

ICON PLC

Menaga G
Ponnupandy

plagiarism

SSO1

Fishing for Code Plagiarism in Clinical
SAS Programming



Agenda

01

Quality Control in CRO

02

Challenges in Double
Programming

03

How is Code Managed?

04

Potential Solutions

05

Utility 1 – 3

06

Questions?



Problem Statement

Quality Control

Double Programming is used in Clinical SAS Programming



1

Same Specifications

Code written from scratch for the same specs and outputs compared

2

Output Validation

Reduced errors and increased accuracy of the data

3

Quality Assurance

Smallest of errors are caught and rectified, ensuring quality of data

4

But what's wrong?

Let's take a look



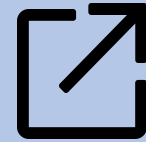
Challenges in Double Programming



Copying other
programmer's
code



When pressed for
deadlines a quick
way to get to
results.



Code plagiarism,
missing potential
bugs



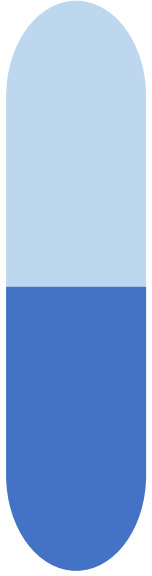
Undetected till
late and leads to
defective code



No access rules on
.sas files lead to
quality issues



Code Management



Source Control

Tracks changes over time; who changed what lines of code when, and why



Version Control

Retain original version and any subsequent revisions that contain modifications



Track Changes

History of why or who broke something; understand code evolution over a period

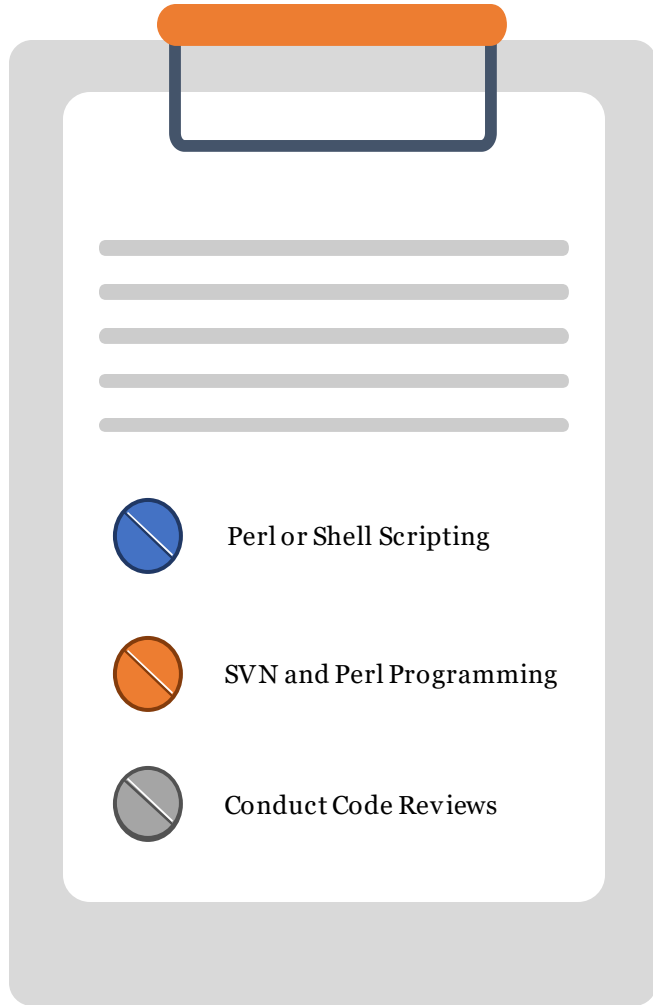


Know Your SCM

Code auditors to have a basic understanding of how SCM systems work



Potential Solutions



How to overcome the challenges?

01

Perl or Shell Scripting

Versatile and can be used for automating tasks, writing small utilities, or prototyping new ideas.

