Randomly Generating Interviews to Validate a Complex Survey

Lindsay Walton, National Research Bureau, Auckland, New Zealand

1. Introduction

The New Zealand version of the worldwide mental health survey was an extremely complex instrument (> 3000 questions and a seemingly unlimited number of routings) being a combination of the original US survey together with many changes made by New Zealand. The instrument was divided into 22 reasonably self contained modules, each representing a different area of mental health. After finishing the initial coding changes we tried a number of ways to validate the instrument. These included:

- A walkthrough "translating" the Blaise code to the hard copy (which served as our specifications)
- Having testers try to go through as many routes in each module as possible and check the routings were consistent with the hard copy.
- Ran a pilot of around 200 interviews, noting any problems and then examining the number of responses to each question to see if some questions had been incorrectly missed out or if a particular question had an inappropriate number of responses when compared to another.

All of these methods were successful in finding problems which we were able to correct. The problem was that we were finding errors on a regular basis and with the sheer complexity of the routings, it became obvious that we would never be able to say with reasonable certainty that the instrument was clean.

We needed to find an alternative approach but at the same time, we were very aware that time was starting to become critical and we needed to get the survey into the field as soon as possible.

2. Overview of the problem

The main problem was the complexity of the routings which were virtually impossible to test using just a trial and error basis. Some examples are:

***NZSU73.0** INTERVIEWER CHECKPOINT (SEE *SU72 SERIES, ONLY
MARIJUANA SYMPTOMS (NZSU72..2, 72a.2,…72i.2)
ZERO TO TWO RESPONSES CODED '1'..... 1
ALL OTHERS..... 2

***SU73**. INTERVIEWER CHECKPOINT: (SEE ***SU72 SERIES – IGNORING NZ MARIJUANA QUESTIONS**)

ZERO TO TWO RESPONSES CODED '1'.....1 **GO TO *SU87**
ALL OTHERS.....2

---

*SU73.1 INTERVIEWER CHECKPOINT: (SEE ***SU47a** , ***SU47b, *SU47c, *NZRSU47e, *NZRSU47f**)

ONE OR MORE RESPONSES CODED '1'..... 1
ALL OTHERS..... 2 **GO TO *SU81**

PH100. INTERVIEWER CHECKPOINT:** (SEE ***D24a** - ***D24c, *D24e** - ***D24f, *D26a** - ***D26e, *D26g** - ***D26i, *D26j,*D26m, *D26o, *D26p, *D26r** ,D26s, *D26u-*D26w, *D26aa-*D26ee, *M1, *M8, *M5,** ***M7a-*M7o, *PD17a, *PD14, *SP16, *SP17, *SO2, *SO16, *SO17, *AG13, *AG14, *AG15, *AG16, *G17.1, *SD4, *SD6, *SD17, *SD19, *SR1, *SR2, *SR9.1**, ***SR10**, ***SR11**, ***SR12**, ***SR13**, ***SR14**, ***SR17** SERIES)

THE SUM OF THE COUNT EQUALS FIVE OR MORE BASED ON INCREMENTING THE COUNT BY ONE FOR EACH OF THE FOLLOWING NINE SETS OF QUESTIONS IF ONE OR MORE QUESTIONS IN THE SET ARE CODED '1': ONE OR MORE '1' CODES IN ***D24a** - ***D24c**, ONE OR MORE '1' CODES IN ***D24e** - ***D24f**, ONE OR MORE '1' CODES IN **(*D26a, *D26b, *D26c, *NZRD26d, *D26e)** , ONE OR MORE '1' CODES IN ***D26g** - ***D26i**, ONE OR MORE '1' CODES IN ***D26j**, ONE OR MORE '1' CODES IN (***D26m** OR ***D26o**), ONE OR MORE '1' CODES IN (***D26p** OR ***D26r** OR ***D26s**), ONE OR MORE '1' CODES IN ***D26u-*D26w**, ONE OR MORE '1' CODES IN ***D26aa-*D26ee** AND (***D9** EQUALS '1' OR ***D12** EQUALS '1') 1

