PhUSE 2013

Paper IS02

Useful Principles and Practices for Professional SAS Programmers

Patrick René Warnat, HMS Analytical Software GmbH, Heidelberg, Germany

ABSTRACT

In professional software development, certain general principles and practices exist that are an important complement to pure technical programming skills.
Recently, the gathering of such recommendations for SAS programmers from life science companies got more attention especially by the initiative of the PhUSE community on good programming practices, resulting in a promising start of knowledge exchange.
This paper contributes to this initiative by providing a concise overview on basic as well as advanced best practices, including concepts not yet in widespread use among SAS programmers but also applicable to SAS programming. The topics covered by this paper include but are not limited to: structuring of code, documentation, revision control, unit testing and refactoring. For each concept, an explanation, the benefits of its use, the specific application in SAS programming, as well as references to sources with further details are given.

INTRODUCTION

Did you ever ask yourself: What makes a good SAS© programmer? As a junior programmer, you might ask: What do I need to learn to do a good job? Or as a senior developer, you might ask: What do I want the junior programmers in my team to know?
I did and still do ask myself such questions as part of my daily work, and in this paper I would like to give an overview on what I found to be useful principles and practices for SAS programmers.

First, I would like to state that the selection of presented items represent my personal view on a selection of seven important practices. Thus, this list of useful principles is not complete, but instead is restricted to an amount of essential items I found to be appropriate for this paper format.
In addition, a few words on the scope of this paper:
It should be clear that a professional programmer needs profound technical programming skills (understanding of how the SAS system works, knowledge of syntax of the SAS language, and so on). This can be learned by books, courses, etc., and is assessed by SAS programming exams. This is seen as a fundamental skill base and is not the topic of this paper.
Of course, beside this foundation of profound technical programming skills, general skills are needed as a professional, like communication skills or self-organization, which are also not topic of this paper.
The focus of this paper is on practices that are an important complement to pure technical programming skills, but more specific for the work of a programmer as soft skills or other general skills (as e.g. soft skills are important for nearly all professions).

From my point of view, the discussion of such practices is of special importance for SAS programmers, because in my experience SAS Programmers (especially in the life science industry) are often recruited from career changers who did not go through a formal general training in computer science/software engineering.
The current discussion of the PhUSE community on good programming practices [1] covers some of the topics of this paper, but not all, and many current contributions focus on programming details like “why to prefer WHERE instead of IF statement for sub-setting data”. These details are important, but this paper would like to provide an overview on practices at a bit higher level of abstraction.

The principles named here were selected based on my own training and experience. Each of it is not new and you will find all of them named in different literature sources on programming (for a start, look at section “recommended reading” at the end of this paper), but I did not yet see them written down together in the context of SAS programming.
Thus, the following paragraphs will name my selection of important principles and practices for SAS programmers followed by a conclusion.

USEFUL PRINCIPLES AND PRACTICES – AN OVERVIEW

The following selection of topics is a personal selection of useful principles and practices for SAS programmers besides pure technical SAS programming skills.

KNOW THE APPLICATION DOMAIN OF YOUR PROGRAMS

One should not simply implement some piece of source code, without knowing the context in which the software will be used. Always keep in mind the code is written as part of a solution to a real-world problem.
Often uncared for, but very important, is to have knowledge about the application domain of the software you are creating, because one can find better solutions if one knows the context the software is used in. This can help a lot to find better implementation solutions and to prevent errors.
It includes the need to have an understanding of the methods that are used in a certain context, like statistical methods in medical analyses.
This principle is especially important in SAS programming as SAS programs are often used in very specialized areas (e. g. pharmaceutical companies, finance industry), each having its own set of domain knowledge and specialist terms.

CRITICALLY ACCESS REQUIREMENTS

Clearly, a profound knowledge of requirements engineering ("formulating, documenting and maintaining software requirements" [2]) is essential for many positions in software development.
But even if you are not in charge to formulate requirements, it is important for a programmer to critically assess given requirements and to be able to check for completeness and consistency. If a programmer takes some time to check and critically access requirements before she or he starts coding, it will lead to more efficient programming as issues with the requirements are detected early. A special aspect of requirements assessment from the programmer's point of view is to check for every requirement what the criteria are that have to be met to be able to say the requirement is met. This automatically leads to more focused programming and is a base for efficient testing of the software.

