARE_SAVE_REFERENCE***(baselib = LibRef, complib = outref, new = ADAEPL);

The ENDTESTCASE macro starts the log analyses, puts the different log parts together, replays them from the temporary file in the SAS log screen, and provides a result summary for the test case.

%***ENDTESTCASE***(check4UnexpectedLogMsg=Y);

LOG ANALYSIS

DEFINITION OF MESSAGES

The "expect result" section of the test case is used to print a verbal description and the definition of log text to search for with put statements. A log parser assumes every text included in brackets is a string that has to be shown in the log during the test execution. The elegance of this solution is that the final log contains all information. No external meta data storage holding the test case related strings for which the frame work searches is needed or has to be maintained.

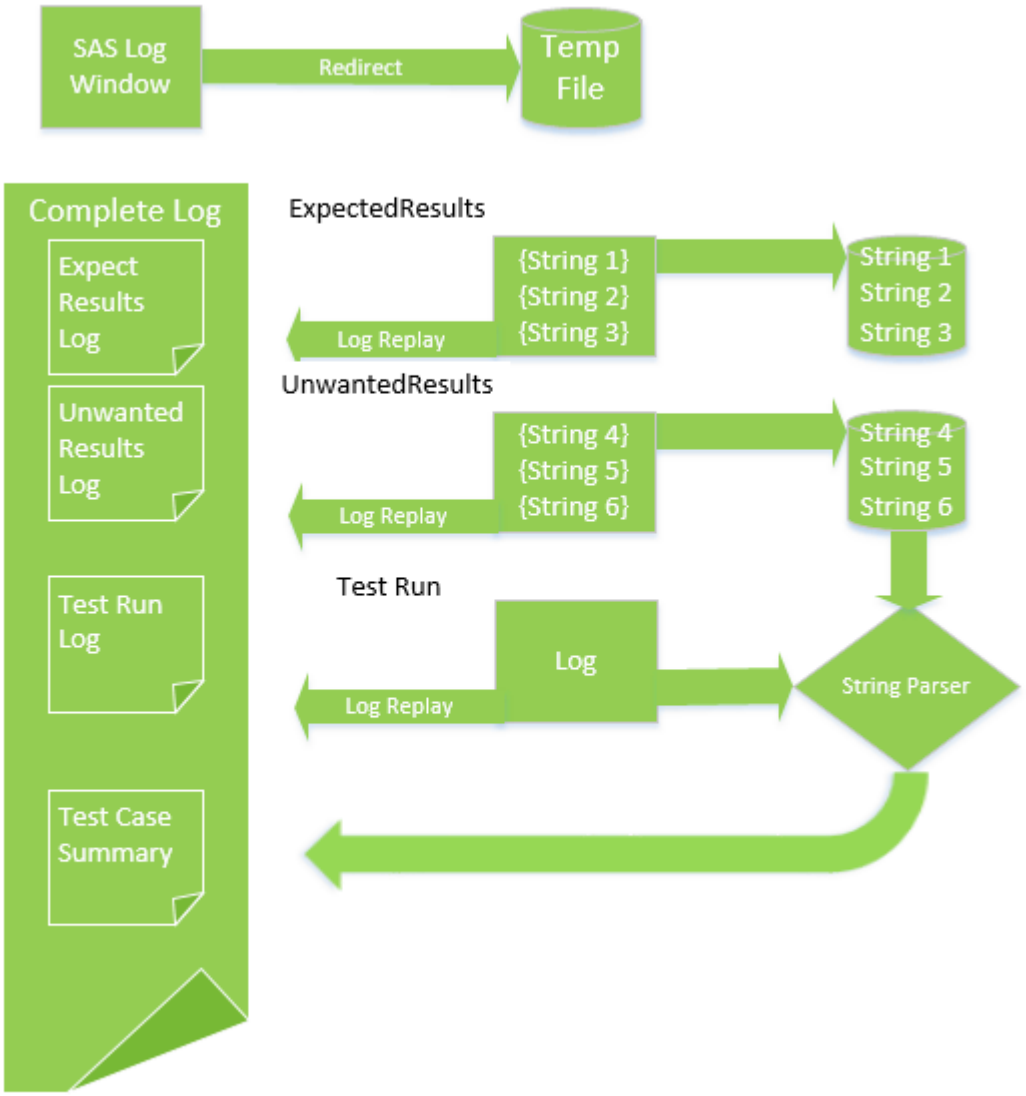
```
%EXPECTRESULT;
%PUT A table of the subjects per study with missing or partial dates is shown;
%PUT {WARNING: Missing or partial start dates in the following AEs:};
%PUT {*** STUDYID    USUBJID          AESEQ};
%PUT {-------------------------------------------------------------------};
%PUT {*** TEST-0002   TEST_0002_0000001001  1};
```

