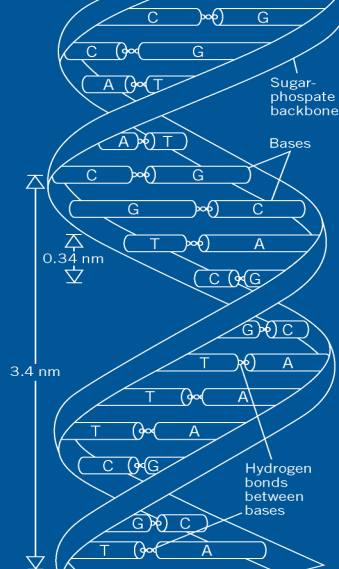




Modular Programming

Some Lessons Learned and Benefits Gained

Ross Farrugia, Roche

Contents

- Introduction

- Planning

- A real-life example

- Implementation

- Influence on validation

- Benefits/Lessons learned

- Conclusions

Introduction

/ What is Modular Programming? */*

%macro Wikipedia;

“**Modular programming** is a software design technique that increases the extent to which software is composed of **separate, interchangeable components**, called **modules**”

%mend Wikipedia;

Introduction

/ What does that mean to us? */*

Splitting our programs up where possible into clearly separated modules of code which can be **easily maintained and re-used** across different reporting events

The easiest way of doing this being the use of **macro modules** – depending on your requirements you may choose different granularity of code/modules

Note: even if macros are not used then at least structuring your program according to these ideals will make for greater maintainability, re-usability, understandability, and more.

Introduction

/ Where could this be used? */*

In any program! Be it for mapping code, an analysis dataset, an output, ...etc

If we are performing any action that will need to be repeated again in future, then why not take this out into a separate module that can be referenced by any other program

The module of code could perform a number of data steps or even just one function

Examples

/ An AE analysis dataset program using modular programming */*

```

** Highest CTC 345 day of/after or during inf AE in Cycle **;

%g_minmax(dsin      = ae_ctc5,
          invalobs   = ,
          incmiss    = Y,
          inwhere    = %str(upcase(aecpe) like ('CYCLE__%') and (aeinfu eq 'YES' or aeinfng eq 'Y
          dsinsub    = aecpe aept,
          dsinseq    = aeintc aebedgt aept,
          dsinval    = aeintc,
          style      = MAX MIN MIN,
          outfg      = aeina3fg,
          dsout      = ae_ctc6,
          debug      = &debug);

*****
** Add details of concomitant treatments **;
*****

%ae_conmd(dsin      = ae_ctc6,
          dsin_medo  = &medo,
          dsout      = ae_conm1,
          debug      = &debug);

*****
** Add details of last study treatment before AE **;
** (Treatment name, start date, study day) **;
*****

%ae_lstmd(dsin      = ae_conm1,
          dsin_medt  = &medt,
          mttxt      = ██████████ PLACEBO ██████████,
          dsout      = ae_lstm1,
          debug      = &debug);

```

Examples

/ An efficacy analysis dataset */*

/--- call EVBOR macro which derive BOR, CBR, date of first response, patient is a responder,
 /*--- Objective response , cycle at which best response occurred;*

```
%ev_bor    (      dsin    =__event1,
                dsinresp  =tmresp,
                dsincpe    =tmcpe_o,
                dsindt     = tmdt,
                dsintrt1dt  =trt1dt,
                style      =UA,
                dsincnfm    =tmconfm,
                varbor      =evbor,
                varcybor    =evborc,
                varfrdt     =evfrdt,
                varresp     =evresc,
                varcbr      =evcbr,
                varor       =evor,
                varocbr     =evocbr,
                dsout      =__event6,
                debug      = N
            ) ;
```

/-----Patient with PD identified only using SYMPT.DETERIORATION;;*

```
proc sort nodupkey  data =__event1 out =__event9 (keep=  &_uproto &_ucrtn &_upt tmsympt);
  by &_uproto &_ucrtn &_upt tmsympt;
  where tmresp="PD" and tmsympt ne .;  ;
run;
```

```
data __event10 (keep = &_uproto &_ucrtn &_upt ) ;
  set __event9;
  by &_uproto &_ucrtn &_upt tmsympt;
  if first.&_upt + last.&_upt =2 and tmsympt=1 then output;
run;
```

