IBUC 95

3th International Blaise Users Conference
Helsinki Finland
18 – 20 September 1995

Table of Contents

Macro-Editing with Blaise III

Jelke Bethlehem and Lon Hofman, Statistics Netherlands

1. Introduction

Sample surveys are carried out to collect information about a specific population. Results are presented in the form of statistics, i.e. estimates of unknown population characteristics. The statistics will rarely correspond exactly to the unknown values of the population characteristics. Every survey operation is affected by errors, and the magnitude of these errors affects the accuracy of the results.

Generally, two broad categories of errors are distinguished: sampling errors and nonsampling errors. Sampling errors are due to the fact that only a sample is observed and not the entire population. Every replication of a survey experiment would result in a different sample, and therefore in different estimates. Sampling errors will not occur in a census (a complete enumeration of the population). The survey researcher is in control of the sampling error. By making a proper sampling design, and selecting a sufficiently large sample, unbiased estimates can be obtained with small standard errors. Of course, the available financial budget may impose restrictions on the size of the sample.

Nonsampling errors relate to all survey errors originating from sources other than the use of a sampling mechanism. Such errors will also occur in a complete enumeration of the population. Examples of nonsampling errors are errors due to nonresponse, measurement errors (e.g. wrong answers), coverage errors, and processing errors. Nonsampling errors are difficult to control. The best way to deal with nonsampling errors is to take preventive measures at the design stage. One can think of well-considered selection of the sampling frame, careful design of the questionnaire, adequate training of interviewers, and thorough pilot surveys. Nevertheless, many causes of nonsampling errors are inherent in the survey process, and as such are virtually impossible to avoid.

Nonsampling errors have an impact on the quality of survey results. To avoid publication of inaccurate statistics, the collected data must undergo extensive treatment. This treatment is called the data editing process. Aim of data editing is to detect and remove errors in the data. Traditionally, data editing was a manual process, carried out by subject-matter experts. It was costly and time-consuming, and not very effective. The introduction of computers for data editing improved the situation dramatically. Particularly, systems like Blaise provide a powerful and user-friendly environment for extensive error checking, and assisting the subject-matter experts in correcting these errors.

Many systems for data editing are micro-editing systems. *Micro-editing* means that each questionnaire form is processed separately. Checks compare the answers within one form, and do not take into account the results on other forms. The micro-editing approach is able to detect many errors, but not those relating to the distribution of answers over the forms. For example, it is not possible to detect outliers, i.e. answers that are within the range of valid answers, but which differ so much from other answers that probably something is wrong. Such errors can only be detected if all answers to the relevant questions are available.

The data editing approach dealing with distributional aspects of the answers to the questions is called *macro-editing*. Typically, macro-editing is a file-oriented approach, whereas micro-editing is a form-oriented approach. Therefore, macro-editing can not be carried out during data collection, but only after all (or a large part of the) data has been collected.

The Blaise system was originally designed as a form-oriented system. The idea was to integrate a number of activities required to collect, enter, and edit survey data. The form-oriented approach of Blaise makes the system very suitable for micro-editing. This papers describes how Blaise can also be used for macro-editing activities. Using Blaise tools like the DEP and Manipula, we describe simple ways to implement some forms of macro-editing procedures.

Section 2 describes macro-editing in some more detail. Particularly, it is explained why users should sometimes not concentrate too much on micro-editing, and pay more attention to macro editing. Sections 3 and 4 discuss some techniques for macro-editing. Section 5 describes how some of these techniques can be implemented in the current version of Blaise. Section 6 describes a prototype of a special data viewer that allows for graphical macro-editing.

This paper should not be seen as a document containing the specifications of modules that will be in the next release of Blaise. It is discussion paper proposing a possible future way to extend the system. The authors appreciate any comments on the contents of the paper.

2. Why macro-editing?

Traditionally, most survey data is collected by means of paper questionnaire forms. In the process of asking the questions, writing down the answers, and entering the information into the computer, many thing can go wrong. Respondents may not understand a question, and therefore give a wrong answer. They may understand the question, but do not want to give the right answer (e.g. for sensitive questions). Furthermore, interviewers can mistakes in writing down the answers, and data entry operators can make errors when entering the data. Consequently, the information in the data file does always not reflect the real situation being investigated. Analysis of such data may lead to wrong conclusions. Therefore, it is vital to detect and correct any errors. This process is called editing.

The traditional approach to editing is a manual form of micro-editing. After collection of the forms, subject-matter specialists check the forms for completeness. If necessary and possible, skipped questions are answered, and obvious errors are corrected on the forms. Sometimes, the forms are manually copied to a new form to allow for the subsequent step of data entry. Next, the forms are transferred to the data entry department. Data typists enter the data in the computer at high speed without much error checking. Usually, only some simple range checks are carried out to detect data entry mistakes.

After data entry, an error detection program is run. Detected errors are printed on lists. The lists with errors are sent to the subject-matter experts. They investigate the error messages, consult corresponding forms, and correct errors on the lists where possible. Lists with corrections are sent back to the data entry department, and data typists enter the corrections, after which corrected records and already accepted records are merged.

Usually, the cycle of batch-wise error detection and manual correction is repeated a number of times, until the number of rejected records is sufficiently small.

The data editing procedure described here, has it limitations. Since different departments and different computer systems are involved in a cyclic process, it is time-consuming. Furthermore, the manual activities by the subject-matter experts are costly, and not very effective.

To improve the efficiency of the statistical production process, and the quality of the produced statistical information, Statistics Netherlands developed different approaches to data collecting and data editing.

CADI (Computer-assisted data input) was applied in economic surveys. The idea was to improve the handling of paper questionnaire forms by integrating data entry and data editing tasks. The traditional batch-oriented data editing activities, in which the complete data set was processed as a whole, was replaced by a record-oriented process in which records (forms) were completely dealt with one at a time. Basically, CADI is a form of micro-editing. CADI was implemented in version 1 of Blaise. It is used in two ways.

Subject-matter specialists take care of both data entry and data editing, thus avoiding the cycle through different departments.

Data typists use the CADI system to enter data without much error checking. After completion, the CADI system checks in a batch run all records, and flags the rejected ones. Then subject-matter specialist handles the rejected records one by one, and correct the detected errors (also with a CADI system).

Computer-assisted interviewing (CAI) techniques are applied in social surveys. The paper questionnaire is replaced by a computer program containing the questions to be asked. The computer takes control of the interviewing process.

The computer program determines the route through the questionnaire. It determines which question is to be asked next, and displays that question on the screen. Such a decision may depend on the answers to previous questions. Hence it relieves the interviewer of the task of taking care of the correct route through the questionnaire. As a result, it is not possible anymore to make route errors.

The computer program also checks the answers to the questions which are entered. Range checks are carried out immediately after entry, and consistency checks after entry of all relevant answers. If an error is detected, the program gives a warning, and one or more of the answers concerned can be modified. The program will not proceed to the next question until all detected errors have been corrected.

The CAI approach also implements a form of micro-editing. The big difference with the CADI approach is that micro-editing is moved from the statistical office to the field. All micro-editing is taken care off during the interview, so that the statistical office only receives 'clean' (accepted) forms. Almost no editing activities are left for the office. CAI was implemented in version 2 of Blaise.

Systems like Blaise are very powerful instruments for micro-editing. They have simple facilities to specify a large number of checks. Checks may also analyse complex relationships between many questions. Application of these micro-editing systems has proved to be successful. They are capable of detecting and correcting many errors, and therefore they improve data quality. However, micro-editing also has some drawbacks.

The first point to be mentioned is the risk of 'over-editing'. With systems like Blaise it easy to implement almost any check one can think of. On the one hand, this seems to be a nice feature, based on the idea that the more checks are carried out, the more errors will be corrected. However, there are also disadvantages:

- It is possible to implement checks that contradict one another. This will cause all records to be rejected.
- It is possible to implement redundant checks. In such a situation, one problem may result in many error messages. This will make the work of the subject-matters workers or the interviewers very difficult.
- It is possible to implement checks that detects problems that almost have no impact on the quality of the published statistical information. Such checks generate a lot of work that does not pay off.

The second point to be made with respect to micro-editing, is that this approach is not capable of detecting all problems. One example of such a problem is outliers. An outlier refers to a value of a variable (answer to a question) that is within the defined range of valid values, but is very unlikely when compared with the distribution of all valid values. An outlier can only be detected of the distribution of all values is available, and for that one needs the whole file with records.

Macro-editing offers a solution to some of the problems of micro-editing. Particularly, it can deal with editing relating to the distributional aspects. Like in micro-editing, checks can take all kinds of forms in macro-editing. Here, two forms are summarised. The first form is sometimes called the 'aggregation method'. See also Granquist (1990). The idea is to divide the data file in groups, and to compute aggregate statistics for each group. The distribution of the values is analysed (for example with the techniques of section 3), and only if an unusual value is observed, a micro-editing procedure is applied to the individual records in the group. The advantage of this form of editing is that it concentrates on detecting errors that have an impact on the final results of the survey. No superfluous micro-editing activities are carried on records in groups that do not produce unusual values at the aggregate level.

A second form of macro-editing could be called the 'distribution method'. The available data is used to characterise the distribution of the variables. Then, all individual values are compared with the distribution. Typically, measures of location and spread are computed. Records containing values that could be considered uncommon (given the distribution) are candidates for further inspection and possible editing.

Both forms of macro-editing involve a check component that requires the data file to be available, and a correction component requiring the individual records.

The rest of this paper concentrates on the 'distribution method'. Several techniques to characterise the distribution of one ore more variables are discussed in the subsequent sections. Some of these techniques can implemented in the Blaise system. This is described in sections 5 and 6.

3. Macro-editing techniques for one variable

Many macro-editing techniques analyse the behaviour of a single observation in the distribution of all observations. We will discuss some of these techniques. We restrict ourselves to the analysis of quantitative variables. Quantitative variables measure a size, amount, or value. Typically, it is possible to compute quantities like sums and averages. We will not discuss qualitative variables. These variables divide the observations in groups. Values are group labels. It is not useful to carry out arithmetic operations on the values. Furthermore, we will restrict ourselves to the analysis of a single variable, and to the analysis of the relationship between two variables.

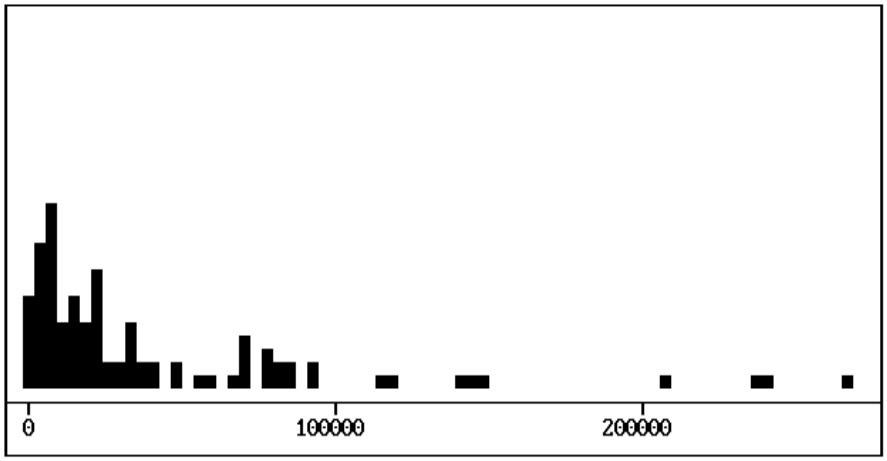
There is an area in statistics providing all kinds of techniques for analysing the distribution of variables, and that is exploratory data analysis (EDA). See for example Tukey (1977). Many of the techniques found here can be applied in macro-editing. Here we concentrate on two groups of techniques. The first group analyses the distribution of a single variable, and concentrates on detecting outliers. The second group analyses the relationship between two variables, and tries to find records showing a different relationship. All techniques discussed can be seen as special cases of the 'distribution method' of macro-editing.