02

SVN and Perl

Powerful tool for code review can be created by programmers using SVN and Perl.

03

Code Review

Using these tools reviewers can conduct code audits and identify potentially plagiarized code.



How to Conduct Code Audit?

Grep/Awk

Basic grep or awk command to an advanced script specifically designed to capture problematic code blocks of one program that closely resembles with another

Scripts

Three scripts that can help the end-user perform a code audit.

SIMILARITIES.PL
COMPARE-USER.PL
COMMIT-HISTORY.SH

Log -Diff

Deep dive into the use svn log -diff command to dig into the details of code of special interest

Code Review

Proper guidelines and introducing a comprehensive code review process where the code written by the two programmers are cross-checked



Code Audit Scripts

COMMIT-HISTORY.SH

Output details of SVN
Commits of each programs

3

SVN LOG --DIFF COMMAND

Provides information about the
history of a file of special interest

2

COMPARE-USER.PL

Compare user list between
development and validation
programs

SIMILARITIES.PL

Identify code similarities
between development
and validation programs

1



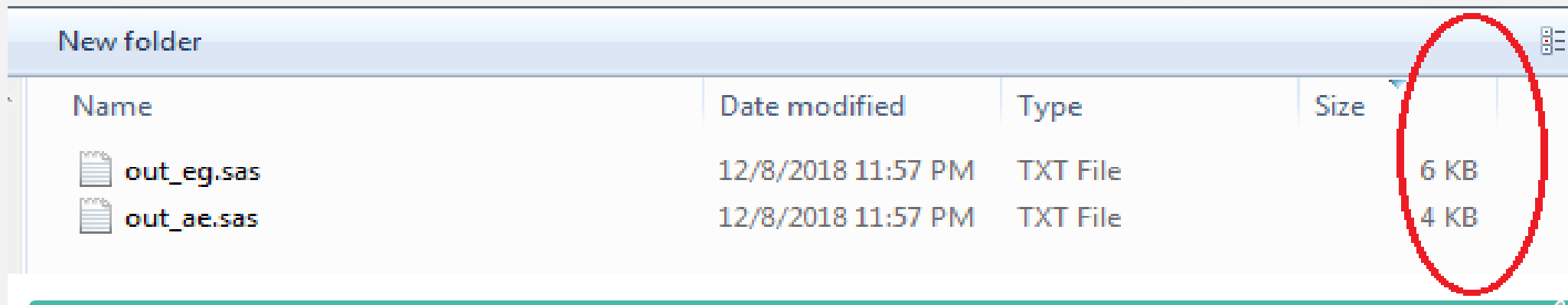
SIMILARITIES.PL

- It is used to compare the development & validation codes and print the common lines between the two files to an output file.
- No output files to be generated
 - a) If there is no similarity between the two files or
 - b) If the relevant file for comparison is not found in any one of the directory
- The common lines like “end; quit; run; blank lines” are eliminated from the output file for better visualization of results.
- This comparison can be recursively performed for all files within a directory.
- The reviewer to pass on 3 arguments to run this script:
“perl similarities.pl -main /projects/prog -qc /projects/qcprog
-output /projects/qcprog/matches”



Execute Script...

- The execution of this script generates an output file named:
“out_programname.sas.txt”
- The size of the output file can be used to judge the extent of similarities between two files.
- The reviewer can sort the files based on its size to see the files with maximum similarities.



Name	Date modified	Type	Size
 out_eg.sas	12/8/2018 11:57 PM	TXT File	6 KB
 out_ae.sas	12/8/2018 11:57 PM	TXT File	4 KB