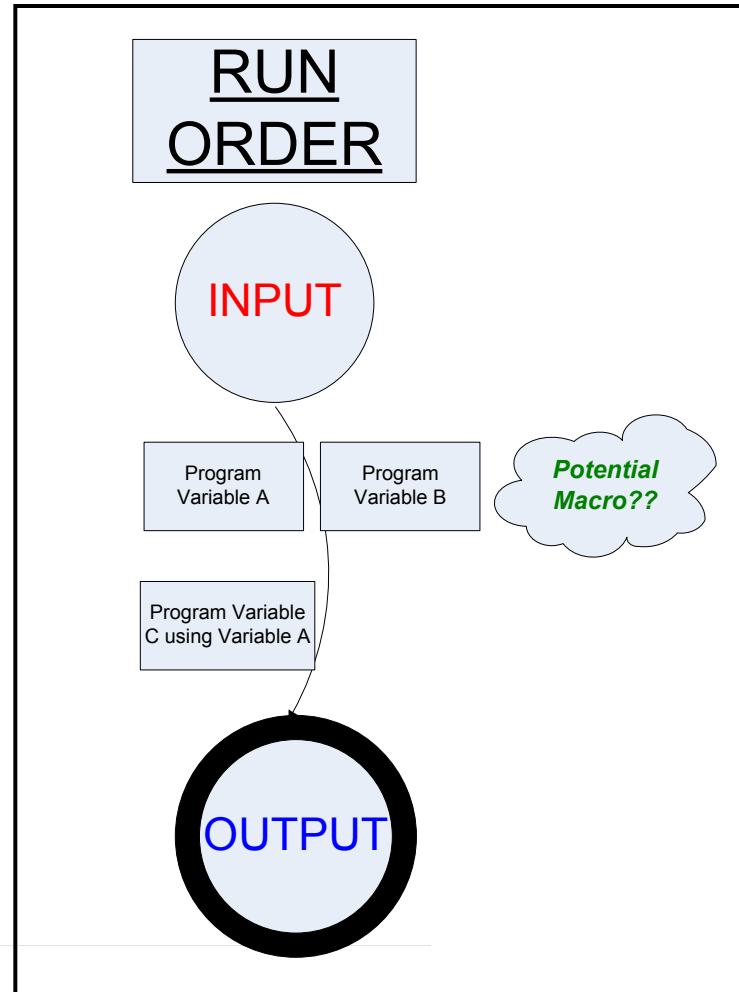
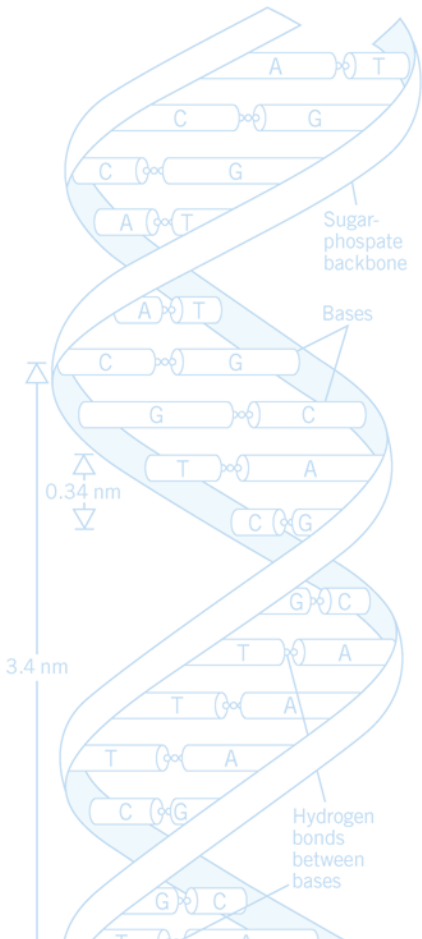
Planning

/ Requirement Analysis*/*

When you are asked to create a program, start by **breaking it up** into set derivations (or coding tasks) required

These will form your individual modules of code

Consider **run order** of these modules, any **dependencies**, **complexity**, and the overall **efficiency** of the program - these factors will impact your modular design



Planning

/ Separate macro VS module of code within your program */*

Is this piece of code likely to need to be **repeated** in this program or in another program on this study?