Advocates of EDA stress the importance of the use of graphical techniques. These techniques provide much more insight in the behaviour of variables than numerical techniques do. This also applies to macro-editing. Graphs of the distribution of the data show a lot of information, and are capable of showing unexpected properties that would not have been discovered if just numerical quantities were computed.

In this section, we describe some techniques for the analysis of the distribution of a single quantitative variable. These techniques characterise the distribution by a measure of location, and a measure of spread around this location. They also attempt to detect values with a different behaviour. This special values are called 'outliers'. A typical macro-editing application will concentrate on the analysis and possible treatment of these outliers.

A technique that displays the distribution in its most elementary form, is the *one-way scatterplot*. In such a plot each individual value of the variable is displayed on a horizontal scale. Figure 3.1 shows an example.

Figure 3.1. One-way scatterplot of the manure production



The variable displayed in the plot is the yearly manure production by farms in 98 municipalities in the Dutch province of Zuid-Holland. Each small square represents the manure production within one municipality. Where rounded values coincide, the corresponding squares are stacked. A one-way scatter plot can be used to analyse the following properties of the distribution:

- *Outliers*. Outliers appear as single squares that are far apart from the rest of the observations. Outliers require further analysis, since the may indicate wrong values;

- *Grouping*. Grouping means that values are not more or less evenly spread over the whole domain of possible answers, but instead appear to be concentrated in separate groups. Grouping may be an indication the observations originate from differently behaving subpopulations, and therefore it might be more appropriate to analyse these groups separately;

- *Concentration.* Only if the values concentrate around a certain location, it is acceptable to use this location to characterise the distribution;

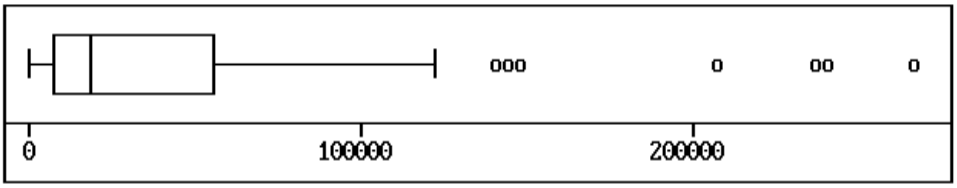
- *Symmetry*. Only if the distribution is symmetric and concentrates around one location, it is acceptable to use numerical techniques assuming normality.

If we look at our example, we see four values at the right-hand side of the distribution that might be considered outliers. There is no grouping of observations. The distribution is far from symmetric, making it very difficult to use numerical quantities like means and standard deviations to characterise the distribution.

For asymmetric distributions, one might consider a transformation. If the transformed distribution resembles the normal one, more techniques are available for analysing the distribution. In case of skew distributions like the one in the example, taking logarithms or square-roots might help.

A next technique to study the distribution the values of a single variable is *the box-and-whisker plot*. The box-and-whisker plot displays the shape of the distribution in a schematic way, without showing all individual values. Figure 3.2 contains an example of a box-and-whisker plot.

Figure 3.2. The box-and-whisker plot of the manure production



The rectangular box represents the middle part of the distribution. It extends from the *first quartile* to the *third quartile*. The line within the box is the *median* (the second quartile). The whiskers connect the box to the so-called *adjacent values*. The *upper adjacent value* is defined to be the largest value less than or equal to the third quartile plus 1.5 times the length of the box. Likewise, the *lower adjacent value* is smallest value greater than or equal to the first quartile minus 1.5 times the length of the box. Any value falling outside the interval between the two adjacent values is considered to be an outlier, and therefore plotted as an individual point. The following aspects of the box-and-whisker plot are worth taking a look at:

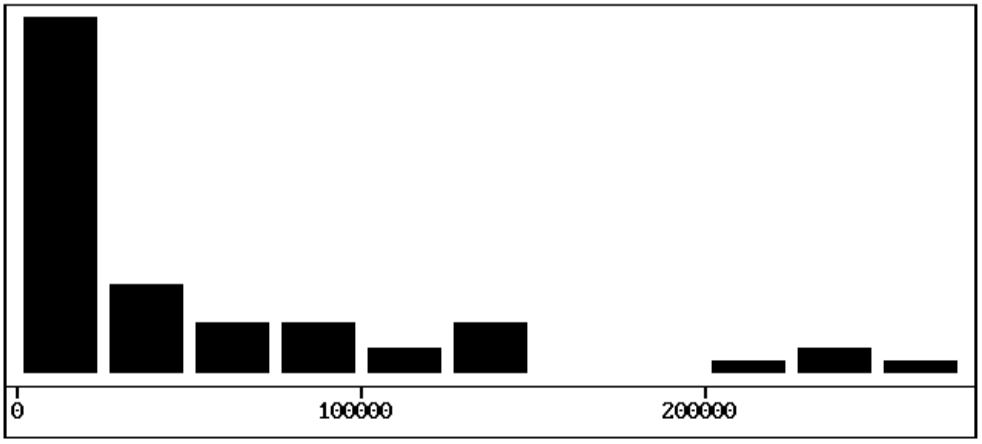
- *Outliers*. Outliers appear as single points. These points require further analysis, since the may indicate wrong values;

- *Symmetry*. If the box-and-whisker plot is symmetric, the underlying distribution is symmetric. Only if the distribution is symmetric and concentrates around one location, it is acceptable to use numerical techniques assuming normality;

- *Length of whiskers*. If the length of the whiskers is much shorter or much longer than the 1.5 times the length of the box, this indicates a deviation from normality. One should be careful in using numerical techniques that assume normality of the underlying distribution.

Finally, a more traditional way of portraying the distribution is the *histogram*. The domain of possible values is divided into a number of intervals of equal length. The number of points in each interval is counted, and these counts are represented as bars with lengths proportional to the counts. Figure 3.3 shows an example of a histogram.

Figure 3.3. Histogram of the manure production



The choice of the number of intervals is rather critical. Too few intervals will cause many details to be hidden, and too many intervals will cause too much focus on the details. An often used rule of thumb is to take the number of interval approximately equal to the square root of the number of observations, with a minimum of 5 and a maximum of 20 intervals.

The histogram is mainly useful for studying the symmetry and concentration of the distribution. In some cases this graph might also be able to detect grouping of values. The histogram is not the proper instrument for detecting outliers.

There are also numerical ways to characterise the distribution, and to search for outliers. One of the most frequently used techniques is based on the mean and variance of the observations. Suppose we have N values X_1, X_2, ..., X_N. Then the *mean* is defined by

$$\overline{X} = \frac{1}{N}\sum_{i=1}^{N} X_i$$

and the *variance* is equal to

$$S^2 = \frac{1}{N-1}\sum_{i=1}^{N}(X_i - \overline{X})^2 .$$

The *standard deviation* S is defined as the square root of the variance. If the underlying distribution is normal then approximately 95% of the values must lie in the interval

$$(\overline{X} - 2S; \overline{X} + 2S),$$

and approximately 97% in the interval

$$(\overline{X} - 3S; \overline{X} + 3S).$$

Outliers can now be defined as values outside one of these intervals. This simple technique has two important drawbacks:

1) The assumptions are only satisfied if the underlying distribution is approximately normal. So, this technique should only be used if a graphical technique has shown that this model assumption is not unrealistic.

2) The technique is very sensitive to outliers. A single outlier can have a large effect on the values of mean and variance, and therefore even hide the outlier. Also here the message is that the technique should only be used after graphical techniques have shown no dramatical deviations from normality.

Less sensitive techniques are based on the median and the quartiles of the distribution. For example the box-and-whisker plot could be applied in a numerical way. Values smaller than the lower adjacent value or larger than the upper adjacent value would then be marked as outliers.

4. Macro-editing techniques for two variables

Now we discuss some techniques to study the simultaneous distribution of two quantitative variables. Also here we stress the importance of first taking a look at plots before relying on numerical techniques.

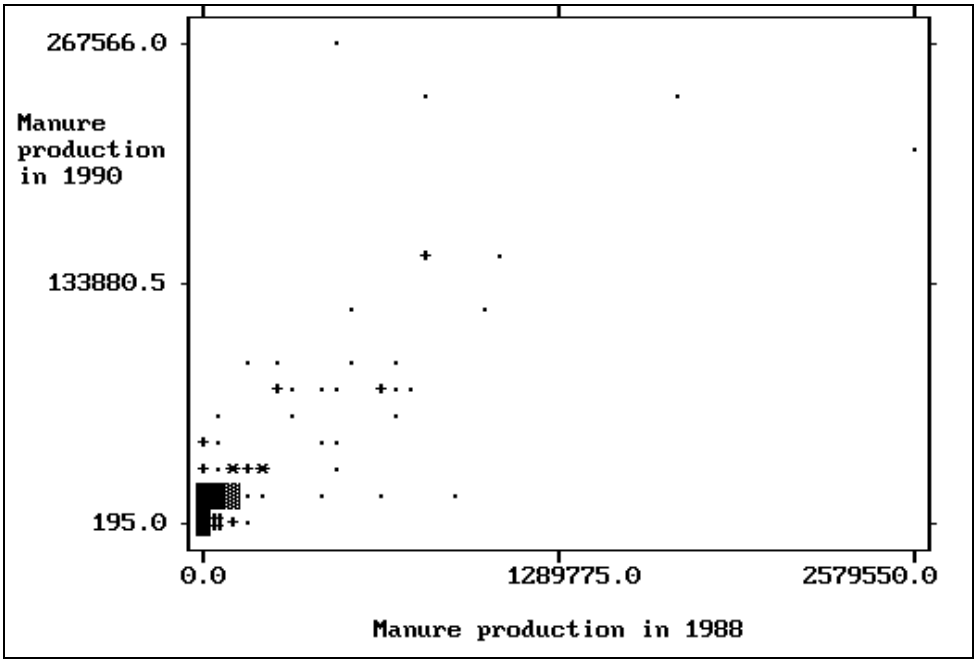
The obvious technique for displaying the relationship between two variables is the *two-dimensional scatterplot*. Suppose, we have two variables X and Y. For all N cases in the sample we have the pairs of values (X_1,Y_1), (X_2,Y_2), ..., (X_N,Y_N). The scatterplot displays each pair (X_i,Y_i) as a point with co-ordinates X_i and Y_i. So, the scatterplot will contain N points, each representing a case. Here are some patterns to look for in a scatterplot:

- *Outliers*. Outliers appear as single points that are far apart from the rest of the observations. Outliers require further analysis, since they may indicate wrong values;

- *Grouping*. Grouping means that points are not more or less evenly spread over the whole domain of possible answers, but instead appear to be concentrated in separate groups. Grouping may be an indication the observations originate from differently behaving subpopulations, and therefore it might be more appropriate to analyse these groups separately;

If clear patterns and structures can be distinguished in the plot, this indicates a certain relationship. The simples form of a relationship is that of a linear relationship. In this case, all points will (approximately) lie on a straight line. If such a relationship seems to hold, it is important to look for points not following the pattern. These point may indicate errors in the data. Figure 4.1 shows an example of a two-dimensional scatterplot.

Figure 4.1. Example of a two-dimensional scatterplot



The variable on the X-axis is the yearly manure production in 1988 by farms in the municipalities in the Dutch province of Zuid-Holland. The variable on the Y-axis is the same variable, but measured in the year 1990. The plot seems to indicate a linear relationship between the two variables, but there are also some points that do not conform to this pattern.

