Testing a Complex Blaise CAPI Instrument

John Fitzgerald, Central Statistics Office, Ireland

1) Introduction

The Central Statistics Office (CSO) currently uses CAPI interviewing for all of our Household Surveys. We have been developing Blaise Questionnaires and customising Blaise tools since 1997. Our Survey Instruments are generally used for longitudinal Surveys such as the Quarterly National Household Survey (QNHS) which has a 13 week development cycle each quarter and the Survey on Income and Living Conditions (SILC) which is developed for bi-annual release. We also provide Questionnaires for 'one off' Surveys such as the Programme for the International Assessment of Adult Competencies (PIAAC) in 2011 and the Household Finance and Consumption Survey (HFCS) in 2013.

This paper will demonstrate new testing practices and procedures we implemented during the development of the HFCS Questionnaire in 2012/2013. Our goals were to improve the efficiency and flexibility of our Questionnaire testing, while also improving the quality of the instrument and to ensure we were delivering the Questionnaire as specified.

I will present an overview of the testing throughout the development lifecycle, but the emphasis will be on Independent testing performed by the Blaise team on the completed versions of the Survey Instrument.

2) Planning & Implementation

The HFCS Instrument was one of the largest and most complex we have yet developed. A development period of 130 working days was originally estimated. The final Survey instrument that was developed contained over 850 variables. There was also in excess of 370 checks and signals. It contained a large number of text fills to cater for direct or proxy interviews and previous responses. The fills made the questions more relevant for the respondent but were time consuming to code and test.

Table 1 and Table 2 illustrates the size of the instrument and the challenges we faced in planning for the testing. It was important to achieve maximum test coverage even though we had limited resources and a limited amount of time.

Table 1 Technical description of the model HFCS2013 - Data Types

Data Fields types in DB	Total
Integer	1729
Real	106
Enumerated	3344
Set	114
Classification	0
Datetype	73
Timetype	48
String	922
Open	0

Table 2 Technical description of the model HFCS2013 - Fields

Field types	Total
Number of uniquely defined fields*1	918
Number of elementary fields*2	857
Number of defined data fields*3	6336
Number of defined block fields*4	61
Number of defined blocks	66
Number of embedded blocks	0
Number of block instances	487
Number of key fields	5
Number of defined answer categories	132
Total length of string fields	40010
Total length of open fields	0
Total length of field texts	99580
Total length of value texts	17539
Number of stored signals and checks	10127
Total number of signals and checks	10127
*1) All the fields defined in the FIELDS section *2) All the fields defined in the FIELDS section wich are not of type BLOCK *3) Number of fields in the data files (an array counts for more than one) *4) Number of fields of type block	

As this was a new project, it was decided to implement a more structured approach to software testing than we had used in the past. We devised metrics that would provide a quantitative measure of the testing. Realistic targets were set in order to achieve a high level of testing coverage within the resources and time constraints. Testing was scoped and all stakeholders were assigned responsibilities for each stage of the development life-cycle.

Table 3 illustrates how we customised Software engineering test levels into our own Bliase development life-cycle.

Table 3 - Test levels in the development lifecycle

Test level	Who	Test Type deployed	Techniques\methods
Requirements Testing	Development Manager in collaboration with programmer and author of requirements	Static testing (doesn't require code)	Reviews Walk-through of documentation flowcharts
Component Testing	Blaise Programmer tests their code.	Functional Testing (white box) Structural testing	Boundary Testing, Equivalence Partitioning, Statement Testing, Decision tables, Question text, variables, routing. We implemented peer-reviews and code reviews
Independent component Testing	Developer other than programmer	Functional Testing (black box)	Boundary Testing, Equivalence Partitioning, Decision tables, Question text, variables, routing, Computations. Informal Reviews
Integration testing	Developers - overseen and documented by team leader	Non-Functional testing (Black box)	Interaction between components of the system. Hardware v software tests Software design Use case testing
Acceptance testing	Business area and/or Interviewers	Non-Functional (Black box)	Verification of delivery on specifications

2.1 Test Levels

Throughout the development process, testing played a significant role. It was important to ensure that Questionnaire was being developed correctly and that it would meet the Business needs. The following is a description of the customised test levels that were used throughout the HFCS development lifecycle.