M1** EQUALS '1' AND **M8 EQUALS '1'** AND (M9** EQUALS '4' or '5' OR

***M9a** EQUALS '1' OR '2' OR ***M33** EQUALS '1')..... 2

***M5** EQUALS '1' AND SUM OF '1' RESPONSES IN ***M7a-*M7o** IS '4' OR MORE

AND (***M9** EQUALS '4' or '5' OR ***M9a** EQUALS '1' OR '2' OR

***M33** EQUALS '1')..... 3

***SP16** EQUALS '4'-'5' OR ***SP17** EQUALS '1'..... 6

SO2** EQUALS '3' AND (SO16** EQUALS '4'-'5' OR ***SO17** EQUALS '1')..... 7

***AG13** EQUALS '1' OR ***AG14** EQUALS '1' OR ***AG15** EQUALS '4'-'5' OR

***AG16** EQUALS '1'..... 8

***G17.1** EQUALS '1'..... 9

***SD4** EQUALS '1' OR ***SD6** EQUALS '1' OR ***SD17** EQUALS '1' OR

***SD19** EQUALS '1'..... 11

***SR1** EQUALS '1' OR ***SR2** EQUALS '1'..... 12

ALL OTHERS..... 16

GO TO *PH101

*PH100.1 (SEE ***PH100**)

***PH100** EQUALS '1'-'15'..... 1 **THESE ARE SOME OF THE "LONG GROUP." GO TO *PT1**

ALL OTHERS 2

While most questions weren't this complex, there were many checkpoints throughout the survey, often with some complexity, and when considered together represented an instrument that was

difficult to manage.

3. Overview of the approach

The complexity of some of the modules meant that it wasn't realistic to do manual checking. Our solution was to automate the process using a 2 pronged approach.