If the assumption of linearity is not unreasonable, it can be summarised in the form of a regression line. The formula for this regression line is

$$Y = a + bX$$

where a and b are coefficients that have to be computed using the available data. For the best line, i.e. the line that follows the pattern most closely, the value of b is equal to

$$b = \frac{\sum_{i=1}^{N}(X_i - \overline{X})(Y_i - \overline{Y})}{\sum_{i=1}^{N}(X_i - \overline{X})^2},$$

and a is equal to

$$a = \overline{Y} - b\overline{X}$$

If the assumption of a linear relationship is correct, the regression line can be used to detect outliers. First, define the *residual* R_i for case i by

$$R_i = Y_i - a - bX_i.$$

The residual is equal to the difference between the value Y_i and the prediction $a-bX_i$ for Y_i based on the regression line. It is the (vertical) distance between the point and the regression. Large residuals indicate outliers. It can be shown that the *studentized residual* T_i, defined by

$$T_i = \frac{R_i}{S(T_i)},$$

where $S(T_i)$ is equal to

$$S(T_i) = \sqrt{1 - \frac{1}{n} - \frac{(\overline{X} - X_i)^2}{\sum_{k=1}^{N}(X_k - \overline{X})^2}},$$

has an approximately normal distribution. Therefore, studentized residuals with values outside the interval (-2;2) or (-3;3) can be considered outliers. These cases need further investigation.

The computation of the coefficients a and b of the regression line is sensitive to outliers. A few big outliers can have a considerable effect on the shape of the regression line. So, be careful to look at the scatterplot before computing regression lines.

There are ways to compute regression lines that are less sensitive to outliers. One technique is iteratively reweighted regression. A weighted regression is repeated a number of times. In each regression, weights are assigned to cases. These weights are determined by the residuals of the previous regression. The larger the residuals, the smaller the weights. For more details, see e.g. Chambers et al. (1983).

5. Simple macro-editing with Blaise and Manipula

There are several ways to implement macro-editing in Blaise. In this paper we want to present two such approaches. The first approach is described in this section. It is a simple approach that does not take full

advantage of the graphical possibilities for macro-editing. This approach can be implement using available tools like Blaise and Manipula. The second approach requires an add-on to the Blaise system, but provides a more effective environment for macro-editing. This approach is described in the next section.

We start with macro-editing a single variable, and concentrate on the detection and treatment of outliers. To be able to do that a number of quantities must be computed, and inserted in the data model. In this approach, we go through the following steps:

1) Enter data with the DEP, using CAPI, CATI, or CASI, depending on the survey at hand. Reserve fields in the data model to hold aggregate statistics required for macro-editing;

2) After all data has been collected, use Manipula to compute aggregate statistics, import these statistics in the Blaise data file, and carry out a check on every record;

3) Edit the rejected forms with the DEP in CADI mode.

The traditional approach to finding outliers is based on the prediction interval described in section 3. To compute this interval we need the mean and the standard deviation.

We will describe an example of this approach using an example based on artificial data. It relates to manure production by farms in municipalities in the Dutch province of Zuid-Holland. For two years (1988 and 1990), data has been collected on three variables: area of grass, number of cattle, and manure production. Figure 5.1 contains the Blaise metadata specification for these data.

The data model contains three blocks: one block with information to identify the municipality, one block with the agricultural variables, and one block dealing with macro-editing. The second block is used twice, once for the year 1988, and once for the year 1990. Note that this model is used for reasons of simplicity. It is not a realistic data model. In a more practical situation the data on 1988 would probably be imported, and the data on 1990 would be asked.

Note that this model can be used in two ways. In data entry mode, the DEP is used as a program for computer-assisted interviewing. The macro-editing block is skipped. In data editing mode, the checks in the macro-editing block are carried out.

Once all data has been collected in data editing mode, the aggregate statistics must be computed, imported in the Blaise data file. Then the forms have to be checked. Figure 5.2 contains a Manipula setup that takes care of these activities. The target variable for this macro-editing operation is the manure production in 1990.

The Blaise data file is an update file. This means that in the same Manipula session information is read from and written to the file. The setting *AUTOREAD = NO* implies that Manipula does not automatically reads records from the data file, but only when it is instructed to do so.

The description of the Blaise data file contains the setting *CHECKRULES*. The consequence of this setting is that every processed record is checked. The checks carried out are those defined in the rules section of the Blaise data model *Manure1*.

```
DATAMODEL ManureProduction

TYPE
   TReal    = REAL[12, 3]
   TInteger = 0..99999999

BLOCK TMunicipality
   FIELDS
      MunCode "Municipality code": 0..999
      MunName "Municipality name": STRING[26]
   RULES
      MunCode MunName
ENDBLOCK

BLOCK TData
   PARAMETERS
      Year: INTEGER
   FIELDS
      Grass   "Area of grassland in ^Year": TInteger
      Cattle  "Number of cattle in ^Year" : TInteger
      Manure  "Manure production in ^Year": TInteger
   RULES
      Grass Cattle Manure
ENDBLOCK

BLOCK TMacroEdit
   PARAMETERS
      FieldValue: INTEGER
      FieldName : STRING
   LOCALS
      LBound, UBound: Real
   FIELDS
      Mean  "Mean of ^FieldName"              : TReal
      StDev "Standard deviation of ^FieldName": TReal
   RULES
      Mean.SHOW
      StDev.SHOW
      LBound:= ROUND(Mean - 2 * StDev)
      UBound:= ROUND(Mean + 2 * StDev)
      (FieldValue > LBound) and (FieldValue < UBound)
      "Value of ^FieldValue outside the interval (^LBound ; ^UBound)!"
ENDBLOCK

FIELDS
   Municipality: TMunicipality
   ThisYear    : TData
   LastYear    : TData
```

Figure 5.1. The first Blaise metadata specification

The auxfields section of the Manipula setup contains a block definition. The definition of this block is similar to that in the Blaise data model. Such use of similar block makes it easier to specify manipulations. For example, the manipulate section contains the assignment *UpFile.MacroEdit:= MEdit.* This causes the values of all variables in the block *MEdit* to be copied to the corresponding fields in the block *MacroEdit* of the Blaise data model.

Figure 5.2. Manipula setup for the mean and standard deviation

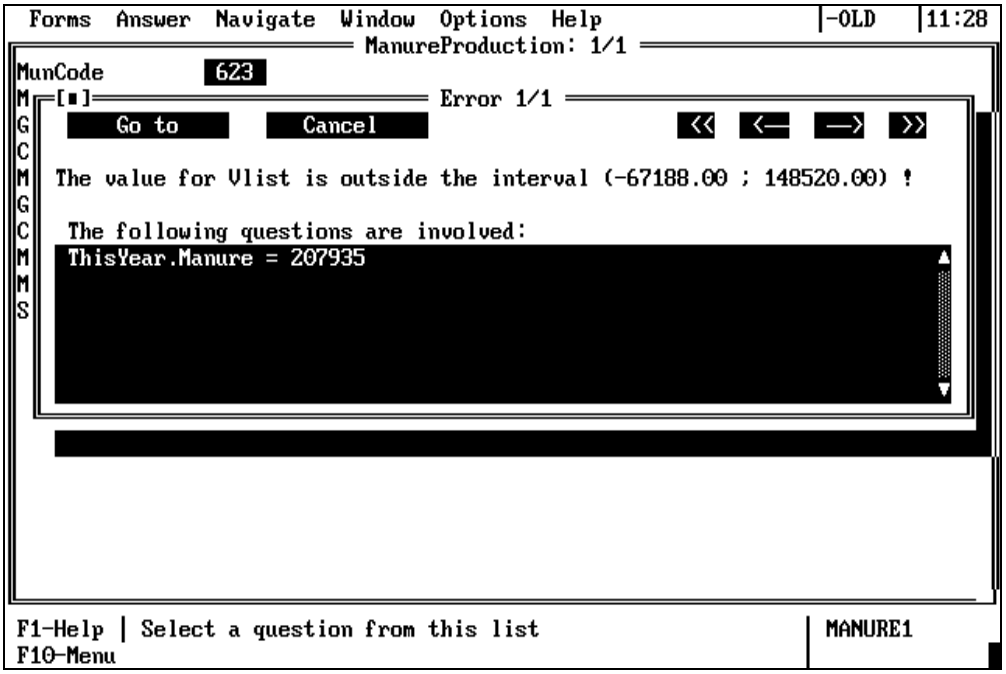
```
SETTINGS
   AUTOREAD = NO

USES
   BlaiseData 'Manure1'

UPDATEFILE
   UpFile: BlaiseData ('Manure1', BLAISE3)
SETTINGS
   CHECKRULES = YES

AUXFIELDS
   TYPE
     TReal = REAL[12, 3]

   BLOCK TMacroEdit
      FIELDS
         Mean : TReal
         StDev: TReal
   ENDBLOCK

FIELDS
   MEdit  : TMacroEdit
   Sx, Sxx: REAL
   RecNum : INTEGER

MANIPULATE
   FOR RecNum:= 1 TO UpFile.RECORDCOUNT DO
       InFile.READNEXT
       Sx := Sx  + ThisYear.Manure
       Sxx:= Sxx + SQR(ThisYear.Manure)
   ENDDO

   Sx:= Sx / MEdit.UpFile.RECORDCOUNT
   MEdit.Mean:= Sx
   MEdit.StDev:= SQRT((Sxx - UpFile.RECORDCOUNT* SQR(Sx)) /
                 UpFile.RECORDCOUNT)

   UpFile.RESET
   FOR RecNum:= 1 TO UpFile.RECORDCOUNT DO
       UpFile.READNEXT
       UpFile.MacroEdit:= MEdit
       UpFile.WRITE
   ENDDO

   READY
```

The manipulate section contains three series of statements. The first series reads the Blaise data file, and computes the sum and the sums of the squares of the field *Manure*. Note the use of the statement *UpFile.READNEXT* to instruct Manipula to read the next record. The second series of statements computes the mean and standard deviation, and assigns these values the corresponding variables in the block *MEdit*. The third series of statements resets the Blaise data file, and changes each record by reading it, copying the contents of the block MEdit to the block MacroEdit, and writing the record back to the file.

Now all information is available to carry out a macro-editing operation. By activating the Data Viewer from the Blaise Control Centre, we can take a look at the data. Note that is possible in the current version of Blaise to see the values of the status variable. This means that we can see whether forms are clean, suspect, or wrong.

We activate the DEP and run the program in in data editing mode (through the option Data entry mode in the Options menu). All records with a value of *ThisYear.Manure* outside the interval *(Mean ± 2 StDev)* have an error attached to the field *Manure*. Figure 5.4 shows an example of an error message.

Figure 5.3. Example of an error message



Apparently, the manure production in the Municipality of Vlist is so high that is marked as an outlier. Note that the interval has a negative lowerbound, although all manure values are greater than or equal to zero. This rather strange lowerbound is caused by the fact that mean and standard deviation are used in a situation where the underlying distribution does not resemble the normal one. In fact, the distribution is very skew, with a lot of small values, and a few large values (see figure 3.1). In this case it would have been better to carry out a transformation on the distribution before
attempting to detect outliers.

Note that mean and variance of the variable are computed in the Manipula setup, and the bounds of the interval in the data model. Of course, the bounds could also have been computed in the Manipula setup,

but then the values of mean and variance would not have been available (for other purposes) in the data model.

A similar approach could have been used to detect outliers based on the more robust approach of the adjacent values in the box-and-whisker plot. Of course, that would need a different version of the Manipula setup and the macro-editing block in the data model.

We will now show how the analysis of two variables can be implemented using Blaise and Manipula. We want to compare the manure production in 1990 with that of 1988, and studentized residuals outside the interval (-2 ; 2) must be flagged as outliers.

Figure 5.4 contains a part of the Blaise metadata specification for this example. The only difference with the Blaise specification of figure 5.1 is the block with the macro-editing specification. Figure 5.4 contains that block for our two-variable example.