2.1.1 Requirements testing

The Questionnaire specifications and other work products around the requirements would become the baseline for the new formal testing process. Static testing techniques were employed to find errors or defects in the specifications. Static techniques are tests performed without executing any code. This approach finds errors and defects earlier in the life cycle of the project and this makes corrections easier and cheaper than finding those same defects later on in the project.

The development manager reviewed the specifications to find and remove errors and ambiguities in the document before they can become part of the executable code. This review process included informal and formal meetings with experienced programmers and the documents author(s) and other relevant stake-holders. The review objectives included: Finding defects, gaining understanding, generating discussion & forming consensus on the requirements.

2.1.2 Component testing

The Development Manager in collaboration with Senior programmers split the requirements document into logical blocks for coding [see figure 1]. This process was documented and the requirements were split and assigned to programmers.

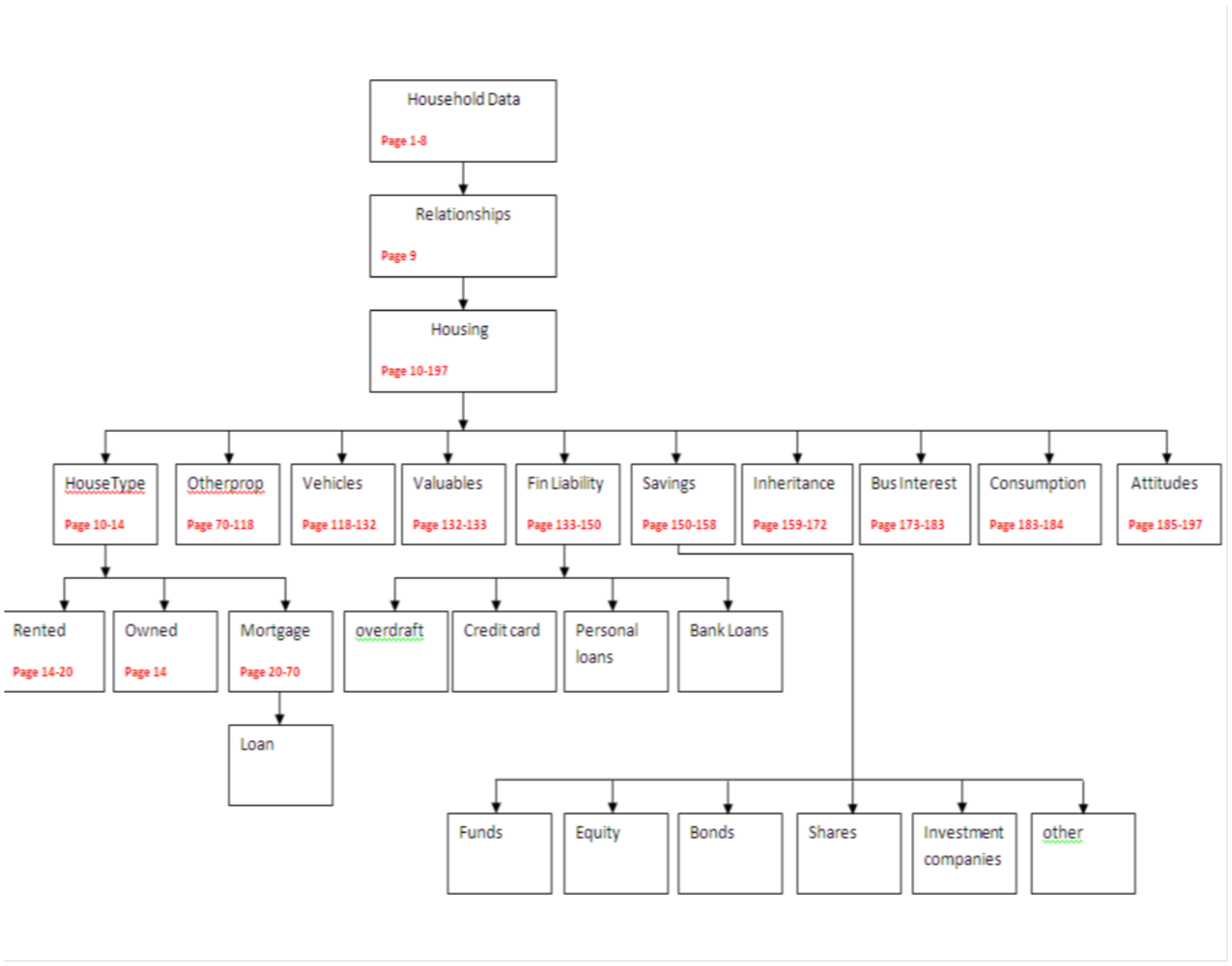
The programmers coded their assigned blocks from the specification document. Any issues or clarifications were logged and followed up by the Development manager at regular development meetings. The programmers performed component tests on their own code to ensure that the specifications were being fully met. Collaboration was encouraged and we also held informal code reviews attended by the code author, his/her peers and Manager.

