KISSing in the workplace

Eduan Cronjé
Statistical Programmer

%let Lazy = Efficient;

%let Lazy = Efficient; (continued)

Are programmers naturally lazy?

Are lazy people drawn to programming or is this a social effect?

Does laziness stem from the best programming practices?

Do the best programming practices stem from laziness?

Keep It Simple, Stupid

After attending a training session, I asked a friend what he had learned, and he simply answered: ”Keep It Short and Simple”. In my mind this is the most important contributing factor to greatness and the successful completion of tasks in a timely manner. Just keep the KISS concept in mind and adhere to it, whether you are drafting an email or a document, conducting a meeting, designing or coding.

I later found that KISS is an abbreviation for “Keep it Simple and Stupid” which is a design principle articulated by Kelly Johnson. In addition, there are variations to this principle and one of them is “Keep It Short and Simple”.

Efficient date conversion

- Something as simple as a date conversion serves as a perfect example for explaining the KISS principle:
  - Handling a date as a whole may prove troublesome and prone to error due to the fact that we do not live in a perfect world and some individual parts may be missing or contain unexpected values.

- An ISO8601 date value in one of its simplest forms (YYYY-MM-DDThh:mm:ss) may be broken down into 6 parts
  - Year part(YYYY)
  - Month part(MM)
  - Day part (DD)
  - Hours part(hh)
  - Minutes part(mm)
  - Seconds part(ss)

- We as programmers do not have complete control in how a date is captured. Hence, we know that:
  - The format may vary, it may be incomplete, it may be partial, it may be captured in separate parts etc.
  - These are just a few examples which we should be expecting in a real world scenario and thus we need to make provision and possibly flag/report these types of issues where applicable.

Efficient date conversion(continued)

- Good Programming Practices(GPP) encourages us to create our own flags and internal efficiencies in order to allow us to check our own work. I have written a simple macro that produces a utility dataset like this (note that the actual output has already been handled and is written out in a separate dataset):

	Character date(ISO8601)	Numeric date	Possible Issue	Issue #
1	2018-11-12T03:18:16	1857611896.4	N/A	.
2	2018-02-T14:02:32	.	Missing day, Hours present	3
3	2018-UK-12	.	Missing Month, Day present	2
4	09APR	.	Missing year, Month present	1

Testing

One of the most important aspects of automation is to make sure that your automation/efficiency actually works.

Functional tests include:

- Unit testing
- Integration testing
- System testing
- Sanity testing
- Smoke testing
- Regression testing
- Beta/Acceptance testing

Unit Testing

The most common Functional test we use on a daily basis is Unit testing:

- It is typically done by the programmer and not by testers, as it requires a detailed knowledge of the internal program design and code. It may also require the creation of dummy test data.
- Dummy test datasets should be updated as new issues are discovered. As new issues are discovered, our program needs to be updated and the unit test needs to be rerun.

```
data test_data;
  length indate $200.;
  indate = "15JUN2018";
  output;
  indate = "15JAN";
  output;
  indate = "2005";
  output;
  indate = "14-05-2012";
  output;
  indate = "23-UK-2018";
  output;
  indate = "UK-01-2011";
  output;
  indate = "UK-01-2017T12:45";
  output;
run;
```

The Plate Spinner

- Automation gives us a centralised control panel to manage multiple activities. Automation is like spinning plates on sticks. Fundamentally, someone has to make sure that they keep spinning without colliding with each other.

- In our industry, the plate spinners are usually the authors of efficiencies and automation programs. This is usually a more senior programmer with a lot of experience and a great amount of technical skill. The work this “Plate Spinner” has put in, may be overwhelming and very hard to comprehend for a new team member(or anyone who is not familiar with his/her coding style).

Taking over code

Taking over code or assuming the role of the Plate spinner is very similar to a TAKEOVER in business:

- In Business, the takeover of a company is defined as:

“Purchasing a majority stake in the target firm. If the takeover goes through, the acquiring company becomes **responsible** for all of the target company’s **operations**, **holdings**, and **debt**.” - Investopedia

The Plate Spinner (continued)

- When we take over code, we accept responsibility for the logic (**operations**), generated outputs (**holdings**) and **debt** (GPP not followed, flawed logic, non-robust code etc.).

- The **debt** in this case may be overwhelming for many people(myself included).
  - Robert Kiyosaki mentions in his book: ”Rich Dad, poor Dad” that we can have either good debt or bad debt. Other People’s Money (OPM) is a fundamental concept of Rich Dad and a sign of high financial intelligence. By using both good debt and OPM, you can dramatically increase your Return on Investment (ROI)—and you can even achieve **infinite returns**.
  - OPM in this case is the **hard work and dedication** the author has put into the efficiency/code.

"If I have seen further it is by standing on the **shoulders of Giants**.“ – Sir Isaac Newton (1675)