Figure 5.4. The macro-editing block for the two-variable analysis

```
BLOCK TMacroEdit
   PARAMETERS
      XFieldValue: INTEGER
      XFieldName : STRING
      YFieldValue: INTEGER
      YFieldName : STRING

   FIELDS
      N     "Number of records"                : 0..999
      Mean  "Mean of ^XFieldName"              : TReal
      StDev "Standard deviation of ^XFieldName": TReal
      A     "Regression coefficient A"         : TReal
      B     "Regression coefficient B"         : TReal
      RDev  "Standard error of residuals"      : TReal

   AUXFIELDS
      StudRes "Studentized residual": TReal

   RULES
      A.SHOW
      B.SHOW
      StudRes:= (YFieldValue - A - B * XFieldValue) /
      SQRT(1.0 - 1/N - SQR(Mean - XFieldValue) / (SQR(StDev) * N)) / RDev

      (StudRes > -2) AND (StudRes < 2) INVOLVING(YFieldValue)
      "The studentized residual for ^YFieldValue is ^StudRes.
      @/This value is outside the interval (-2 ; 2) !"

ENDBLOCK
```

The manure production in 1990 will play the role of the Y-variable inour example, and the manure production in 1988 will be the X-variable. Given the regression statistics, the rules section of the block *TMacroEdit* computes the studentized residual.

After all data has been collected, the regression information can be computed with Manipula. Figure 5.5 contains a Manipula setup for this purpose.

The setup has the same structure as the one for the one-variables case. The Blaise data file is an update file. Manipula does not automatically read records from the data file, but only when it is instructed to do so. The setting *CHECKRULES* takes care of checking the records after they have been changed. The checks carried out are defined in the rules section of the Blaise data model *Manure2*.

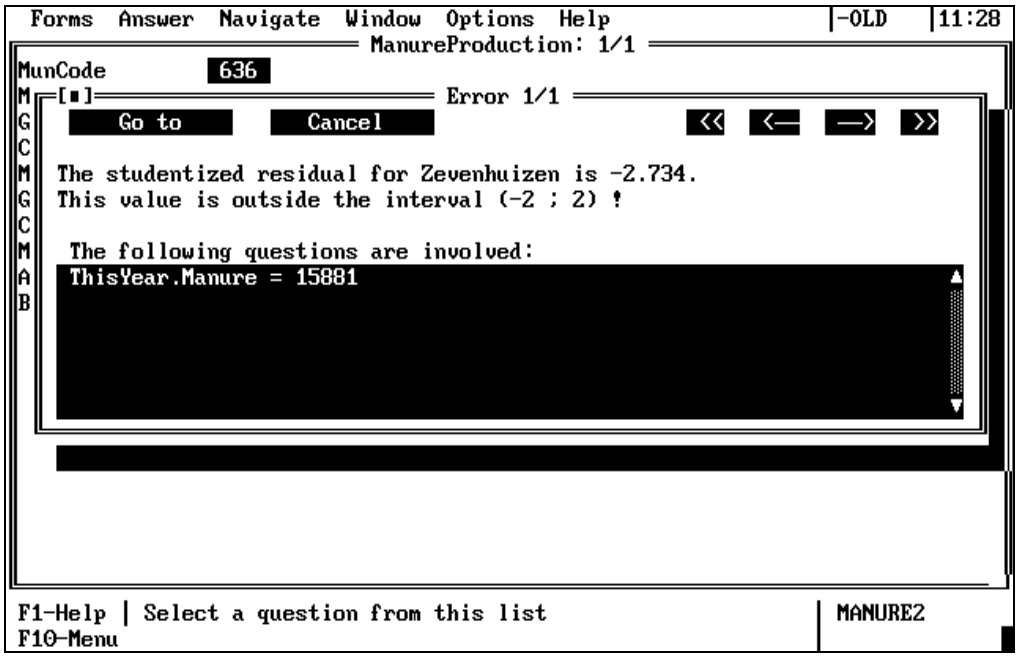
Figure 5.5. Manipula setup for computing regression quantities.

```
SETTINGS
   AUTOREAD = NO

USES
   BlaiseData 'Manure2'

UPDATEFILE
   UpFile: BlaiseData ('Manure2', BLAISE3)
SETTINGS
   CHECKRULES = YES

AUXFIELDS
   TYPE
      TReal = REAL[12, 3]

   BLOCK TMacroEdit
      FIELDS
         N    : 0..999
         Mean : TReal
         StDev: TReal
         A    : TReal
         B    : TReal
         RDev : TReal
   ENDBLOCK

FIELDS
   MEdit: TMacroEdit
   Sx, Sy, Sxx, Syy, Sxy: REAL
   RecNum: INTEGER

MANIPULATE
   FOR RecNum:= 1 TO UpFile.RECORDCOUNT DO
       UpFile.READNEXT
       Sy := Sy  + ThisYear.Manure
       Sx := Sx  + LastYear.Manure
       Sxx:= Sxx + SQR(LastYear.Manure)
       Syy:= Syy + SQR(ThisYear.Manure)
       Sxy:= Sxy + LastYear.Manure * ThisYear.Manure;
   ENDDO

   MEdit.N:= UpFile.RECORDCOUNT
   Sx:= Sx / MEdit.N
   Sy:= Sy / MEdit.N
   MEdit.B:= (Sxy - MEdit.N * Sx * Sy) / (Sxx - MEdit.N * SQR(Sx))
   MEdit.A:= Sy - MEdit.B * Sx
   MEdit.Mean:= Sx
   MEdit.StDev:= SQRT((Sxx - MEdit.N * SQR(Sx)) / MEdit.N)
```

The definition of the block *MEdit* in the setup is similar to the one in the Blaise data model. Again, the manipulate section contains three series of statements. The first series reads the Blaise data file, and computes some variables required for calculation of the regression quantities. The second series calculates the regression quantities, and assigns these values the corresponding variables in the block *MEdit*. The third series of statements resets the Blaise data file, and changes each record by reading it, copying the contents of the block *MEdit* to the block *MacroEdit*, and writing the record back to the file.

After the Manipula session has been successfully completed, all information is available to carry out a macro-editing operation. By setting the *ProcessMode* variable to the value *DataEditing*, and re-preparing the data model, the DEP can be run in data editing mode. All records with a studentized residual outside the interval (-2 ; 2) have an error attached to the field *Manure*. Figure 5.6 shows an example of an error message.

Figure 5.6. An example of a DEP error message



The manure production of the Municipality of Zevenhuizen in the year 1990 is 15881. The studentized residual of this value is equal to -2.734. Since this value is less than -2, its flagged as an outlier.

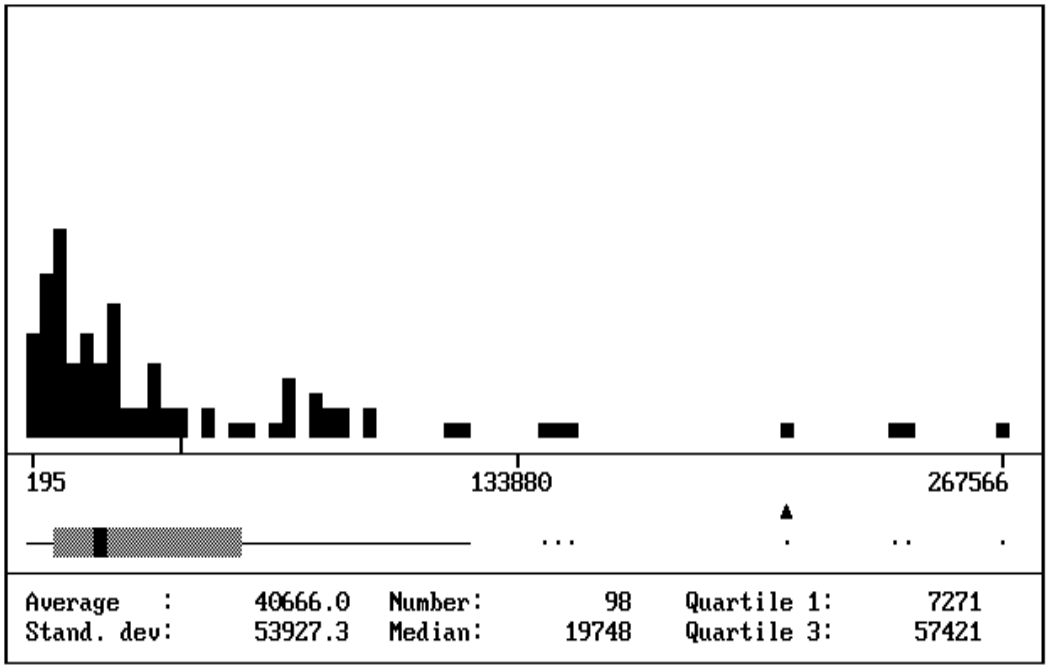
6. A front-end for macro-editing

The macro-editing approach in the previous sections has it limitations. One of the most important one is the lack of graphical facilities. In this section it is shown that it is not too difficult to build a macro-editing tool with more possibilities.

The basic idea is the following. Use the Data Selector of the Blaise Control Centre to select the variables that are to be included in the macro-edit analysis. Extract all values of the selected variables from the Blaise data file, and import these values in the macro-editing front-end tool. This tool displays all values graphically. The operator can selected a point in the graph. This causes the DEP to be activated for the corresponding record. Then the operator can change values in the fields of the record. After that, the point in the graph is adjusted accordingly.

The prototype of the macro-editing front-end can carry out both a one-variable and a two-variable analysis. First, the one-variable analysis is described. After having selected the analysis variable, a graphical overview appears on the screen containing all necessary information about the distribution. Figure 6.1 contains an example. The screen contains information about the manure production in 1990.

Figure 6.1. Analysis of the variable Manure in 1990



The top part of the screen contains the one-way scatterplot. It is the same scatterplot as displayed in figure 3.1. It is clear that the distribution of the manure production is very skew. There are many municipalities with a small manure production, and only a few municipalities with a large manure production. It might be better to carry out a transformation on this variable to transform its distribution into a normal one. However, this feature is not available in this prototype.

A box-and-whisker plot is displayed just below the scatterplot. It is a simplified version of the one in figure 3.2. The same scale is used. According to this plot, there are seven outliers. Part of these outliers may be due to the skewness of the distribution, but it may be worth while to look at the four extreme outliers.

Between the scatterplot and the box-and-whisker plot you can move a triangle-shaped cursor to the left and to the right (with the cursor-keys). By positioning the cursor under a point of interest, you can open the corresponding form. Since several forms may share the same location on the X-axis, first a list will appear containing field values identifying the forms. In this list, the form to be edited can be selected. The DEP will be activated for this form, and you change the value of the relevant variable. After the form has been closed, the screen with the scatterplot and the box-and-whisker plot re-appears. Note that as consequence of changing the value of the analysis variable, the square representing the form may have moved to a different location.

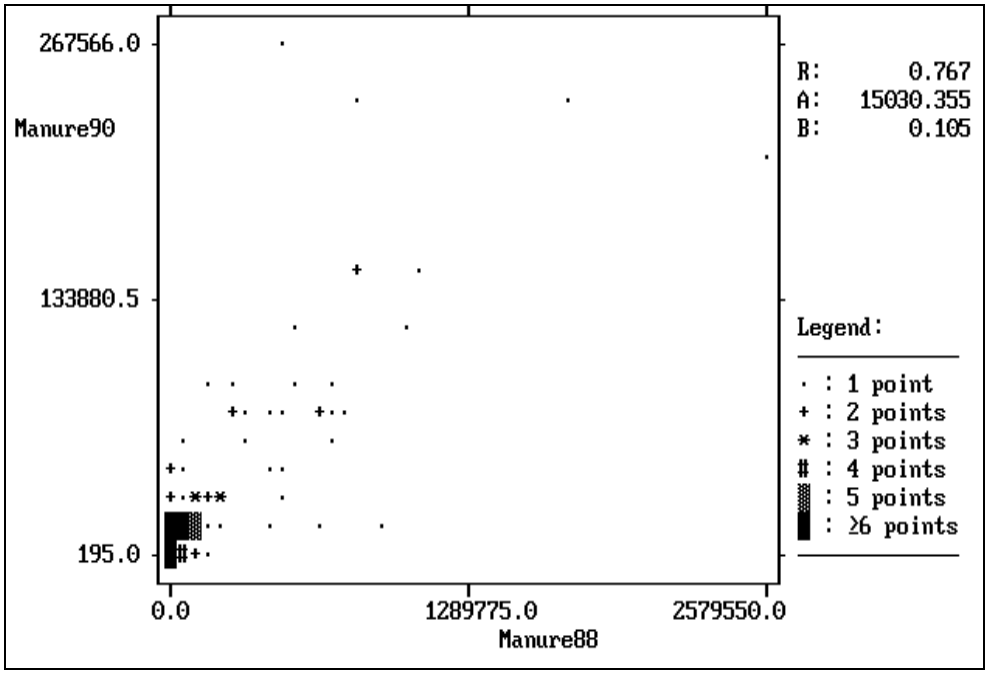
The aggregate information, like the box-and-whisker plot, and the numerical quantities at the bottom of the screen, are not effected by changes in individual values of the variables. They are only re-computed, if the program is instructed to do so. Note that re-computation of outliers after each individual change may lead to different results then waiting with re-computation after all original outliers have been taken care of.