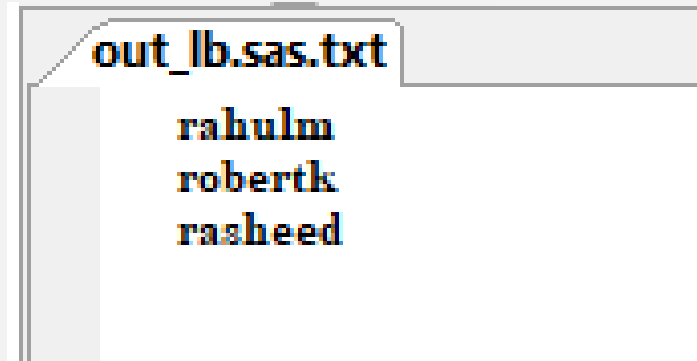
COMPARE-USER.PL

- The compare-user.pl is similar to the script 'similarities.pl'.
- It compares the user list between each of the development and validation programs to output programmer names that are common between both.
- Backstage the SVN log of development and validation program are pulled to produces the common user list.
- In an ideal case, programmer is expected to work on or check in codes only on one side and are not supposed to work or check in on the other side.
- This utility ensures there is no cross-contamination between development and validation programs
- The reviewer to pass 3 arguments to run this script:
“perl compare-user.pl -main /projects/prog -qc /projects/qcprog
-output /projects/qcprog/common_user”



Execute Script...

- This script execution generates an output file named:
“out_programname.sas.txt”
- In the example below, the programmer names who had checked in or worked on both the development and validation programs of lb.sas are printed.



```
out_lb.sas.txt
rahulm
robertk
rasheed
```



COMMIT-HISTORY.SH

- It is a simple shell script.
- It is used to output the details of SVN commits of all the programs under the development or validation folder.
- The option `--quiet` is used to print only the essential information.
- `\n` is used to print in newline for each file that is processed in the execution of this script.
- These commit history details are presented for all the codes located under development directory `‘/transfer/prog’`.
- The reviewer to execute the script and direct the results to an output file:
“commit-history.sh > `commhist_sdtmmn.txt`”

```
#!/bin/sh
for f in *.sas
do
    echo "processing $f ... \n"
    svn log $f --quiet | grep "^r"
done
```

For new line

For readability



Execute Script...

- The execution of this script generates one output file named:
“commhist_sdtmmn.txt”
- The output file contains the revision number, user id and the date the update has been made.

commhist_sdtmmn.txt

```
processing ae.sas ...
```

r3471	rahulm	2018-12-10	16:23:30	-0500	(Mon, 10 Dec 2018)
r3195	robertk	2018-11-26	14:12:27	-0500	(Mon, 26 Nov 2018)
r2457	rasheed	2018-08-31	03:45:50	-0400	(Fri, 31 Aug 2018)
r2201	rasheed	2018-07-30	14:54:41	-0400	(Mon, 30 Jul 2018)



Code of Special Interest

- In the 'similarities.pl' run, for instance, if one file is noticeably larger than the other, there are probably more similarities between them.
- This information along with the information generated from the two other scripts can be used to prioritize review efforts and ensure that the most important files are reviewed first.
- If the code reviewer has itemized the most important file or the file of special interest, then they can make use of the below commands:
 - a. `svn log --diff vs.sas > vs_chghist.txt`
 - b. `svn log --diff qc_vs.sas > qc_vs_chghist.txt`
- `--diff`: Display all the differences or the updates made i.e., from initial version to the current version.



Final Report

- These .txt files generated from the above commands can provide a complete history of a file of special interest.
- By looking at the history of a particular development and validation file, it's possible to see exactly who changed what, and when.
- This can be very helpful in tracking down who introduced a particular bug, or in understanding why a particular piece of code was added.
- Additionally, auditors can use this information to identify potentially plagiarized code, or code that has been copied from development to validation or vice versa over a period.
- This process is extremely important in auditing code for errors or security issues.



Avoid Code Plagiarism



Quality Assurance

Code audit is essential part of quality assurance for clinical programming projects that face quality issues



Code of Special Interest

A reviewer can identify potential problems by careful examination



Automation

Manually reviewing a large codebase can be a daunting task. Automate review process using tools.



Use Source Control

Use SCM to see who, why and what changes has been made to a particular file. **Avoid Code Plagiarism**



Talk to me to know more...



menaga.guruswamyonnupandy@iconplc.com