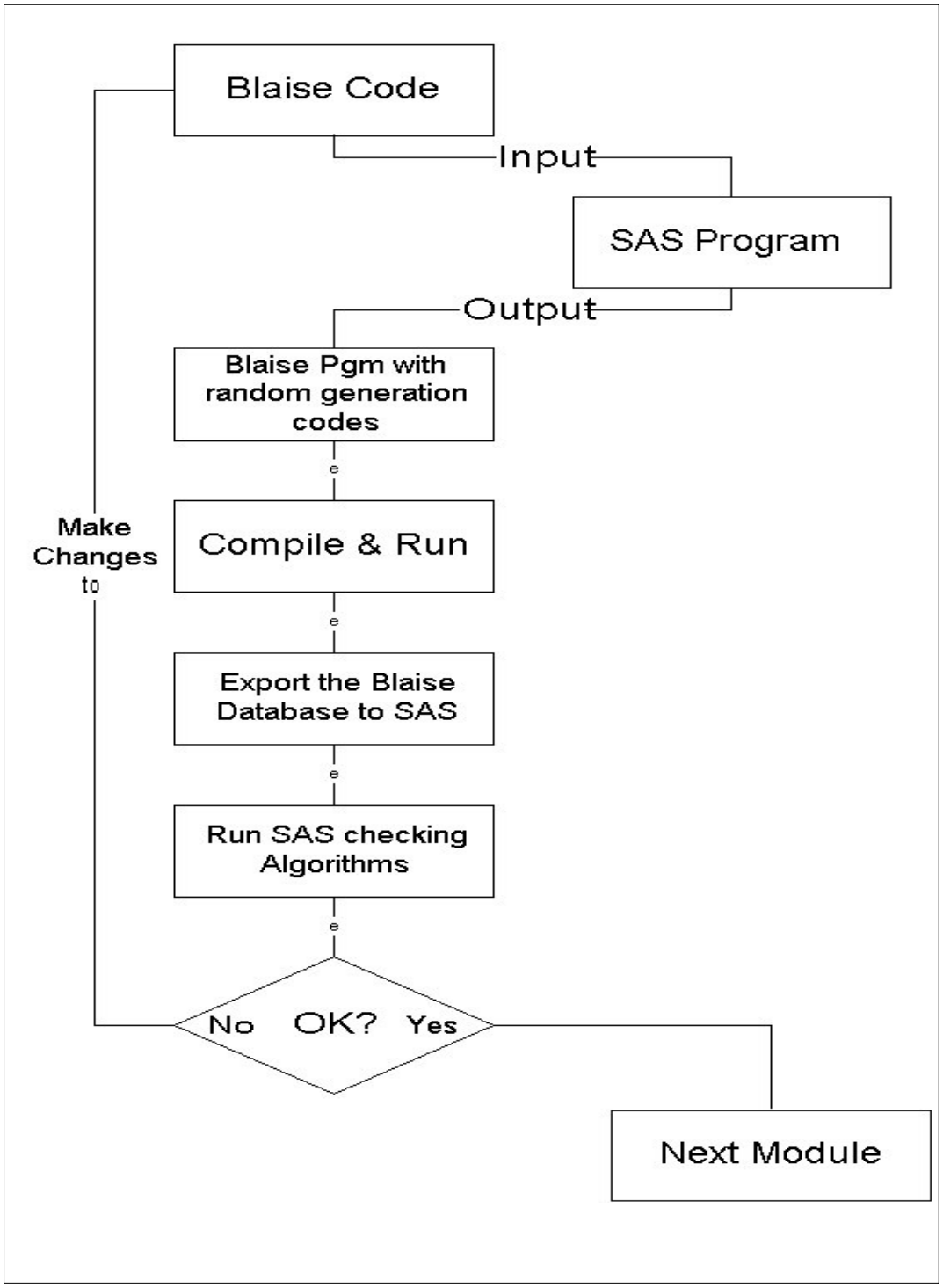
3.1 Generate random surveys

We felt somehow we needed to be able to generate a large number of surveys, using a random walk technique, so that we could feel reasonably confident that we had covered all or virtually all combinations of answers. To do this, we decided to try to convert all the questions in the survey into assignment statements using random generation. For example we needed to change a standard yes/no question into an assignment statement that would give yes 45% of the time, no 45%, and don't know and refused 5% each.

3.2 Check the surveys

The second part of the approach was to find a way of confirming these “interviews” were correct based on the logic of the paper version of the survey. For this to be successful, we felt it was important to change the emphasis of the checking from the routing based Blaise approach to a question-by-question analysis and using a different language, SAS. This was to reduce the chance of making the same mistakes in checking as were in the survey itself. For example, if the answer to question 1 is yes you ask question 2 otherwise you ask question 3. In our approach we would focus on each question and for question 3, decide that it should be populated if and only if question 1 is not yes, otherwise it must be empty.

3.3 Flow Chart of System



The above diagram shows that we needed to have 2 locations containing Blaise source code – one for the "normal" code and one, with exactly the same routing logic, but containing the random functionality. However it was vital that we only maintained one lot of source code. The SAS program at the top is how this was achieved. Essentially it read the "normal" Blaise code as an Ascii file, made some changes to the code and output it to another Blaise program containing

only assignment statements and no actual questions. This program is then compiled and run, producing a Blaise database containing anything from 200+ records. This database is then exported to a SAS dataset and then checked by the specially written SAS programs. In this way we could keep making changes to the "normal" Blaise code and only have to run a SAS program to produce the code to generate the random interviews. It was very quick and simple and meant we didn't have to make manual changes to both versions.

4. Randomly generating interviews – an example

The change in code for a simple Yes/No question was something like:

Original Module

```
LOCALS
...
FIELDS
...
SU2b2 "..." : TYesNo
...
RULES
...
SU2b2
...
```

After Random Generation

```
LOCALS
Rnum : INTEGER
...
FIELDS
...
SU2b2 "..." : TYesNo
...
RULES
...
{SU2b2}
 Prandom(101, Rnum)
 if RNum <=45 then SU2b2 :=C01
 elseif RNum <= 90 then SU2b2 :=C05
 elseif RNum <= 95 then SU2b2 :=DK
 else SU2b2 :=RF
 endif
...
```

Note that C01 is "yes" and C05 is "no". All that the SAS program did was comment out the original code and add the new code segment as well as adding the "RNum" field to the locals section.

The SAS program relied on certain consistencies in the programming standards of the Blaise modules. In general the tidier and more consistent the Blaise code, the easier the SAS job is. Probably the most important programming standard was "one question on one line" in the RULES section. This made it much simpler to identify the questions.