For the two-way analysis, the prototype front-end produces a screen like the one in figure 6.2. It contains a two-way scatterplot of the variables involved. Note that overlapping points are indicated by special symbols (see the legend). The values of the correlation coefficient (R), and two regression coefficients (A and B) are displayed to the right of the plot.

A square cursor can be moved through the scatterplot (with the cursor keys). By positioning the cursor on a point of interest, you can open the corresponding form. Since several forms may share one location in the plot (due the same rounded X- and Y-co-ordinates), first a list will appear containing field values identifying the forms. In this list, the form to be edited can be selected. The DEP will be activated for this form, and you can change the value of the relevant variable(s). After the form has been closed, the screen with scatterplot and the box-and-whisker plot re-appears. Note that as consequence of changing the value of the variables involved, the dot representing the form may have moved to a different location.

Also here, the aggregate information (R, A and B) can be recomputed, but only if the program is instructed to do so. It is important to realise that re-computation of outliers after each individual change may lead to different results then re-computation after all original outliers has been taken care of.

Figure 6.2. Two-way analysis of the variable Manure in 1989 and 1990



7. Conclusion

Macro-editing can be interesting alternative at a time where resources to be spend on data editing must be minimised, at the same time maintaining a high level of data quality. The newest version of Blaise has some interesting possibilities for macro-editing, be it that they are somewhat limited due to the lack of graphical facilities. The paper also shows that a front-end can be implement with useful graphical macro-editing possibilities.

References

Chambers, J.M., Cleveland, W.S., Kleiner, B., and Tukey, P.A. (1983). *Graphical Methods for Data Analysis.* Duxburry Press, Boston, USA.

Granquist, L. (1990). *A Review of some Macro-editing Methods for Rationalizing the Editing Process.* Proceedings of the Statistics Canada Symposium 90, pp. 225-234.

Tukey, J.W. (1977). *Exploratory Data Analysis.*

Computer Assisted Occupation Coding

Diane Bushnell, Office of Population Censuses and Surveys, UK

1. Introduction

Many papers have been written describing the advantages of using computer assisted personal interviewing (CAPI) compared to paper and pencil interviewing (PAPI). However, the potential of CAPI has not yet been exploited fully for the process of coding open-ended responses. Traditional methods are still widely used, i.e. the response is entered verbatim into the computer and coded manually later, either by the interviewer at home or by specialised coders back in the office.

In recent years, computer assisted coding has become more sophisticated, allowing the interviewer to search through a large database of responses and select an appropriate code during the interview. The usual benefits of computer assisted interviewing apply. The elimination of a separate data entry stage results in lower costs and a quicker turn-around of interview data, particularly where coding was previously carried out at HQ; data quality is improved due to a reduction in data entry and consistency errors and the data collected is more likely to be an accurate reflection of the interview as the data entry and editing stages are carried out in the presence of the data source (i.e. the respondent).

In Social Survey Division (SSD) of OPCS, we use computer assisted coding (CAC) in the interview for straightforward coding frames, such as country of birth and nationality, and most diary components of surveys are now coded by interviewers using CAC either at home or in the SSD Telephone Unit, e.g. items bought in last week, journey destinations. However, the coding of occupation is still carried out using the traditional methods.

We have been evaluating three systems for computer assisted occupation coding against a set of ideals for a model system. This paper describes our progress so far and a few preliminary findings, in terms of the technical abilities of the systems and their impact on data quality.

2. Occupation coding

In the United Kingdom, the Standard Occupational Classification (SOC) (OPCS, 1990) is the most widely used occupational coding scheme for social research. The SOC was developed for use on the 1991 Census of Population by OPCS, the Department of Employment and the Institute for Employment Research at the University of Warwick.

The classification consists of 371 occupational unit groups (SOC codes) which may be aggregated into minor and major groups. The system is hierarchical so, for example, unit group 270 (Librarians) is in minor group 27 (Librarians and Related Professionals), which is in major group 2 (Professional Occupations).

The major use of SOC in social research is for assigning individuals to various social class and socio-economic classifications.

In SSD the interviewers collect verbatim details on the job title, main duties and responsibilities of the job, qualifications required and industry, as well as asking specific questions about employment status (e.g. whether self-employed and the size of establishment). At home the interviewers go back into the questionnaire, review the details and select the SOC code which they think is most appropriate from a paper coding index. They then type the code into the Blaise instrument.

In theory, the allocation of a code to an occupation is quite straightforward: the interviewer simply looks up the job title in the SOC coding index. In practice, the procedure is much more complex. The index was published in 1990 (to tie in with the 1991 Census of Population) and contains around 21,500 entries. Even within five years many changes can occur in occupations: job titles become obsolete, new titles appear and job titles do not necessarily reflect the occupation of the job holder. Fashions may dictate that the same type of work is given a different title over time, e.g. Personnel Officers become Human Resources Managers, or whole new groups of workers are labelled with a title that had a very specific membership previously, e.g. a sudden increase in the number of able seamen was found to be due to a trend for staff working in MacDonalds to be titled 'Crew members'.

As well as problems with the index, people tend to vary the information they supply about their jobs according to the situation, so the same job could be described in many different ways. People may describe what they have been working on most recently, or may feel differently about their job at different times or may tailor the description according to the audience, e.g. I could describe myself as a civil servant, Senior Executive Officer, Senior Social Survey Officer, Statistician, Social Researcher etc.

Thus, interviewers (or coders) will need to make decisions based on all the information supplied, not just job title, in order to assign the relevant code. As with any process requiring human judgement this will give rise to variation over time and between coders. In the past, specialised coders were trained to apply standard rules rigorously; supervisors resolved queries and quality checks took place regularly. It is clearly more difficult to provide this amount of guidance on coding for face-to-face interviewers. Studies in SSD comparing field coding of occupation with office coding (Dodd 1985, Martin et al 1995) have shown that reliability is lower when coding is carried out by interviewers. In the most recent study, 8 interviewers, 5 office coders and 1 expert in occupation coding assigned SOC codes to 200 occupations and the reliability and accuracy of coding were examined. In this context, reliability is defined as the extent to which different coders assign the same code to the same case, and accuracy defined as the amount of agreement between the coders in the trial and the 'expert'. The average reliability of office coders was 0.82 and of field coders (interviewers) was 0.74 (a figure of 1 indicates perfect agreement on all cases). Accuracy was 0.80 for office coders and 0.77 for interviewers.

Although accuracy is not too badly affected by field coding the consistency of coding obviously suffers. On the plus side, the impact of individual coder bias, i.e. a systematic deviation in assigning

codes compared to the other coders, is reduced by interviewer coding since the individual workloads are much smaller and spread over many more coders (the whole interviewing force compared to a handful of office coders).

In an attempt to address the difficulties with occupation coding we decided to investigate the use of a computer-aided coding system. In addition to the usual benefits of CAPI mentioned earlier, we hypothesised that computer assisted occupation coding would improve reliability by applying the standard coding rules more consistently than manual coding; accuracy would be increased by reducing the number of inappropriate codes presented to the coder; some of the burden on the interviewer would be removed as the task would be easier and the effect of coder bias would be reduced. It is important to note that although coder bias may decrease, the system itself may favour particular codes so a 'coding system' bias may be introduced.

In SSD, we began our investigations by considering three different coding strategies: using existing Blaise coding facilities, incorporating an external coding application into a Blaise questionnaire and using standalone occupation coding software.

Using Blaise was the obvious starting place as all our CAPI surveys are conducted using Blaise and we are already using the coding facilities for other coding frames. Tests were carried out with both Blaise 2.5 and Blaise III and the results are described in Section 3.2 below. The alternative strategies both involved using a specialised occupation coding application. In the UK, the most sophisticated software of this type is CASOC (Computer Assisted Standard Occupational Coding). The University of Warwick, with the University of Cambridge, began development of CASOC in 1986. CASOC is based on the OPCS Standard Occupational Classification and uses the same standard coding rules as OPCS. To investigate the second approach to coding occupation, using an external application from within Blaise, OPCS commissioned one of the authors of CASOC to produce a cut-down version which could be called as a user unit from Blaise. Progress so far is outlined in Section 3.2. The third option, using a standalone occupation coding package, although possibly the quickest and easiest to implement, was not considered compatible with our ultimate goal of integrating the occupation coding with the interview so no further work has been carried out in this area.

3. Computer Assisted Occupation Coding

3.1 Evaluation criteria

As a means of evaluating the coding procedures we devised a set of characteristics for a model coding system. The basic requirements were that it should use the job title to search through a coding index, present various code options, store the code for future use and then continue with the interview. The 'ideal' system should include as many of the following characteristics as possible.

The system should:

a. Be easy to use

Interviewers should need a minimum of special instructions.

b. Display exact matches first

If an exact match for the entered job title is found in the coding index then this match should always be displayed at the top of the suggestion list.

c. Be at least as reliable and accurate as manual coding

As one of the main aims of introducing computer assisted occupation coding is to improve reliability and accuracy of occupation coding this criterion is a very important one. We know that one of the problems with computer assisted coding in general is that interviewers favour codes at the top of the list and may select the top code even when it is clearly inappropriate. Thus it is important that, where an exact match does not exist in the index, the agreement between the code presented at the top of the suggestion list and the 'best' code is high.

d. Be fast

The process of assigning a code should be at least as fast as manual coding using a paper index.

e. Be able to pick up information from the interview

The system should be able to pick up information which has already been collected during the interview. This is useful for two reasons: coding may not always take place at the exact moment that the information is collected, e.g. initially the system will be tested by interviewers coding at home so information from the interview will be passed to the coding system at a later stage of the questionnaire; several pieces of information may be added together and used for coding so these would need to be picked up after all the data have been collected.

f. Save the occupation code in the Blaise data file

The code should be saved with the rest of the Blaise interview data for ease of analysis and to prevent losing data, making errors, etc.

g. Have a simple mechanism for creating and updating the coding frame

Although the SOC coding frame does not change very often it is essential that minor amendments can be made easily. The process of constructing the coding frame should be fairly straightforward and need no special technical expertise.

h. Be able to restrict search by auxiliary fields

It would be useful if the codes suggested could be restricted to a subsection of the coding frame by filtering on information other than the job title. For example, if the respondent was self-employed, then only valid codes for self-employed occupations would be displayed initially (although it would be necessary to have the option to extend the search to show all codes).

i. Include the possibility of displaying supplementary information on the occupation

It would also be useful if information pertinent to particular codes could be displayed. For example, synonyms for job titles (gaffer, foreman, supervisor etc.) or a summary of typical tasks carried out by job holders in those occupations.

j. Allow interviewers to take advantage of the hierarchical structure of the coding frame

Job titles will be assigned a code and this code will have a general name, known as the unit group heading. This heading can be used as a check that the correct code has been assigned. It might also be useful for the interviewers to step through the hierarchy, as a check on accuracy and also to increase awareness of the coding frame structure.

k. Include the possibility of automated and/or semi-automated coding

Automated coding can be defined as a system which assigns a code to the job title without any interaction with the coder. Semi-automated coding, in this context, is defined as a system where coding is automatic if there is one match with a goodness-of-fit above a certain cut-off point but where the coder assists with coding if the fit is not good enough. Both types of coding would obviously reduce the burden on interviewers and increase reliability but bias and accuracy may suffer unless the system is very good.

l. Include the option of excluding selected words from the matching process

To improve the performance of the matching algorithms it would be useful to exclude some words from the matching process. For example, if the job title contains the word 'and' this will introduce superfluous information into the match.

m. Be cost effective

The cost of developing and using a computer assisted coding system is obviously one of the key factors in our final decision making process. However, it is also one of the most difficult criterion to evaluate and so far, the cost implications of the various systems have not be explicitly analysed.