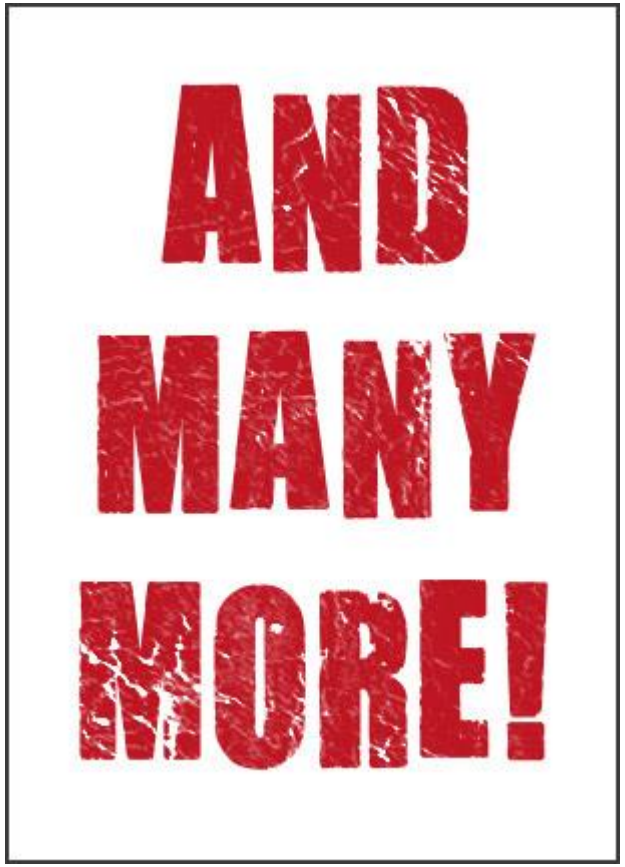
Debt/Bad coding example

```
data To_SAS_DataSet;
   set From_SAS_DataSet;

   if VarA = "Yes"            then %doSomething();
   if VarA = "      Yes"      then %doSomething();
   if VarA = "Yes           " then %doSomething();
   if VarA = "      Yes"      then %doSomething();
   if VarA = "YES"            then %doSomething();
   if VarA = "Y"              then %doSomething();
   if VarA = "yes"            then %doSomething();

/*VS*/

   if index(upcase(compress(VarA)), "Y") gt 0 then %doSomething();
run;
```



SDTM and ADAM shells

When we create ADAM or SDTM outputs, we normally have a specification document which already has all variables with the expected lengths, variable type, variable label etc. included within the document. We can use the information captured in the document instead of assigning it manually.

- I personally see the set statement in SAS as a form of forcing inheritance – OOP concept.
- When the variable names are the same, attributes are “inherited” from the first dataset in the set statement. Thus, we can have an empty dataset with the correct structure which is dynamically generated, based on the specification document and simply utilize that. The program will work based on the assumption that the spec is correct and will incorporate any spec changes every time it is re-run.
- A shell dataset will be created, ready to be utilized:

```
%Create_Shell (spec_path=C://Shell_loc/shell.xlsx, domain=AE);
```

SDTM and ADAM shells (continued)

A few Advantages and Assumptions of this approach:

Advantages:

- Responsive – Adaptable to spec changes.
- Repeatable – This can be utilized on any study if the spec is in the **expected format.**
- Reliable – This program handles incorrect parameter entries and produces **custom warnings, errors and notes** where applicable.
- Fewer lines of code.
- Time saving.

Assumptions:

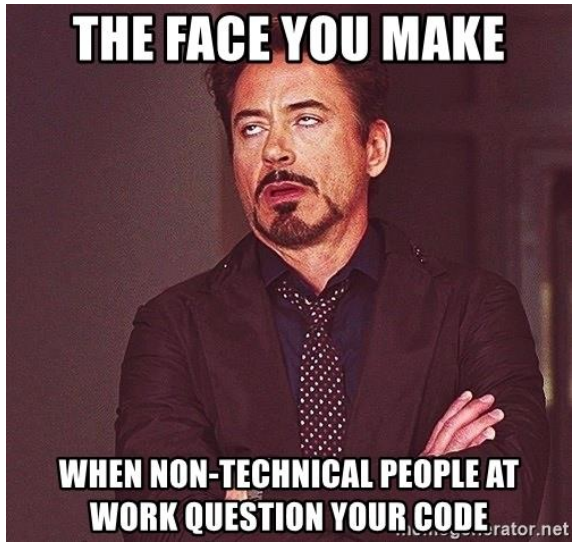
- The specification document has to be validated as per CDISC standards in advance.
- The specification document is in a standard, expected format.
- The specification document has to be validated by an automated spec validation macro.

Why automate? - 3 R's

- **Responsiveness**
  - In a manual process, any communication you get must be reacted to. Instead of being prepared and moving, you're behind and rushing. The ability to remediate an issue that you're alerted to automatically means you've already saved time on your response. Now instead of manually remediating that issue, we can apply a task to the issue that will take care of the remediation for me.
- **Repeatability**
  - Sure, doing something once or twice would be okay; but if you have to repeat a task several times, then you're doing nothing but duplicating your efforts. So we have to be able to create an automated process that does certain tasks for us.
- **Reliability**
  - Once the tasks are written and the templates are built, we have a process. Instead of applying after the fact, no matter how easy that may be, we can automate the deployment process altogether. We need to keep in mind that we should always apply a set of checks that we want to ensure are used.

Commandments of egoless programming

Commandments of egoless programming (continued)



Commandments of egoless programming (continued)

Conclusion

Keep it Simple Stupid

- Automation is good, but easy to use automation is better
- Smaller pieces are easily understood
- Spaghetti code will let you choke
- Egos can get in the way of life and code

Code simple, write beautiful automation and let the light shine over your fellow lazymen.

Conclusion (continued)

```
"""The Zen of Python, by Tim Peters."""

Beautiful is better than ugly.
Explicit is better than impl..
Simple is better than complex.
Complex is better than c0mp1|c@ted.
Flat is better than nested.
S p a r s e  is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
raise PythonicError("Errors should never pass silently.")
# Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
```

Thank you!

Sources

- Present and former Co-workers

- Rich dad, Poor dad – Robert Kiyosaki
- The Psychology of Computer Programming - Jerry Weinberg
- Simplicity, Inference and Modelling: Keeping It Sophisticatedly Simple - Edited by Arnold Zellner, Hugo A. Keuzenkamp and Michael McAleer
- From Occam's Razor to the Roots of Consciousness - Gerhard D. Wassermann
- The KISS principle – Karen Booth

- https://www.solarwindsmsp.com/ - 3 R’s of Automation
- Google scholar
- Refseek