The "not wanted result section" defines, in the same way, the string that a log of a successful shall not contain. This section is started with the macro NOTWANTEDRESULT.

```
%notwantedresult;
%PUT {Value of variable XYZ truncated;}
```

PARSING DIFFERENT LOG PARTS

The log is the main evidence for the test script run and should contain all information in a human-readable form. In addition to normal PUT Statements to include descriptive comments for the test case, the complete log from the running test objective is also included in the log. Below this part, the log shows the result from the automated comparison and a test case summary.



With the start of the test case, the test macros take control over the log and redirect it to a temporary file.
The test framework identifies the defined strings in the EXPECTRESULT and NOTWANTEDRESULT sections of the log with brackets and stores them in temporary datasets, and replays the content back to the log. At the end of the test case run the macro parses the log of the test objective (and only this part) to look for the test-case-specific strings stored in the temporary datasets, and for overall strings that are unexpected in general and defined in an Excel sheet for all test cases in the test script.
At the end of the test case, the result of the log analysis will be printed in the log, and all log parts are replayed back to the SAS log. See the following log example of a test case log for details:

```
*  Start of Test Case number C1A_1
*******************************************************************************
*  MACRO2TEST Unit C1A ' Missing or partial start dates'
*  Test case 1
*  Default call with mandatory variables
*******************************************************************************
Test data:
----------
Macro call:
-----------
MACROCALL = MACRO2TEST (INAE = inae_testcase_1,OutAdamAE = outref.ADAE_TESTCASE_1);
Expected result:
----------------
A table of the subjects per study with missing or partial dates is shown;
{WARNING: Missing or partial start dates in the following AEs:};
{*** STUDYID    USUBJID            AESEQ};
{-------------------------------------------------------------------};
{*** TEST-0002   TEST_0002_0000001001  1};

Test result:
------------
*************************************************************************
*           YOU ARE RUNNING THE FOLLOWING MACRO                 *
*************************************************************************
* MACRO : MACRO2TEST                    VERSION = 2.3        *
*************************************************************************
++++ Starting parameter check
++++ Starting data-build

WARNING: Missing or partial start dates in the following AEs:
*** STUDYID    USUBJID            AESEQ
-------------------------------------------------------------------
*** TEST-0002   TEST_0002_0000001001  1
*******************************************************************************
++++++    Analysis dataset OUTLIB. ADAE_TESTCASE_1 is created successfully.
*******************************************************************************
++++++    MACRO2TEST has finished
*******************************************************************************
<reference data C1A_1_ADAE_TESTCASE_1 were restored from corresponding export file (.exp) in LibRef>
Comparison of data sets LibRef.C1A_1_ADAE_TESTCASE_1and outlib.ADAE_TESTCASE_1:
*** No differences
<see dataset C1A_1_ADAE_TESTCASE_1in libref outlib>
*******************************************************************************
*  End of Test Case number C1A_1
*  Automated check of test results (case insensitive): SUCCESSFUL
*******************************************************************************
```

RESULT COMPARISON

The comparison of the results is more complex than the log scans and depends on the output format of the results. The example above compares the results of the macros with the macro COMPARE_SAVE_REFERENCE. This specific macro deals with datasets. The framework contains more macros, each specialized for a specific data format of the results. Each compare macro compares the resulting artefact with stored prepared references.

To have a high degree of flexibility, the framework can be customize with an Excel sheet called control matrix. Here we place all rules to rename the results for storing them after the results are compared. This is needed to prevent overwriting relevant result data with the results from the next test cases.