Code reviews proved a very effective tool for the following reasons:

- Ensure Blaise standards have been implemented
- Best coding approach has been used
- Removal of bugs
- Forming consensus on requirements
- As a knowledge sharing exercise

It was important to establish with everyone involved in the review process that the emphasis should be on learning and improvement. Defects or deficiencies should be welcomed and suggestions or solutions should be expressed objectively.

Figure 1 Household level Requirements broken into manageable Blocks



2.1.3 Independent Component testing

Once a version of the Questionnaire was compiled and ready for Independent testing we could begin to complete the logs that were written from the specifications. The approach we adopted was 'Question-by-Question' testing where each variable in the specification document would be tested against the fully coded Questionnaire. While there was an awareness of the big commitment of time and resources to this approach, the trade-off of a high quality Questionnaire would be worth the effort.

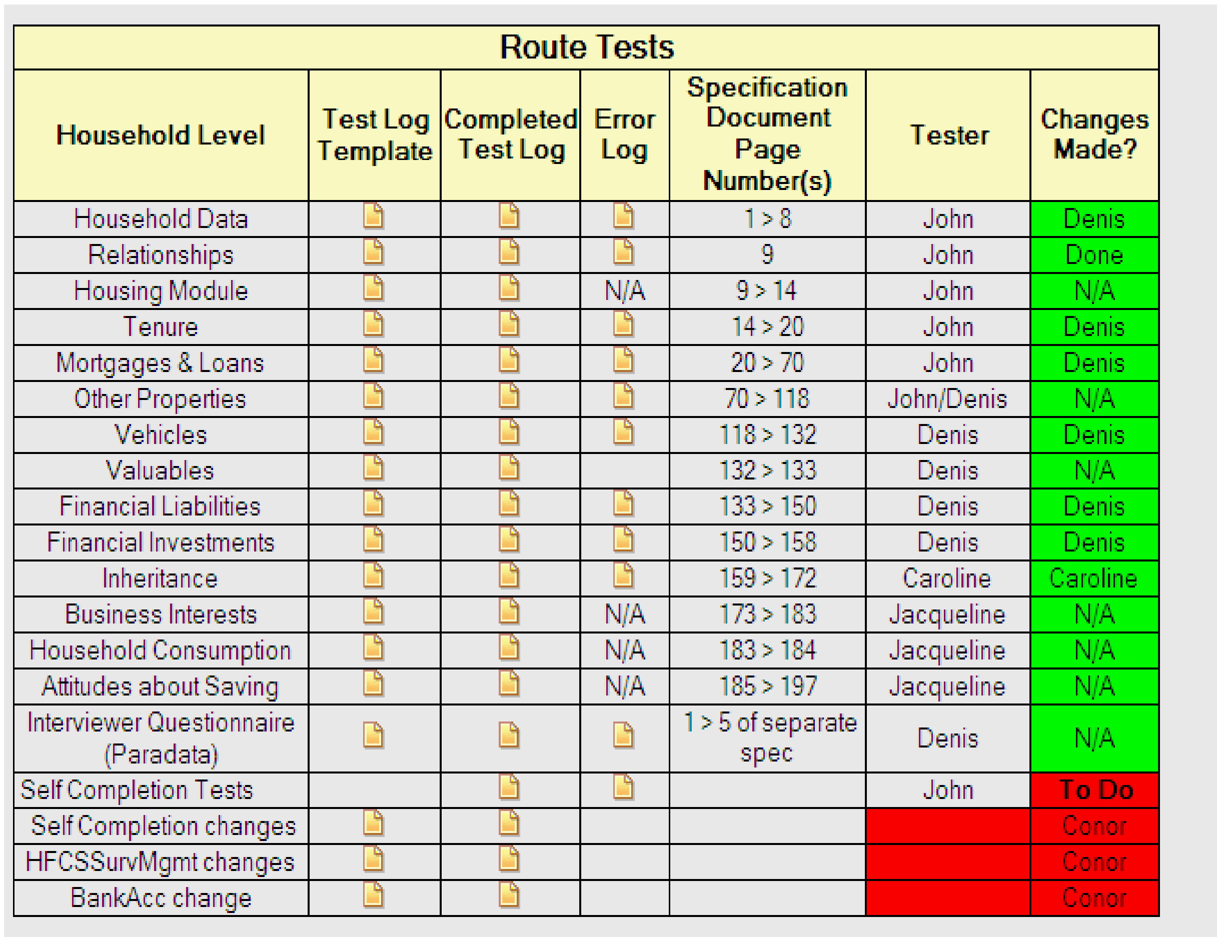
Based on our own experience and also using risk analysis techniques we prioritised logs [or subjects] where potential bugs were more obvious. For example, the routing and computations on the 'Mortgage & Loans' block was very complex, therefore there was a high probability of bugs which could potentially have a high negative impact on the functioning of the Questionnaire.