WATCH YOUR ATTITUDE AND APPROACH

As a professional programmer it is important to inspect one's attitude towards programming. Certainly, a person's attitude towards doing programming work comprises many aspects, but the following three items provide a good basic set of principles to start with:

"Boyscout rule" [3]:
Leave a place in a better condition than you found it. Applied to programming: If you are to change a piece of software, do not only change the parts absolutely necessary but also improve exiting running code (applying naming conventions, refactoring , …) if you see something worthy of improvement. Thus, the code gets better and better every time you work on it.

Defensive programming:
During programming, try to anticipate what could go wrong during program execution and implement checks and error handling. For SAS programming, this could mean to implement checks of the input parameter values of a macro, creating and using project specific error handling macros, etc. (see [4] for details how to apply defensive programming to SAS macros).
From time to time, it is very helpful to try to think more as a tester than a programmer. A programmer often puts the focus on what a program must do to succeed and fulfill its requirements. A Tester often puts the focus on what the flaws of a program are, what input values (macro parameter values) might cause a problem, and this mindset is helpful to create more robust software.

Code reviews [5]:
Be open-minded to code reviews with colleagues in order to learn from others or give others a better chance to learn from you. Avoid strong code ownership in order to be more flexible, and get better code quality. Strong code ownership would mean that a certain software unit is only changed by the person who originally programmed it.

There exist many recommendations regarding the approach to software development. Every programmer should know two very basic practices and should take heed of them as often as possible:

Iterative development [6]:
Don't try to make too much in one step, try to work in small steps, e.g. first do not write macro code (only PROC and DATA steps), then introduce macro variables, then macro code. This makes it easier to handle complex tasks and enables one to perform test runs as early as possible.

Refactoring (changing the internal structure of code without changing the external behavior [7]):
Take your time for refactoring to improve code structure, readability and maintainability.

If possible, don't stop programming as soon as your code runs, but take some time to go through it again and check whether you could improve the internal structure of the code.

HANDLE THE LIFECYCLE OF YOUR SOFTWARE

Especially the following two practices are a great help in a programmer's daily work.

Revision control:
First and most important, a version (revision) control system [8] (e.g. Subversion [9]) should be used in order to track and control changes to source code (and other) files of a software project. Most importantly (among other good reasons) because it takes away the fear of changing a running program and messing up your source code irreversibly.

Build automation:
Additionally, in large projects where you have to deal with many files, you might consider to use a build tool (e.g. Apache Ant [10]) to automate routine tasks (e.g. copy actual version of your source code files and archive them into a zip-file for delivery of the code to a customer).

STRUCTURE YOUR CODE

Structuring of source code is a very broad topic and one aspect of software architecture. Again, I only want to name a few important basic principles of what every SAS programmer should know and apply from my point of view:

Separation of concerns [11]:
Do not implement software that tries to fulfill many requirements in one unit of source code. Separate parts of the code dealing with different tasks. For tasks that are comprised of different steps, separate the different steps at least by proper source code formatting and helpful comments.
This leads to software that is better testable and will give you parts that are more easily reusable. For SAS programming, this means to divide your code in manageable sized parts and to put each part in a separate SAS macro.

Don't repeat yourself [12]:
Simply don't implement the same functionality twice at different parts of your software, but create a reusable unit that contains the functionality used at several places in your software.
Otherwise errors caused by inconsistencies become possible. Clearly, this is one of the most important reasons why to use SAS macro techniques.

Single Level of Abstraction (in one unit stay on one level of abstraction):
In a software unit only one level of abstraction should be used, otherwise someone reading the code will have difficulties to distinguish essential steps and details.
Detail operations like access to data set attributes in "data null steps" should not be implemented right beside calls to macros implementing a high level abstraction like a self-defined macro "generateOutputReport()". Instead, detail operations should be encapsulated, too. Thus, the structure of the processes implemented by a source code will get clearer. You will get better readability and maintainability if you stick to one layer of abstraction per macro level.
See the book "Clean Code" from R. C. Martin for more details on this principle.