Could this be **re-used** on future studies?

Is this code **complex** enough to justify the extra overhead of creating a separate macro?

Planning

/ Considerations if you decide on a separate macro */*

Is the requirement applicable at the **project level** or just specific to a study?

Does the derivation rule apply only to a **single domain** of data or multiple?

/ Considerations for project/across-project level */*

What rules have you seen here in **previous study** requirements?

What are you aware is requested or could possibly be for **future studies**?

A real-life example

/ Requirement */*

AE analysis dataset requiring flag variables for the following 3 cases:

“Most severe adverse event by treatment cycle”

“Most severe *treatment-related* adverse event by treatment cycle”

“Most severe *during infusion* adverse event by *visit*”

A real-life example

/ Planning */*

The programmer identified that the fundamental code would be the same across the 3 derivations.

They then looked back to a **past requirement** on an old study where this was also needed for a different domain, cardiac symptoms. So it was decided the macro could be designed to use across domains.

By **consulting the team** it was decided that this could be a **future requirement** on any subset of adverse events, not just *'treatment-related'* or *'during infusion'*, so this would also need to be made flexible.

A real-life example

/ Result */*

`%g_cycfg` - a macro to flag events according to some user-defined “worst” criteria by timepoint

The flexibilities to allow this are shown below:

OPTION NAME	MANDATORY/ OPTIONAL	DEFAULT	OPTION DESCRIPTION
<u>dsin</u>	Mandatory	<none>	Input dataset
<u>worstvar</u>	Mandatory	<none>	Variable containing “worst” values
<u>hilo</u>	Mandatory	<none>	‘H’ or ‘L’; Is high value or low value worst?
<u>where</u>	Optional	<none>	Where clause for input data
<u>byvar</u>	Mandatory	<none>	By cycle/timepoint variable
<u>outvar</u>	Mandatory	<none>	Output flag variable name
<u>dsout</u>	Mandatory	<none>	Output dataset

A real-life example

/ Result */*

Now if we take the “Most severe *treatment-related* adverse event by treatment cycle” variable, this would have been programmed just as:

```
%g_cycfg (    dsin          = ae1,
               worstvar      = aesever,
               hilo          = H,
               where         = %str(aerel='YES'),
               byvar         = aecycle,
               outvar        = relsevcy,
               dsout         = ae2
               )
```

Implementation

/ Project standards */*

Naming conventions:

- Generic macros - '%g_xxx'
- Domain specific macros we use the first 2 letters to indicate the domain, e.g. %ae_xxx, %dm_xxx, %lb_xxx
- Utility macros - '%u_xxx'

Generic specifications:

1. Scope and Purpose of macro.....
2. Outputs produced by macro
3. Macro Assumptions
4. Description of Derivations.....
5. Macro Flexibility
6. Validation Checks
7. Macro Parameters.....
8. Example

Implementation

Generic program template

- May be useful to set up a generic template for programmers to start with
- Agreed programming conventions applied

Keep a Macro Index file

- Helpful for new starters on your project

Name	Description / Notes	Contact for advice	Validation Method
g_centfile	Contains variable/dataset attributes at project level	Littish / John	Code review
g_attrb	Add final dataset attributes according to project standard	Littish	Code review
g_baselinech	Calculate baseline change and change from baseline change	Littish	Code review
u_abort	Used everywhere - aborts processing, used as part of check code.	Littish	Code review
u_chkattrchg	Check attributes against reference library. Useful in particular for GDMs	John	Code review
dm_popn	Merge population data from EXCL onto DEMO	Waseem	Code review
ev_bor	Derive EVENT VAD variables such as best overall response, Clinical Benefit (CBR)	Littish	Unit Testing
tm_cnfm	Derive tumor response confirmation as per RECIST	Littish	Unit Testing

Implementation

/ Backward compatibility considerations */*

The easiest method we found was to just add a new macro parameter that has no impact on the old ones

But a more efficient solution for programming run times could be to update existing macro parameters. In this case, use **regression testing**

- set up an area where you can run benchmark programs upfront using your initial macro
- then once macro is updated you can re-run these later to verify no impact