To manage testing, the questionnaire was divided into Household and Personal sections. A test log was created for each block and for each of the following test approaches:

- Routing
- Variable Ranges
- Fills/Inserts & Question text
- Errors/Signals
- Overall Appearance
- Computations/Errors/Signals/Don’t Know/Refusals

Every Block had potentially 6 test logs to be completed by an independent tester. We used a Lotus Notes repository to manage all testing documentation A table was devised for each test topic. A link was provided to each log template [Figure 2] which would be assigned or signed out by an Independent tester. Any bugs or issues raised by the tests were written into an error log [Figure 4] which was fed back to the developers.

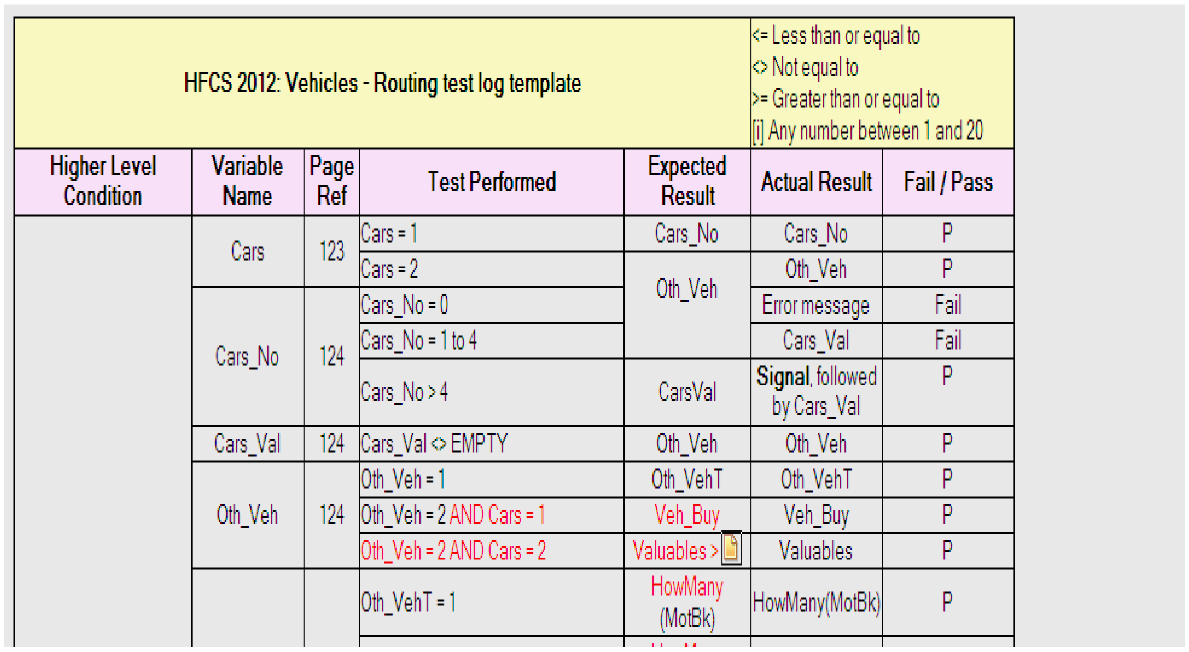
Figure 2: Routing Tests of Household Blocks