3.2 Evaluation

Coding systems were set up using Blaise 2.5, Blaise III and a combination of Blaise 2.5 and CASOC. Each of the systems was evaluated against the model characteristics: the results are summarised in Table 1. The occupation data used for the study on coding reliability was also used to test some aspects of reliability and accuracy for the systems.

Table 1 Three coding systems evaluated against 'ideal' criteria

Ideal Criteria	**Blaise 2.5**	**Blaise III**	**Blaise/ CASOC combination**
a. Easy to use	Y	Y	Y
b. Exact matches at top of list	N	N	Y
c. Reliability and accuracy at least as good as for manual coding	N	?	Y
d. Quick	Y	N	Y
e. Ability to pick up information from the interview	Y	N	Y
f. Occupation code saved in Blaise data file	Y	Y	Y
g. Easy to construct and update frame	Y	N	na
h. Ability to restrict search by auxiliary fields	N	N	Y*
i. Possibility of displaying supplementary information	N	Y	N
j. Ability to view information on levels of hierarchy	Y	N	Y*
k. Possibility of automated and/or semi-automated coding	N	N	Y*
l. Possibility of excluding selected words from matching process	N	N	Y
m. Cost effective	?	?	?

Y Yes, system fulfils this criterion

N No, system does not fulfil criterion

* Not included in current specification for customised system but available in standalone version of CASOC

? Not tested yet

na Not applicable

Blaise 2.5

The three types of coding available in Blaise 2.5 (hierarchical, trigram and alphabetical) were used in combination in our test system. The occupational details were collected in the usual way. At the code question the coding window appeared in the bottom half of the screen. The question, displaying the occupational details required for coding, appeared in the top half of the screen. Initially, the top level of the SOC hierarchy was displayed in the coding window and the coder could step through the hierarchy or proceed by word searching. The job title entered previously was used for the search. Matches were displayed in the coding window and the coder could move up and down the list to select a code or could change the description to search for other matches. When a code was selected the unit group heading was displayed; the coder could accept this or continue searching until satisfied with the code selected. Finally, the code and description used for searching were saved in the data file and the coder returned to the Blaise interview.

The main advantages of this system were that it was easy to use and was familiar to the interviewers. The disadvantages were that only one field could be used in the coding process and no other information could be displayed to help the coder make the most appropriate decision, there was no automated or semi-automated coding and it was not possible to exclude words from the matching process. A significant problem was that the code at the top of the suggestion list was often not the most appropriate. Where an exact match to the job title exists in the coding index this was NOT always presented at the top of the list. Some further work is being done on changing the format of the index to reduce this problem. Reliability was not assessed but we did look at the extent to which the first code suggested agreed with that assigned by an expert coder (using the paper index). The data used for assessing the reliability and accuracy of field and office coders was copied into the coding system and trigram matches found for the 200 job titles. The code at the top of the search list was the same as the expert's code in 43.5% of cases (87 out of 200). Again, it may be possible to improve this figure by changing the format of the entries and removing punctuation etc, but as it stands, this figure is unacceptably low.

The principal costs for this coding system will be incurred in development of the coding frame and piloting.

Blaise III

At present, the coding module of Blaise III is still under development so all the following comments should be regarded in that light.

The basic coding functions of Blaise 2.5 are available in Blaise III. It is possible to carry out coding by means of a hierarchical, trigram or alphabetical search although at the time of testing it was not possible to integrate hierarchical with word match searching. A new feature is the ability to include more than one search field in the coding index. The coding scheme tested involved only trigram and alphabetic search procedures. It was not possible to pick up text previously entered so the job title was entered directly onto the coding screen. Trigram searching was set as the default with the

option to switch to alphabetic searching. After searching the database index entries with a 'good' match were displayed and the coder could move up and down the list and change the search description as for Blaise 2.5. When the code was selected it was written to the Blaise data file, along with the description held with the code in the coding index (not the description entered by the coder).

Not many of the ideal criteria were met by the Blaise III system (see Table 1 for a summary). The poor performance of this system was partly due to the incomplete development of the coding module in Blaise III and more of the criteria will be met as the program is refined.

The only real advantage of this system over the Blaise 2.5 system was that it was possible to have more than two fields in the coding index and any of these fields could be used for searching. For instance, industry could be used as a search field or it could just be displayed on the coding screen with the other information.

The process of creating the coding frame has changed significantly in this version of Blaise and, even allowing for learning time, took much longer than in Blaise 2.5. Manipula was used to convert the text coding index into a Blaise datamodel and 21,500 codes and descriptions took almost eight hours to convert. In most circumstances the coding frame would only need to be converted once so this should not be a critical constraint. However, it does mean that it may take a while to make relatively minor changes to the frame. The size of the subsequent datamodel and auxiliary files required for coding was about 12MB compared to under 2MB in Blaise 2.5. The time delay before trigram matches appear on the coding screen has also substantially increased so that there is a noticeable gap between typing the job title and the matches appearing on the screen. As the coding window fills most of the screen it obscures the occupational details displayed but presumably it will be possible to customise the location of the window in later versions. It should be possible for the Blaise team to fix all these problems with further work.

As in Blaise 2.5 there is still no facility for automated or semi-automated coding, nor the ability to restrict the search to subsets of the coding index. The building blocks for these functions seem to be in place already so it would not be too difficult to provide them, if there was enough demand by users.

As there are obviously some problems to be sorted out with this version of Blaise no tests of reliability or accuracy have been carried out.

As with the Blaise 2.5 option, the major expenditure will be on creating a good coding frame and on piloting the system.

CASOC in the Blaise instrument

It is not possible to call external programs from within Blaise III yet so we were only able to test this option using Blaise 2.5. A version of CASOC was written in Pascal and compiled with the Blaise instrument as a user defined question type. At present the system is still under development so it is only possible here to describe the specification for the system. The occupation details will be collected during the interview as usual. At the coding question the customised version of CASOC will take over and use the job title to search through its database. The appropriate codes and descriptions will be presented on the screen and the coder will be able to move around as if using Blaise. In fact, we have decided to make the interface with CASOC as similar to using Blaise 2.5 coding as possible. This means that the interviewers will be able to use the system with a minimum of training above that normally required for Blaise 2.5. When the code has been selected it will be saved in the Blaise data file and the interview will continue as normal.

As well as having the benefit of looking and behaving like Blaise, the system will have all the advantages of CASOC. The CASOC coding system employs a weighting system to arrive at the 'most likely' code as well as a word matching mechanism. Where job titles are similar the most common occupation will be first, e.g. Secretary/typist will be listed before Secretary of State. In the standalone version of CASOC it is possible to incorporate extra information into the search, for example, employment status or industry, and to move around the hierarchy of the coding frame, but these facilities have not been included in the specification for the current customised system. The frame itself is prepared by the authors of CASOC so that changes will be carried out automatically when appropriate. Potential disadvantages of the system are the costs of development and of using CASOC in tandem with Blaise, also the lack of control inherent in any system which is not developed in-house.

Due to the incomplete status of this project there are obviously no figures for the reliability or accuracy of this method of coding yet although some tests have been carried out using standalone CASOC. Campanelli et al carried out two separate experiments where occupations were coded using CASOC. In the first, five coders experienced in coding occupation on the Census assigned occupation codes to 401 cases. Three occupation experts coded the same information and their results compared to the professional coders' as a means of testing the accuracy of coding. Average reliability for the coders was 0.78 and agreement with the expert occupation coders ranged between 0.69 and 0.79.

In the second study, coders assigned codes to 322 occupation titles. Coding was carried out using the traditional technique first (consulting the paper index) and after a suitable interval, the process was repeated using CASOC. The occupation experts from the first study also coded information in this second study. There was little difference in reliability for the two methods (around 0.80) and agreement between the coders and experts was slightly higher when using CASOC than using the paper-based index (accuracy ranged from 0.78 to 0.85).

The accuracy figures for CASOC found from these studies may not be as high as they appear as two of the experts used for assessing accuracy were the authors of the CASOC system. Thus, one might expect good agreement between CASOC and its creators.

The reliability statistics are lower than those achieved by office coders using manual coding in the study conducted by SSD (see Section 2) (0.82) but higher than those for interviewers (0.74). The accuracy figures seem similar for all studies.

From these results, we might infer that using CASOC for occupation coding will improve reliability, compared to manual coding by interviewers, and that accuracy of coding will not be reduced.

To test how good the first suggested code was when using CASOC, the 200 cases from our reliability study were coded in fully automated mode. 69% of the cases (138 out of 200) were the same as those assigned by the expert coder. This is much better than the equivalent test carried out using Blaise 2.5 where only 43.5% of the codes listed first agreed with the expert coder's.

4. Summary

From the investigations into computer assisted occupation coding in SSD to date, a system which combines Blaise and CASOC seems to offer the most promising way forward.

The system using Blaise III met very few of the criteria we had devised for a model coding system but as Blaise develops some of the problems should be resolved. Blaise 2.5 fared better and was an attractive option in its ease of use, requiring little special training for interviewers. The main drawback of this system was the lack of a suitable coding frame. It was clear that a straightforward conversion of the paper coding index to computer was not acceptable: although it was quicker to look up the code electronically than using the paper index, there was a greater chance that the codes presented to the interviewer would be totally inappropriate. This would result in a decrease in accuracy and possibly an increase in bias (as some codes will appear more frequently than others).

Combining a specialised coding application (CASOC) with Blaise met many of the essential criteria for a good system as they were deliberately built into the specification or existed in the chosen software already. However, no conclusions can be drawn yet about the method as it is still under development. Apart from the possibility that this solution will not work at all, the main disadvantage is that it is only available under Blaise 2.5 at the moment. We hope that Blaise III will allow the inclusion of user functions and procedures soon. Studies have found that reliability and accuracy of occupation coding are as good or slightly better for coders using CASOC compared to traditional manual coding techniques. We will be carrying out our own tests when the combined Blaise 2.5/CASOC system is ready.

Over the next year, we intend to pursue the use of the combined system and to keep a close eye on developments in Blaise III. We will also be carrying out some qualitative work with interviewers on the process of coding occupation with the intention of tailoring our procedures to improve the quality of coding.

References

Campanelli P C, Thomson K, Moon N, Staples T (1995, forthcoming) *The Quality of Occupational Coding in the UK* In Survey Measurement and Process Quality, ed. L Lyberg, P Biemer, M Collins, E DeLeeuw, C Dippo, N Schwarz, D Trewin

Dodd T (1985) *An Assessment of the Efficiency of the Coding of Occupation and Industry by Interviewers* New Methodology Series No. NM14, London, OPCS

Martin J, Bushnell D, Campanelli P and Thomas R (1995) *A Comparison of Interviewer and Office Coding of Occupations* Joint Proceedings of ASA/AAPOR: Section of Survey Methods Research, American Statistical Association, Washington, DC.

The CAI system of Statistics Norway

Hilde Degerdal, Thomas Hoel and Truls Thirud, Statistics Norway

1. Introduction

After nearly two years of planning and projecting Statistics Norway now is implementing its CAI system. The first course in the new technique for the interviewers took place in the middle of February this year. By August an interviewer staff of approximately 150 persons was operative, to relieve the old staff of cirka 250 persons. A rise in the number of working hours per year from approximately 300 to up to 800 will insure that as least as much interview work will be performed, on the whole. According to the scheme all interviewing on paper questionnaires will be abandoned from January 1 1996.