Influence on validation

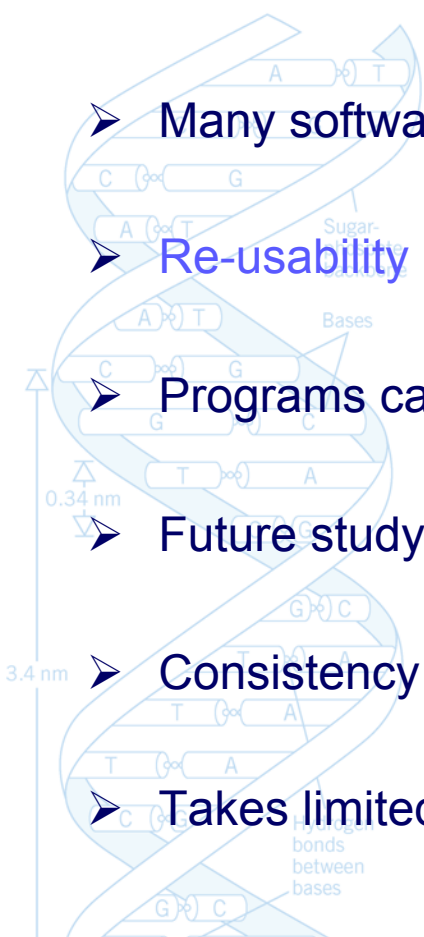
Using modular programming can not only make first line programming more efficient, but also **validation**

Firstly by re-using already validated macro modules the level of future QC can be reduced

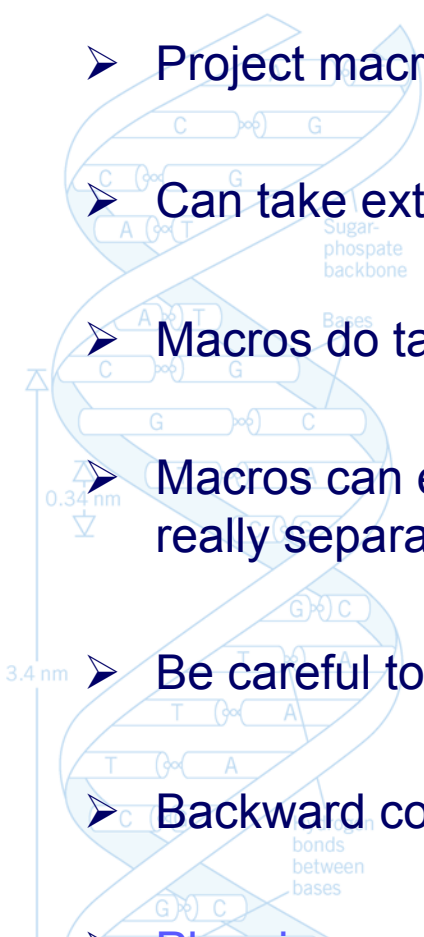
But even more beneficial in my opinion is when **unit testing** is used for initial QC

Modular programming and unit testing can go hand-in-hand to create **clear and robust programs** that can assure us of programming accuracy regardless of the input data. If you decide to adopt any of the approaches shown above then I highly recommend trying out unit testing too.

Benefits

- 
- Many software quality factors can be improved by modular programming
 - **Re-usability** is the key driver
 - Programs can easily be picked up, adapted and understood across all studies
 - Future study programs take significantly less time to produce and validate
 - Consistency of derivations across studies is increased
 - Takes limited SAS experience to create future study programs

Lessons learned

- 
- Project macros do need to be well managed (e.g. macro index)
 - Can take extra time for new starters on your project to pick up
 - Macros do take more initial time investment
 - Macros can easily be over-complicated with many different parameters, when really separate macros may have been the ideal
 - Be careful to ensure efficiency of the program
 - Backward compatibility should be in your forethoughts
 - **Planning** up front is key!

Conclusions

If modular programming is in the minds of the team from an early stage, then this approach definitely can give substantial future efficiency for just a little extra initial time spent

Then when you're thinking project-level remember it is highly beneficial to have project rules and requirements defined and considered to gain real benefits

Ensuring true re-usability and avoiding re-inventing rules means that we can re-produce and deliver **more timely** and with **greater consistency** of the rest of the project deliverables



AnyQuestions

Macro Modular QC BoundaryCases IntegrationTesting Implementation UnitTesting RegularCases TestCases