Route Tests						
Household Level	**Test Log Template**	**Completed Test Log**	**Error Log**	**Specification Document Page Number(s)**	**Tester**	**Changes Made?**
Household Data	📄	📄	📄	1 > 8	John	Denis
Relationships	📄	📄	📄	9	John	Done
Housing Module	📄	📄	N/A	9 > 14	John	N/A
Tenure	📄	📄	📄	14 > 20	John	Denis
Mortgages & Loans	📄	📄	📄	20 > 70	John	Denis
Other Properties	📄	📄	📄	70 > 118	John/Denis	N/A
Vehicles	📄	📄	📄	118 > 132	Denis	Denis
Valuables	📄	📄		132 > 133	Denis	N/A
Financial Liabilities	📄	📄	📄	133 > 150	Denis	Denis
Financial Investments	📄	📄	📄	150 > 158	Denis	Denis
Inheritance	📄	📄	📄	159 > 172	Caroline	Caroline
Business Interests	📄	📄	N/A	173 > 183	Jacqueline	N/A
Household Consumption	📄	📄	N/A	183 > 184	Jacqueline	N/A
Attitudes about Saving	📄	📄	N/A	185 > 197	Jacqueline	N/A
Interviewer Questionnaire (Paradata)	📄	📄	📄	1 > 5 of separate spec	Denis	N/A
Self Completion Tests		📄	📄		John	**To Do**
Self Completion changes	📄	📄			🟥	Conor
HFCSSurvMgmt changes	📄	📄			🟥	Conor
BankAcc change	📄	📄			🟥	Conor

Figure 2 shows the Routing tests for the Household blocks. The table contained a link to the log template. The tester signed and took a copy of the log template. The test log was then completed (along with the error log) and then attached back to the table.

Figure 3: Completed test log for Routing of Vehicles Block



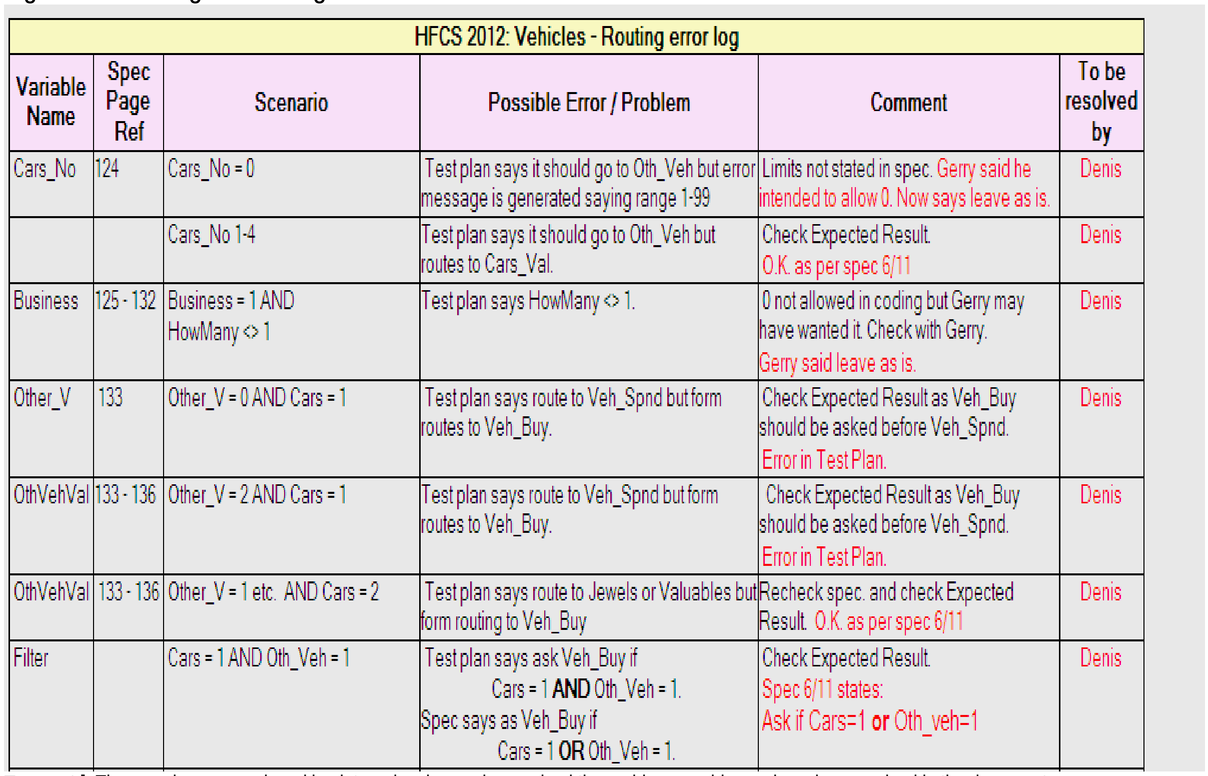
HFCS 2012: Vehicles - Routing test log template