DATASETS

The simple data set comparison is not much more than a PROC COMPARE of two datasets. If the stored reference has no difference in the data set or meta data (variable names, formats, labels, length and types), and every observation contains the same values, then the test runs successful.

TEXT FILES

Text files are, in the case of SAS macros, mainly outputs such as listings or tables. In principle, the files can be imported into a dataset and then compares by algorithm, the same as datasets. However, there are some specialties of outputs. For instance, outputs often contain the current date (e.g., in the footnote). It is not comfortable to have to update this date inside of each references before the test run can be run successful. Therefore, the test framework contains a global matrix where it is possible to specify strings that are excluded from the comparison. In combination with wild cards and regular expressions, it is possible to describe a string that contains the line where the date is positioned and put the date as a combination of wildcards corresponding to the specified date format. If the comparison routine finds such a string that matches the pattern, it will test it positively, and the test case will be successful.

```
Respiratory, thoracic and          0     2     0     0     0     0     0
mediastinal disorders
 Epistaxis                         0     2     0     0     0     0     0

FOOT1
FOOT2
FOOT3
FOOT4
FOOT5
FOOT6
FOOT7
FOOT8
Containing data from studies TEST_0001 and TEST_0002                    27SEP2016
```

For instance, the current date in the last footnote in the right lower corner of a table ("27SEP2016" in Line 52 on Position 142, in the shown example) can be excluded with the following definition in the control matrix in various ways:

- Solution 1: Define the mask relative to the value itself
- Solution 2: Define the mask relative to a static string
- Solution 3: Define the mask as a complete line

String	Row from	Row to	Start	End	Compare blanks	Complete Row	Dstype
SEP20			i-2	i+6	No	No	LST
Containing data from stud			i+(142-26)	i+(142-26)+9	No	No	LST
	52	52			No	Yes	LST

For more complex text files where whole parts are not of interest, the test frame work implements an algorithm that certain parts of the file are masked. That mean that only parts that fit the mask are relevant for the file comparison. The more complex routines are quite often not used. With the normal text file and dataset comparison, more than 95% of test objects can be tested very comfortably.

GRAPHICS

For the png format, identical figures have different binary representations on different machines. This depends on the physical devices and the drivers that are used.

For the figures using SAS GTL, the smartest solution catches the input dataset and the source code for PROC SGRENDER. We store both as reference data and compare the template and the source to the input data. With these three artefacts it is possible to verify if two outputs are identical, using the methods for text files and datasets described above. Moreover, with a short code snippet the SGRENDER source code can be called to produce the output graph on the fly, for instance, to review the content of the figure.

MODULE TESTS

For module tests, the test objective must have the ability to preserve all or at least the main important internal temporary datasets. From the perspective of the test framework, the same routine as for datasets can be used, but you have to include a switch in the test objectives to prevent temporary dataset from deletion to have this capability.

RESULT GENERATION

The programmers only accept the test framework if it supports the developers in generating and storing the references. The quality of the references is the kingpin of the quality of the software. It must be easy to store the references according to the naming conventions at the defined place and to control the contents of the reference. Otherwise, the quality assurance of the references will take too much time and effort, and it will break the advantages of the agility. Therefore, we implement a special mode to save the references of the results with the test framework.

CONCLUSION

The time invested in creating and maintaining the automated test cases will have a positive return if the developed programs exist for more than one version. Fixing a certain bug is usually easy when you know in which situations it is visible.

With every developing round of the test objective, the test cases cover more and more functionality. That leads to the fact: quality grows with the features. This makes automated tests optimal for agile software development with a high deployment rate.

Once the test are available, programmers can use them for regression analysis. By programing a patch for the software and running the test cases give the developer a quick overview of which consequences and side effects this change for the software has.

My experience with agile software deployment in combination with automated test cases shows that after a short period of investing time for the test case implementation you get this investment back in form of higher quality and getting a quicker output in each agile software development cycle (called sprint)

CONTACT INFORMATION

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

Sven Prasse
Entimo AG
Stralauer Platz 33-34
10243 Berlin
Germany

Work Phone: +49 30 52 00 24 100
Fax: +49 30 52 00 24 101
Email: pr@entimo.de
Web: www.entimo.de