2. The technical point of view

The interviewers of the new staff are equipped with Toshiba T1910 CS laptop computers, with a 120 Mb hard disk and a color screen (passive matrix). The operating system of the computers is Windows 3.1. Each computer has an PCMCIA modem (14400 bps) for communication with the central office.

The interviewing part of the system is, so far, constituted by Blaise 2.5, which has been used by Statistics Norway since 1990. The communication part of the system is a slightly modified version of Microsoft Mail version 3.2. The modifications consist of facilities to receive interview tasks on the laptops and to return interview data. These facilities work nearly automatically, on a «push one button» basis. The regular functions of the MS Mail are not influenced by the modifications and offer very good opportunities for communication with the interviewers. The interviewers can also communicate among themselves through the mail system. It is presupposed that all regular communication with the interviewers shall go through MS Mail from 1996. As the telephone rates in Norway are far cheaper than the postage, this will help in paying back some of the costs for the equipment and in making the CAI system economically sustainable(!!).

One of the main concerns of the project has been to utilize standard «shelfware» programs which could be modified to suit our needs, rather than becoming dependent on solutions programmed especially for the task. The combination of Blaise and Microsoft Mail offers most of the facilities we need to be able to manage interview projects with a decentralized staff of interviewers. Interview projects are distributed to the interviewers as «special» messages in Microsoft Mail. A facility built into Microsoft Mail takes control over these messages and puts the projects in their right place, according to instructions contained in the messages. The same facility is utilized to send more respondents to an interviewer, for instance in cases of transfer of respondents between interviewers.

Every evening after completing their interviewing work, the interviewers start another in-built facility of Microsoft Mail that «snoops» around the hard disk, collects all completed interviews, compresses and encrypts these data and attaches them to a message of the special type. In one operation this facility searches all interview projects for completed data. If needed this facility can easily be instructed from our central office to collect **all** data from the Blaise systems, whether completed or not.

The facilities built into Microsoft Mail all consist of two parts, an interface to the mail system, programmed in Visual Basic, and an interface to the Blaise part, consisting of MS-DOS bat-files and DOS

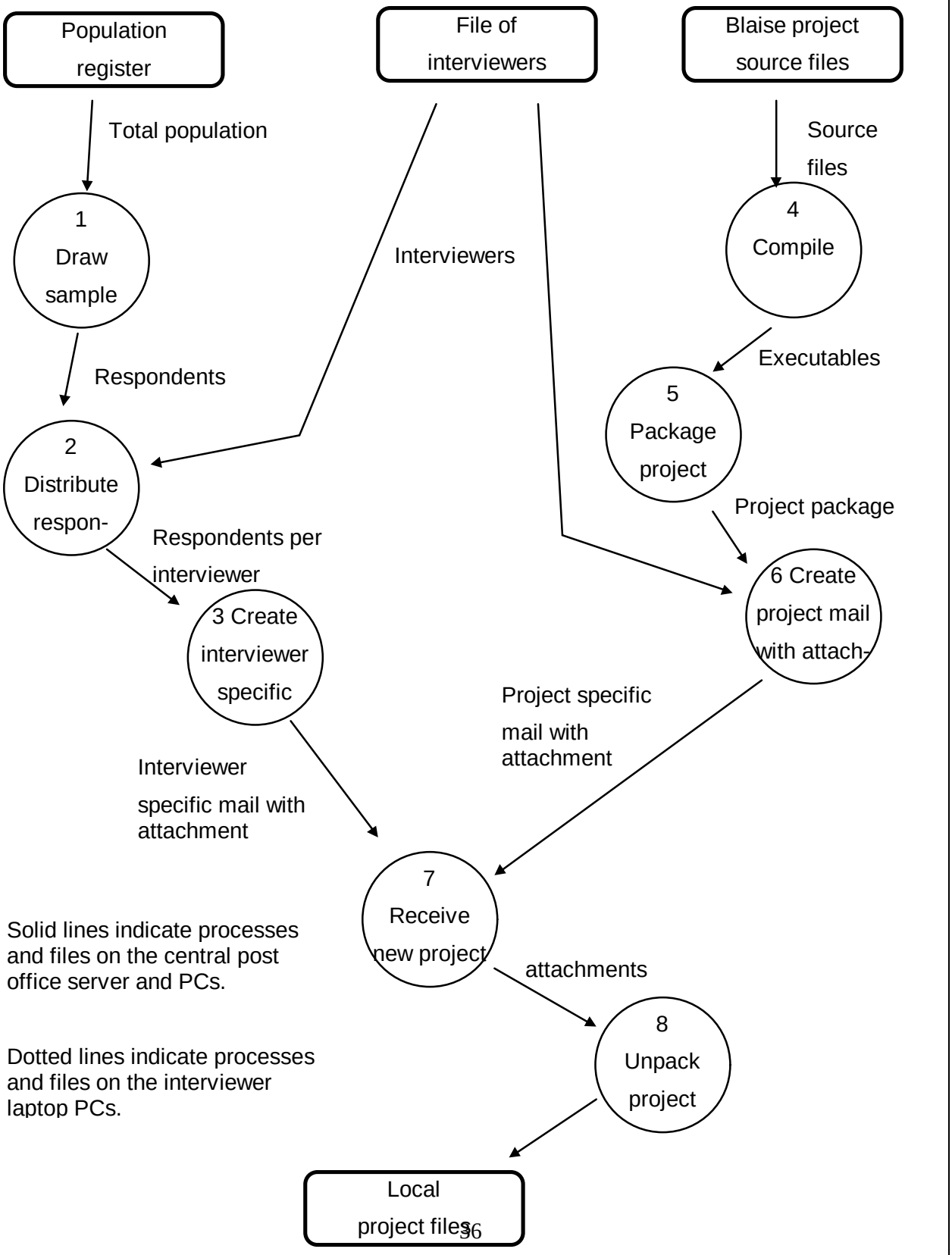
programs compiled with Turbo Pascal. As Statistics Norway for the present do not possess the necessary expertise in Visual Basic, the Visual Basic part has been bought from a firm that delivers software solutions. The interface to the Blaise part, however, has been made in-house.

The development of the decentralized part of the system has been completed according to the time schedule for project. Regrettably, this is not due to us having psychic powers as project planners. As usual in projects of this size we have used more hours than planned for ahead. To balance the schedule, it has been necessary to postpone the development of the central administration part of the system. This task will be a main concern for us next year. In the meantime we have to manage the projects in more primitive ways.

In the trainng period, from February to August this year, a number of smaller surveys has been conducted on the laptops. The main objectives have been to exercise the interviewers and test our system. The testing has pointed out to us some parts of the laptop system that should be improved. The part that mostly needs improvement, is the decentralized respondent management part. As it is now the projects are managed individually, with no synchronisation between them. In periods with more surveys being conducted at the same time, the interviewers may experience some problems with double booking appointments with the respondents of the surveys. We need to develop a system that can manage all the respondents of an interviewer in an integrated way, no matter how many surveys they belong to.

As you will have understood by now, we do not believe our system to be the perfect solution. On the other hand, one might wonder what the perfect solution should look like? Probably the mixing of MS-DOS and Windows is the greatest weakness of our system. The mixing of these two systems seriously limits the possibility for integration of the different software parts. Windows alone would offer a nicer and more uniform user interface than our present system. You may ask why we did not go for a consistent MS-DOS system. The reason is that we anticipate Windows to become the prevailing programming environment in the future, and we thought we could just as well get used to the idea already now. So you will understand that we are looking forward to a Windows version of Blaise. Then our interviewers will be prepared and ready for the task.

Dataflow from Statistics Norway to the interviewers



3. Computer assisted payroll system

The transition to computer assisted interviewing has had consequencies for certain administrative routines as well. Earlier the interviewers filled in a special paper form with the necessary information for paying their wages and rembursing their expenses. The transition to computer assisted interviewing has made it possible to simplify this process in two ways. Firstly, much of the information needed for payroll work is now coded into the interview data returned from the interviewers, for instance time spent on each interview, type of interview (telephone or personal visit). Secondly, instead of filling in the paper form the interviewers now every evening complete a tiny «interview» with themselves, where data concerning their work that day and possible expenses are specified. These data are returned to the central office in the same way as data from regular interviewing projects, and thereafter enter the payroll system of Statistics Norway.

4. Recruitment and training of the new staff

Approximately 65 percent of the interviewers have been recruited from the old «manual» interviewer staff. The old staff had a slightly skewed distribution between the sexes with 57 percent women. The new staff has roughly the same distribution with 54 percent women.

The average age in the old staff was 49 years; 48 years for the women and 51 years for the men. The same average holds for the members of the old staff that have joined the new staff. There is no tendency for the older interviewers to drop out in the confrontation with the new technique.

The average age of the newly recruited interviewers is 40 years across the sexes, i.e. about 10 years lower than the average of the old staff. There is a more pronounced difference between the sexes as well: The average age is 37 years for the women and 45 years for the men.

5. Training

To be able to use the new equipment, the interviewers were given two courses of two days each. The first course concentrated on teaching basic Windows and some communication. The second course concentrated on the modifications of the MS Mail and on interviewing with Blaise. Between the two courses there was a period of two to three weeks. In this period the interviewers were given some home-work to exercise what they had learnt and to familiarize themselves with the computers. There was a homework period after the second course as well. Shortly after this period the interviewers were assigned to interview projects.

The courses were planned and carried out as a sequence of theoretical plenary sessions and practical group sessions where the theory from the plenary sessions were practised. The duration of a typical session was 45 minutes, succeeded by a 10-15 minutes' pause. As course sites were chosen hotels in 4 different towns in Norway.

The teaching staff consisted of 6 persons, 3 main teachers who were responsible for the plenary sessions and 3 assistant teachers who supported during the practical group sessions. Four teachers (2 main + 2 assistants) were assigned to each course. The average number of participants for each course was 17. For the practical group sessions the participants were split into two groups. The plenary sessions were per-formed by two teachers, and each group was served by one main teacher and one assistant. The

atmosphere during the sessions was kept as informal as possible to encourage initiative and cooperation from the participants, and to turn the edge of possible «datafobia»

The most important part of the teaching equipment, apart from the laptops, was a video projector purchased for the occasion. With this projector it was possible to project the laptop screen, in colours, to canvas screens of 2x2 meters and bigger. The projector was of tremendous help during the plenary sessions.

6. Background and expectations

Before the courses started the participants answered a questionnaire (the **pre** questionnaire, on paper) surveying their background and expectations for the course and their future work. A similar questionnaire (the **post** questionnaire, computer assisted) was filled in when they had completed their course. As the last course before summer (the eighth) was concluded in the middle of June this year, we have not yet had time to analyze properly the data from the questionnaires. Instead we aim to present some of these data at the conference in September. In this article we include some preliminary data from the **pre** questionnaires. The tables are based on the answers from the 7 first courses, 120 persons in all.

Table 1 shows that the male interviewers were much more accustomed to computers than the female ones. While as much as 48 percent of the men used computers at least weekly, the same was true for only 39,4

Percentages	Absolute numbers	Total %	Frequency of computer usage: Never	Once a month or less	Every second week	Weekly	Daily	Missing
Total women....	68	100,0	48,5	14,7	5,9	13,2	16,2	1,5
Age groups								
20-29	7	100,0	28,6	42,9	-	28,6	-	-
30-39	18	100,0	55,6	5,6	11,1	16,7	11,1	-
40-49	17	100,0	52,9	11,8	5,9	11,8	17,6	-
50-59	20	100,0	40,0	15,0	5,0	10,0	25,0	5,0
60+	6	100,0	66,7	16,7	-	-	16,7	-
Total men.....	52	100,0	19,2	15,4	13,5	11,5	36,5	3,8
Age groups								
30-39	9	100,0	11,1	-	33,3	22,2	22,2	11,1
40-49	18	100,0	11,1	16,7	11,1	16,7	44,4	-
50-59	17	100,0	29,4	29,4	11,8	5,9	23,5	-
60+	8	100,0	25,0	-	-	-	62,5	12,5

Table 1: Frequency of computer usage

percent of the women. Nearly half the women and one fifth of the men had never used a computer!