<= Less than or equal to
<> Not equal to
>= Greater than or equal to
[i] Any number between 1 and 20

Higher Level Condition	Variable Name	Page Ref	Test Performed	Expected Result	Actual Result	Fail / Pass
	Cars	123	Cars = 1	Cars_No	Cars_No	P
			Cars = 2	Oth_Veh	Oth_Veh	P
	Cars_No	124	Cars_No = 0		Error message	Fail
			Cars_No = 1 to 4		Cars_Val	Fail
			Cars_No > 4	CarsVal	**Signal**, followed by Cars_Val	P
	Cars_Val	124	Cars_Val <> EMPTY	Oth_Veh	Oth_Veh	P
	Oth_Veh	124	Oth_Veh = 1	Oth_VehT	Oth_VehT	P
			Oth_Veh = 2 AND Cars = 1	Veh_Buy	Veh_Buy	P
			Oth_Veh = 2 AND Cars = 2	Valuables >	Valuables	P
			Oth_VehT = 1	HowMany (MotBk)	HowMany(MotBk)	P

Figure 3 shows a completed test log for Routing tests that were performed on the Vehicles block. If a test was marked 'fail' the variable details along with the test performed were written to the error log [figure 4

Figure 4: Error Log for Routing of Vehicles Block



HFCS 2012: Vehicles - Routing error log

Variable Name	Spec Page Ref	Scenario	Possible Error / Problem	Comment	To be resolved by
Cars_No	124	Cars_No = 0	Test plan says it should go to Oth_Veh but error message is generated saying range 1-99	Limits not stated in spec. Gerry said he intended to allow 0. Now says leave as is.	Denis
		Cars_No 1-4	Test plan says it should go to Oth_Veh but routes to Cars_Val.	Check Expected Result. O.K. as per spec 6/11	Denis
Business	125 - 132	Business = 1 AND HowMany <> 1	Test plan says HowMany <> 1.	0 not allowed in coding but Gerry may have wanted it. Check with Gerry. Gerry said leave as is.	Denis
Other_V	133	Other_V = 0 AND Cars = 1	Test plan says route to Veh_Spnd but form routes to Veh_Buy.	Check Expected Result as Veh_Buy should be asked before Veh_Spnd. Error in Test Plan.	Denis
OthVehVal	133 - 136	Other_V = 2 AND Cars = 1	Test plan says route to Veh_Spnd but form routes to Veh_Buy.	Check Expected Result as Veh_Buy should be asked before Veh_Spnd. Error in Test Plan.	Denis
OthVehVal	133 - 136	Other_V = 1 etc. AND Cars = 2	Test plan says route to Jewels or Valuables but form routing to Veh_Buy	Recheck spec. and check Expected Result. O.K. as per spec 6/11	Denis
Filter		Cars = 1 AND Oth_Veh = 1	Test plan says ask Veh_Buy if Cars = 1 **AND** Oth_Veh = 1. Spec says as Veh_Buy if Cars = 1 **OR** Oth_Veh = 1.	Check Expected Result. Spec 6/11 states: Ask if Cars=1 **or** Oth_veh=1	Denis

Figure 4]. The error log was assigned back to a developer who resolved the problem or addressed any issues raised in the document.

2.1.3.1 Test Design Techniques used to create Test Logs for Independent Component testing

For our Independent Component Testing we used Specification-Based (Black box) techniques to derive our test cases. The test logs were designed from the specifications - independently of the developers to ensure that the specifications have been implemented fully and both processes have arrived at the same results.

We employed a number of Software testing techniques to derive and prioritise Test cases for the Logs. The techniques were to be employed as aids to derive test cases for each of the approaches to testing.