So how did the SAS program know that question SU2b2 is a Yes/No question? Essentially it made two passes of the Blaise module, the first pass read just the FIELDS section and matched the question code with its answer. In other words, it found SU2b2 and then skipped the text of the question and found its answer was the type TYesNo. This was written to a SAS Format and used in the second pass when it was changing the contents of the RULES section. The TYesNo answer was linked to a SAS macro that wrote the code in the box above.

This was the simplest situation: a question in which we were quite happy to have 45% of answers to be "Yes" and 45% to be "No". However there were other questions which needed a slightly different approach.

Ultimately the random answers had to reflect the logic of the survey and our desire to go down as many routes as possible. For example if the first question in a module is a Yes/No type and a "No" answer skips to the next module, then it is better to change the random amounts so that "No" has a very low chance of being selected. This is easy to do in SAS by a statement like

```
If ans='TYesNo' and Question in('SU1') then %yesno(85,5,5,5)
else if ans='TYesNo' then %yesno(45,45,5,5)
```

What is happening here is that for question SU1, the probability of it having an answer of "Yes" is 85% with "No", "Dont Know" & "Refused" at 5% each.

Another reason to change the randomising were for situations where 20 yes/no questions were asked and if the respondent answered "Yes" to 3 or more then they would go a different path to those who had fewer than 3. In this case the ratios can be changed so that "yes" has a lower chance of being selected, so ideally half the "interviews" would have 3 or more "yes" responses and half less than 3.

Multiple choice questions: These needed to be considered on a question by question basis. Some questions had upwards of 20 possible answers and generating all possible combinations of one question together with other long multiple response questions, together with all the other questions in the survey was not practical – and not necessary. For these questions, it was matter of assigning a small number of responses (5-10) that would be appropriate for any logic resulting from the answers to these questions. For example question SR105 had 14 answers:

```
SR105    "What kind of healer did you see?
           Record all mentions!" :
       TSpecialist
```

Where TSpecialist is:

```
TSpecialist = SET OF
        (C01 "Acupuncturist",
         C02 "Biofeedback specialist",
         C03 "Chiropractor",
         C04 "Energy healing specialist",
         C05 "Exercise or movement therapist",
         C06 "Herbalist",
         ...
         C13 "Dietician",
         C14 "Other (specify)" )
```

The SAS macro to generate the “randomised code” was

```
%macro mult14;
 %commentc;
 code=' Prandom(101, Rnum)'; output;
 code=' if Rnum <=15 then ' ||qst|| ':=[C01]'; output;
 code=' elseif Rnum<=30 then ' ||qst|| ':=[C02]'; output;
 code=' elseif Rnum<=45 then ' ||qst|| ':=[C14]'; output;
 code=' elseif Rnum<=60 then ' ||qst|| ':=[C07,C03]'; output;
 code=' elseif Rnum<=75 then ' ||qst|| ':=[C08,C04,C14]'; output;
 code=' elseif Rnum<=90 then ' ||qst||
':=[C14,C12,C13,C07,C06,C05,C04,C03,C02,C01]'; output;
 code=' elseif Rnum<=95 then ' ||qst|| ':=DK'; output;
 code=' else ' ||qst|| ':=RF'; output;
 code=' endif'; output;
%mend mult14;
```

resulting in this Blaise code:

```
{SR105}
 Prandom(101, Rnum)
 if Rnum<=15 then SR105 :=[C01]
 elseif Rnum<= 30 then SR105 :=[C02]
 elseif Rnum<= 45 then SR105 :=[C14]
 elseif Rnum<= 60 then SR105 :=[C07,C03]
 elseif Rnum<= 75 then SR105 :=[C08,C04,C14]
 elseif Rnum<= 90 then SR105
:=[C14,C12,C13,C07,C06,C05,C04,C03,C02,C01]
 elseif Rnum<= 95 then SR105 :=DK
 else SR105 :=RF
 endif
```

If the survey logic after this question takes people who have only seen one specialist down one route whereas those who have seen multiple go a different route, then this will work fine with 45% going the single specialist route and 45% the multiple with 10% of “Don't knows” and “refused” which also must be routed correctly.