YAGNI ("You aren't gonna need it" [13]):
"YAGNI" means to focus on the code needed to fulfill the requirements of the software users. No code should be implemented only because the programmer has the idea that it could be useful sometime in the future. This leads to lean code that is not overblown with generic features which are not needed in the current task.
This is especially important while implementing SAS macros that are used in more than one application.

MASTER UNIT TESTING

In order to assess the correctness of source code, testing is indispensable. The earlier testing starts, the better. Thus, for programmers, Unit Testing matters most. The concept of Unit Testing implies that tests themselves should be implemented as executable pieces of source code. As a result, tests can be repeated anytime, and negative side effects of software changes can be spotted easily. Programmers should implement executable tests for every unit of source code they create. This can be a great help during iterative development of a SAS macro and even of more help during maintenance work happening later. Unit Testing can be implemented with the SASUnit [14] macro framework.
Of course, other types of testing are also very important (e. g. integration testing), but unit testing is the most important testing approach in a programmer's daily work.

APPRECIATE SOURCE CODE DOCUMENTATION

Software documentation is important for users and maintainers of software. As a professional programmer one should especially pay attention to source code documentation. This includes syntax writing conventions, readable and meaningful names for macro variables and macros and comment headers in source code files. Everybody working with the code will benefit from it, enabling a quicker grasp of the structure and inner workings of the code. Headers of source code files could be generated in a tool compatible syntax, enabling automatic generation of external representation of the documentation. For application with SAS programming a tool named Doxygen [15] could be used.

CONCLUSION

The listed principles and practices represent a subjective selection with no claim for completeness. They are to be taken as a starting point for thinking and discussing what could be helpful for professional programmers besides pure programming skills and general soft skills.

Of course, it is not easy to introduce new practices in daily work as there nearly always is a pressing deadline. Try to take little steps and not too many at once and you will gain small but constant improvements in your results.

If you are a senior professional already aware of and mastering all practices listed in this paper, go out and advocate and discuss these and other principles which support professional SAS programmers in creating even better software.

REFERENCES

web links as accessed on 2nd October 2013:

[1] http://www.phusewiki.org/wiki/index.php?title=Good_Programming_Practice
[2] http://en.wikipedia.org/wiki/Requirements_Engineering
[3] http://programmer.97things.oreilly.com/wiki/index.php/The_Boy_Scout_Rule
[4] http://www.phusewiki.org/docs/2009%20PAPERS/AD01.pdf
[5] http://en.wikipedia.org/wiki/Code_review
[6] http://en.wikipedia.org/wiki/Iterative_and_incremental_development
[7] http://en.wikipedia.org/wiki/Refactoring
[8] http://en.wikipedia.org/wiki/Revision_control
[9] http://subversion.tigris.org
[10] http://ant.apache.org/
[11] http://en.wikipedia.org/wiki/Separation_of_concerns
[12] http://en.wikipedia.org/wiki/Don%27t_repeat_yourself
[13] http://en.wikipedia.org/wiki/You_aren%27t_gonna_need_it
[14] http://sourceforge.net/projects/sasunit/
[15] http://www.stack.nl/~dimitri/doxygen/

RECOMMENDED READING

- "97 Things Every Programmer Should Know"; K. Henney et al.;O'Reilly Media; 2010.
  http://programmer.97things.oreilly.com/wiki/index.php/97_Things_Every_Programmer_Should_Know
- "Clean Code"; R. C. Martin; Prentice Hall International; 2008.
- "The Productive Programmer"; N. Ford; O'Reilly Media; 2008.

CONTACT INFORMATION

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

Dr. Patrick R. Warnat
HMS Analytical Software GmbH
Rohrbacher Str. 26
69115 Heidelberg
Germany
http://www.analytical-software.de/en/start/