Routing	- Decision Tables, Flow Charts/State Transition tables
Variable Ranges	- Equivalence Partitioning, Boundary Value analysis
Fills/Inserts	- Use Case testing
Question text/Appearance	- Use Case testing

2.1.3.1.1 Equivalence Partitioning

Equivalence partitioning is based on a very simple idea: Inputs into a program can be classified into groups of similar inputs. For example a variable defined as an Integer will accept as valid any input that is numerical and will reject anything else [characters, symbols]. The range of numbers is infinite (though computers will limit the Integer to a finite definition). Equivalence partitioning requires us to test once an input from the valid partition (any number) and a representative input from the invalid partition (character etc). This limits the number of tests we need to perform to fully test the variable limits

Example:

Test condition: valid inputs are integers in the range 100 to 999 inclusive

These are the test cases for Equivalence partition tests:

- Valid partition 100 to 999 inclusive eg 450

- Non valid partitions: Integer less than 100 eg 55

Integer greater than 999 eg 1055

Decimal numbers eg 1.5

Non-numeric characters eg. woe

While we cannot test for every valid partition (Keying every integer between 100 - 999 inclusive) we should do at least one test with a valid value.

there are 5 test cases (+) derived from Equivalence partition tests on the above test condition:

```
----++--------+-------|---------------+------------------|----+----------
 Decimal/Alpha 55       100           450            999  1055
       Invalid partition        Valid Partition          Invalid Partition
```

2.1.3.1.2 Boundary value Analysis

Boundary value analysis is based on testing at the boundaries between partitions. Here we have both valid boundaries (in the valid partitions) and invalid boundaries (in the invalid partitions).

Example:

Test condition: valid input: integers in the range 100 to 999(inclusive)

Valid Boundary - 100 and 999

Invalid Boundary - 99 and 1000

So there are 4 test cases (+) derived from the test condition in the boundary test

```
-----------+|+--------------------------------------------+|+---------------
            99;100                                      999;1000
```

Incorporated into boundary techniques was creating the test cases for imputation fields of the derived variables to ensure the elimination of Imputation errors. We also created test cases for Don’t Know/Refusal answers that could potentially cause defects when the answer is cross-checked or referenced with other values.

2.1.3.1.3 Decision Tables

Specifications define the conditions under which a function operates. The conditions can get quite complex so it's important to try to ensure that every combination of the conditions has been tested.

A decision table lists all the input conditions and all the actions that arise from them. The conditions are structured into tables as rows and below the conditions are the resulting actions that arise from the combination of true/false conditions in the top part of the table.

Example:

The Credit Card details part of the Financial Liabilities module is only to be asked of People aged 18 (Condition 1) or over, who are Students (Condition 2) or working (Condition 3) and possess a Credit Card (Condition 4)

It might be coded as follows:

```
IF (Age >= 18) and (Credit Card = Yes) and ((Job = Student) or (Job = Working)) Then
   Ask Module
Else
   Goto End
Endif
```

In order to achieve the greatest coverage of all possible conditions, the tester should draft and refine a table something like the following:

	HFCS2013: Test cases for Financial Liabilities module									
	Rule 1	**Rule 2**	**Rule 3**	**Rule 4**	**Rule 5**	**Rule 6**	**Rule 7**	**Rule 8**	**Rule 9**	**Rule 10**
Conditions										
Aged >= 18	Yes	Yes	Yes	Yes	No	No	No	No	Yes	Yes
Students	No	Yes	Yes	Yes	Yes	No	No	No	No	Yes
Working	Yes	No	Yes	Yes	Yes	Yes	No	No	No	No
Credit Card	Yes	Yes	No	Yes	Yes	Yes	Yes	No	No	No
Actions										
Ask Module	Yes	Yes	-	Yes	-	-	-	-	-	-
End Module	-	-	Yes	-	Yes	Yes	Yes	Yes	Yes	Yes

Second reduction of table will remove duplicate tests (cases where respondent is not 18 or over)

	HFCS2013: Test cases for Financial Liabilities module						
	Rule 1	**Rule 2**	**Rule 3**	**Rule 4**	**Rule 5**	**Rule 6**	**Rule 7**
Conditions							
Aged >= 18	Yes	Yes	Yes	Yes	No	Yes	Yes
Students	No	Yes	Yes	Yes	Yes	No	Yes
Working	Yes	No	Yes	Yes	Yes	No	No
Credit Card	Yes	Yes	No	Yes	Yes	No	No
Actions							
Ask Module	Yes	Yes	-	Yes	-	-	-
End Module	-	-	Yes	-	Yes	Yes	Yes