4.1 Some Technical Issues

4.1.1 Compile problem (number 1)

When we finally had all the questions changed to random assignment statements, we found we were unable to compile the Blaise code as it requires a survey to have at least one question!! The way around this was to add a dummy question which would accept “empty” as a response. This enabled the program to compile but meant that couldn't run completely by itself. However this was easily overcome by placing a coffee mug on the space bar – it was all Blaise needed to complete an interview and restart a new one (but see 3.1.4 below)! The dummy question was:

```
intro2     "Press Enter" : STRING[1], EMPTY
```

4.1.2 – Compile problem (number 2)

When we ran a SAS program to rewrite a Blaise module and we already had a Blaise session open with the module open, Blaise would warn us that the contents of the module had changed and we would simply hit the “Ok” button. If the module wasn't open then Blaise gave no warnings. Then when we tried to compile, we got an error:

RULES or LAYOUT section terminated incorrectly or too long.

Initially this was very confusing because the code shown in the window had no obvious error. Although it took some time to find the problem, the solution was very simple – you just had to manually save the module in the Blaise command center. Then it compiled.

4.1.3 Adjustments to the controlling .BLA module

The survey had a key field which was called "IVNum" and was a string of length 5. Generating multiple interviews with unique keys would have required either writing a Manipula program or doing something with the date/time. We took the easiest option and temporarily commented out the PRIMARY and SECONDARY statements in the controlling .BLA program and just assigned the key field with the same constant for each interview.

4.1.4 Modelib Settings

In the modelib editor, under toggles --> Interviewing, ensure the following items were ticked:

Auto create file
Auto save when finished
Prompt save when finished
Auto enter

4.1.5 General

As the layout of the questions in the Blaise programs was not always consistent, the SAS program to read and make sense of them did get more complex than we were hoping.

5. Checking the interviews

This was done by running the SAS checking algorithms against the data that had been extracted from Blaise. In checking an "interview", the algorithm would search for an error, and if it found one, would report it and then exit to the next "interview". In this way it would only report the first error, and thus any subsequent errors in an interview would not be reported. Although this meant we took a few more iterations to clean a module, it meant that we didn't have to worry about the situation of errors in previous questions, incorrectly causing a later question to error.

Although there were a few different types of questions we had to deal with, the majority were handled by the three SAS macros below. If a problem is found, the macro "Fail" is called which outputs the record.

```
%macro fail(field, msg);
        do; errfield="&field"; errmsg=&msg; output; return; end;
%mend fail;

%macro check(field, z);
if (&z) then do;
        if (&field) eq . then %fail(&field, 'no value');
end; else do;
        if (&field) ne . then %fail(&field, 'has value');
end;
%mend check;

%macro value(field, val, z);
if (&field) ne . then do;
        if (&z) and ((&field) ne (&val)) then %fail(&field, 'bad value');
end;
%mend value;
```

an example of how these were used are:

```
data findErrors;
 set <randomly generated interviews from Blaise>;
 ...
 %check(m1, sc24 eq 1);
 %check(m2, m1 in(5,8,9));
 ...
 count=sum(m7a=1,m7b=1,m7c=1,m7d=1, m7e=1,m7f=1) ;
 %value(m8, 1, count ge 3);
 ...
run;
```

The first statement concerns question "M1" and requires that it must have a value (any value) if question SC24=1 otherwise it must equal missing.

Question "M2" is only asked if the answer to "M1" is either "No", "Dont Know", "Refused" ("Yes" in this survey was recorded as C01, "No"=C05).

The third example here is "M8" which is an assignment statement rather than a question. Its value must be 1 if the number of "Yes" responses to questions "M7a"-"M7f" is 3 or more.

6. Summary

The project was completed by 3 people in 3-4 weeks. Initially we thought we would run the random surveys in one hit for all modules and then see where the problem were. However it soon became obvious that it was preferable to do one module at a time. While the ultimate goal was to be able to run a random generation of many thousands of surveys, practically running 200-500 interviews for a module was generally enough although we would usually finish testing a module by running a survey with 2-3000 interviews. By the time it came to run the entire survey, 18000 interviews ran without error.

In the end we found 20-30 new routing errors. These were fixed and at the time of writing, 2000 real interviews had been sent to the USA for checking. The results were that the interviews had no routing errors and we just had some inevitable data cleaning work to do – which is an ongoing process.

This is a project that you would only consider for surveys with very complex routings. For simple surveys where one question flows into the next question, which can be checked manually, there is little need to do something like this to validate it.