Third reduction removes all duplicate tests for Credit Card=No

	HFCS2013: Test cases for Financial Liabilities module				
	Rule 1	**Rule 2**	**Rule 3**	**Rule 4**	**Rule 5**
Conditions					
Aged >= 18	Yes	Yes	Yes	Yes	No
Students	No	Yes	Yes	Yes	Yes
Working	Yes	No	Yes	Yes	Yes
Credit Card	Yes	Yes	No	Yes	Yes
Actions					
Ask Module	Yes	Yes	-	Yes	-
End Module	-	-	Yes	-	Yes

Rules 1 - 5 in the last decision table are the test cases which should be performed by the tester to provide maximum coverage for all possible combinations of conditions. The Expected results of each test are the actions of each rule. This technique is particularly useful in systems where combinations of input conditions produce various actions.

2.1.3.1.4 Flow Charts \ State transition Diagrams

Although generally only used to represent code, we encouraged our test log authors and programmers to draw the specifications using flow charts. This helped to get a greater understanding of the routing and validations in the specification and also proved to be a useful aid in generating test cases.

We adapted and used state transition diagrams to represent the routing through blocks also. This was a useful technique when we were developing our Use cases for scenario testing.

2.1.3.1.5 Use Case [or Scenario] Testing

Based on results from previous household Surveys we were able to design household scenarios to ensure correct routing through the household profile. While we were aware that a comprehensive list of all possible scenarios was not achievable, we could direct our tests based on households that took part in previous household surveys and weight our tests according to those profiles. This method of testing in conjunction with Question-by-Question testing helped us to ensure as near as possible to full testing coverage of the Questionnaire.

2.1.4 Integration testing

The purpose of Integration testing was to expose defects between the Survey Instrument and all of the other system components and interfaces. Workflows and Use case scenarios were the test bases and these formed the strategy for ensuring that the system was functioning as it should. The focus of the integration test was to ensure that all components and interfaces were interacting correctly.

Non-functional requirements were also tested at this level:

- Installability – installation procedures
- Maintainability – ability to introduce changes
- Performance – expected behaviour
- Load & Stress handling – System behaviour at upper limits of usage and data load
- Recovery – Procedures in the event of failure
- Usability – ease with which users can engage with the system

2.1.5 Acceptance testing

These tests were performed by the business users. The purpose was for verification that the requirements had been fully met and that the system provided for the Business needs. This testing was performed independently of IT. The Business area performed their own scenario testing to ensure compliance with the specifications.

3) Results

Independent testing of the Questionnaire was also performed by the Blaise team throughout the latter stages of the development lifecycle. During Integration testing we were simultaneously testing and re-testing the questionnaire to ensure all amendments and fixes had been implemented correctly.

There were over 80 test logs produced from the specifications. It took 2 people approximately 3 weeks to document the specifications and to devise all tables, logs and other documents required for testing. The logs were prioritised in terms of complexity and risk. It took 3.5 Independent testers 15-20 days to complete the logs. Testing and re-testing continued until Questionnaire sign-off which was 1 week before the Questionnaire was released for pilot testing.

Testing documentation was reviewed and updated throughout the lifetime of the testing process. Any changes resulting from the pilot were incorporated into the test logs. The pre-live Questionnaire was released to the testers again for another iteration of testing.

At the outset of test planning we established exit criteria to define when the testing was complete. Our Primary target was that every log should be completed (in priority order) within the time assigned. We assigned extra testers to achieve this. It was critical that all incidents raised were corrected, retested and signed off – or waived by the Development Manager.

Independent Questionnaire testing ensured no critical problems occurred in the field. The commitment of 20% of development solely to testing meant that we released a robust Questionnaire to the field on schedule. The only problems logged to the Helpdesk were generally of a ‘User education’ or training nature.

4) Conclusion

The biggest challenge of the testing process was documenting the Specifications into logs for all of the test approaches. The logs were designed so that they could be given to any member of staff and he/she would be able to launch the Questionnaire and complete the log easily.

The process proved very profitable in terms of Quality assurance. The approach developed for the HFCS has become a valuable template for any new development work performed by the CSO’s Blaise team.

References & Further Reading

"Methods for Testing and Evaluating Survey Questionnaires" - Presser, Rothgeb, Couper, Lesser, Martin, Martin, Singer. ISBN 978-0-471-45841-8

"Software Testing: An ISTQB-ISEB Guide – Brian Hambling (Editor) ISBN 978-1-906124-76-2

"Software Testing in the Real World" – Edward Kit 1995 ISBN 978-0201877564