Table 2 shows the same fact from another angle: While not even half the women had attended a computer course, nearly three quarters of the men had done so. The men were also more acquainted with Windows

Percentages	Total	**Attended a computer course**		**Used Windows before**		
		Yes	No	Yes	No	Missing
Total women	100,0	45,6	54,4	41,2	55,9	2,9
20-29	100,0	71,4	28,6	71,4	28,6	-
30-39	100,0	44,4	55,6	61,1	33,3	5,6
40-49	100,0	47,1	52,9	35,3	64,7	-
50-59	100,0	40,0	60,0	30,0	65,0	5,0
60+	100,0	33,3	66,7	-	100,0	-
Total men	100,0	71,2	28,8	69,2	28,8	1,9
30-39	100,0	88,9	11,1	77,8	22,2	-
40-49	100,0	61,1	38,9	72,2	27,8	-
50-59	100,0	70,6	29,4	70,6	23,5	5,9
60+	100,0	75,0	25,0	50,0	50,0	-

Table 2: Attended a computer course & Used Windows before

than the women. For the latter variable there is a clear correlation with the age of the interviewers as well.

The participants of the courses were optimists - at least when they filled in the **pre** questionnaires, as table 3 shows. There is no discernible difference between the sexes for this variable, but the older participants

Percentages	Total	**It is hard to learn to use a PC**			
		Mostly true	Neither	Mostly false	False
Total women	100,0	2,9	19,1	58,8	19,1
20-29	100,0	-	-	71,4	28,6
30-39	100,0	-	-	72,2	27,8
40-49	100,0	-	41,2	47,1	11,8
50-59	100,0	10,0	20,0	60,0	10,0
60+	100,0	-	33,3	33,3	33,3
Total men	100,0	-	21,2	61,5	17,3
30-39	100,0	-	11,1	66,7	22,2
40-49	100,0	-	11,1	72,2	16,7
50-59	100,0	-	35,3	58,8	5,9
60+	100,0	-	25,0	37,5	37,5

Table 3: It is hard to learn to use a PC

are somewhat more reserved than the younger ones.

With regard to the question whether PCs will make the interviewer job more easy or difficult (table 4), it is interesting to note that about 40 percent of the interviewers don't take a stand on the question. We were

Percentages	Total	**PCs make the job more easy or difficult**				
		More easy	More difficult	Neither	Don't know	Missing
Total women	100,0	52,9	4,4	20,6	20,6	1,5
20-29	100,0	85,7	-	14,3	-	-
30-39	100,0	66,7	-	-	27,8	5,6
40-49	100,0	47,1	5,9	35,3	11,8	-
50-59	100,0	30,0	10,0	30,0	30,0	-
60+	100,0	66,7	-	16,7	16,7	-
Total men	100,0	51,9	5,8	36,5	5,8	-
30-39	100,0	66,7	11,1	22,2	-	-
40-49	100,0	44,4	-	50,0	5,6	-
50-59	100,0	58,8	11,8	29,4	-	-
60+	100,0	37,5	-	37,5	25,0	-

Table 4: PCs make the job more easy or difficult

may be not good enough to sell the concept in advance?

As table 5 shows the interviewers seem to have a stronger faith in the properties of computerized interview systems when it comes to securing that correct information is entered during the interviews.

Percentages	Total	**More difficult to correct errors on a PC**			
		Mostly true	Neither	Mostly false	False
Total women .	100,0	-	13,2	61,8	25,0
20-29	100,0	-	-	71,4	28,6
30-39	100,0	-	11,1	44,4	44,4
40-49	100,0	-	23,5	58,8	17,6
50-59	100,0	-	15,0	70,0	15,0
60+	100,0	-	-	83,3	16,7
Total men ...	100,0	1,9	19,2	38,5	40,4
30-39	100,0	11,1	11,1	22,2	55,6
40-49	100,0	-	22,2	44,4	33,3
50-59	100,0	-	17,6	47,1	35,3
60+	100,0	-	25,0	25,0	50,0

Table 5: More difficult to correct error on a PC

Professional esteem is an important issue in all professions, and not least for the interviewers. For an interviewer the respondent's attitude towards her or him may constitute the difference between an interview and a refusal. Table 6 shows that the interviewers do not feel certain in this matter. There is a

	Total	**PCs increase interviewers' reputation**					
Percentages		True	Mostly true	Neither	Mostly false	False	Mis-sing
Total women .	100,0	4,4	19,1	50,0	23,5	2,9	-
20-29	100,0	-	42,9	57,1	-	-	-
30-39	100,0	5,6	27,8	61,1	5,6	-	-
40-49	100,0	5,9	11,8	52,9	23,5	5,9	-
50-59	100,0	5,0	10,0	40,0	40,0	5,0	-
60+	100,0	-	16,7	33,3	50,0	-	-
Total men ...	100,0	5,8	26,9	46,2	15,4	3,8	1,9
30-39	100,0	11,1	33,3	44,4	11,1	-	-
40-49	100,0	11,1	38,9	33,3	16,7	-	-
50-59	100,0	-	17,6	47,1	17,6	11,8	5,9
60+	100,0	-	12,5	75,0	12,5	-	-

Table 6: PCs increase interviewers' reputation

tendency that the younger interviewers are more optimistic than the older ones, and the men are a trifle more convinced than the women.

In view of this it is no surprise that the majority of the interviewers do not feel certain as to what the attitude of the respondents will be towards being interviewed with computers (table 7). The women are, on the average, somewhat more concerned than the men, and the older ones tend to be less enthusiastic

Percentages	Total	**Expected respondent reaction to the PCs**				
		Posi-tive	Nega-tive	Neither	Don't know	Mis-sing
Total women .	100,0	22,1	8,8	27,9	38,2	2,9
20-29	100,0	42,9	-	42,9	14,3	-
30-39	100,0	33,3	11,1	11,1	44,4	-
40-49	100,0	17,6	-	35,3	47,1	-
50-59	100,0	10,0	15,0	25,0	40,0	10,0
60+	100,0	16,7	16,7	50,0	16,7	-
Total men ...	100,0	19,2	3,8	51,9	25,0	-
30-39	100,0	22,2	11,1	66,7	-	-
40-49	100,0	27,8	-	50,0	22,2	-
50-59	100,0	11,8	5,9	52,9	29,4	-
60+	100,0	12,5	-	37,5	50,0	-

Table 7: Expected respondent reaction to the PCs

than the younger ones.

The previous questions have shown that the interviewers were looking forward to learn to use and to use the computers as a tool in their work, though they were not equally convinced that the respondents would feel the same way. A major question, then, was how they summed up these loose ends to themselves before the start of the course; in other words a question about their motivation for the job. Table 8 shows

	Total	**PCs make the work more interesting**				
Percentages						
		True	Mostly true	Neither	Mostly false	False
Total women .	100,0	20,6	39,7	32,4	5,9	1,5
20-29	100,0	42,9	14,3	42,9	-	-
30-39	100,0	27,8	33,3	38,9	-	-
40-49	100,0	17,6	47,1	17,6	11,8	5,9
50-59	100,0	5,0	40,0	45,0	10,0	-
60+	100,0	33,3	66,7	-	-	-
Total men ...	100,0	23,1	36,5	32,7	7,7	-
30-39	100,0	11,1	55,6	33,3	-	-
40-49	100,0	27,8	38,9	22,2	11,1	-
50-59	100,0	23,5	35,3	29,4	11,8	-
60+	100,0	25,0	12,5	62,5	-	-

Table 8: PCs make the work more interesting

that about 60 percent of the interviewers expect their work to become more interesting with computers, while about one third don't think the computers will make any difference. It is a bit worrying, though, that about 10 percent of the age group 40-59, for both sexes, expect their work to become less interesting.

We want to repeat that the tables above are based on questionnaires filled in before the courses started. It will be interesting to see if the viewpoints of the interviewers have changed during the courses. We will return to this question at the conference in September.

CAPI PLUS et Blaise III : une organisation générale pour les enquêtes de l'Insee

F. Dussert et G. Luciani, Institut National de la Statistique et des Etudes Economiques (INSEE) France

La réussite du passage sur micro-portable de la collecte de l'enquête emploi puis celle de l'enquête budget de famille conduisent l'Insee à généraliser son utilisation pour toutes les enquêtes auprès des ménages.

CAPI PLUS met en place un dispositif général qui doit permettre d'intégrer rapidement tout nouveau questionnaire dans une structure préétablie. Hormis les transmissions, ce dispositif doit intégrer à la fois la gestion de la collecte (case management), le poste de contrôle et d'apurement, définir une ergonomie du poste enquêteur et des normes de développement. Blaise III offre de ce point de vue de grandes possibilités.

1. De l'enquête emploi à CAPI PLUS

Après une implantation progressive, la collecte de l'enquête emploi (Labour Force Survey) est réalisée entièrement sur micro ordinateurs portables depuis mars 1994 : les 900 enquêteurs ont tous utilisé le micro et transmis leur données vers le centre collecteur central à l'aide d'un logiciel spécialisé (Arbiter). En 1994-1995, l'enquête budget de famille (Family Expenditure Survey) était également réalisée avec CAPI, mais les transmissions n'étaient pas assurées : les enquêteurs envoyaient leurs données sur disquette à l'Insee régional pour vérification. A la différence de l'enquête emploi, un poste de vérification (manipula) permettait de vérifier la collecte. En 1995 également, la direction régionale de l'Insee à l'île de La Réunion (dans l'Océan Indien) développait une enquête logement en Blaise 2.4. Chacune de ces applications est réalisée indépendamment des autres.

La réussite en qualité et en délai de ces opérations conduit l'Insee à généraliser progressivement l'utilisation de la saisie sur micro-portable pour toutes ses enquêtes. Restait à créer une véritable infrastructure qui puisse "accueillir" toute nouvelle enquête auprès des ménages.

2. Pourquoi une infrastructure générale ?

D'une enquête à l'autre il a été difficile d'utiliser le travail déjà réalisé ; ainsi le système mis en place pour l'enquête emploi n'a pu être réutilisé pour l'enquête budget de famille notamment pour les transmissions et chaque opération a un coût élevé en conception informatique.

Cette dissociation des opérations conduit à investir de manière importante à chaque nouvelle enquête dans la programmation, l'organisation et la cohabitation sur un même micro de plusieurs

applications. De plus, chaque fois, l'enquêteur doit s'adapter à une nouvelle ergonomie. Par souci d'offrir aux enquêteurs un poste de travail confortable et efficace, il est nécessaire de leur fournir un outil avec un seul mode d'accès quelque soit l'enquête.

Cette difficulté à capitaliser l'expérience est en partie liée à l'organisation de l'Insee.

D'une part chaque "concepteur" d'une enquête en est responsable de sa conception à sa publication : il élabore le questionnaire, rédige les instructions aux enquêteurs puis exploite les résultats et réalise des études à partir du fichier "nettoyé". La durée d'un cycle d'enquête est de trois ans environ. Dans ce cadre, l'apprentissage d'un logiciel de questionnement est une contrainte supplémentaire et il n'est pas souhaitable qu'il investisse dans une programmation relativement complexe.

D'autre part, la sphère informatique est dissociée de la sphère statistique. Dans ce contexte, les rapports entre les uns et les autres doivent être formalisés et les rôles précisés.

Enfin, la responsabilité de la collecte est éloignée géographiquement de la conception. Le réseau de collecte est décentralisé et chaque région Insee est responsable de la gestion, la formation et le contrôle des enquêteurs de sa région. Cette façon de faire nécessite une formalisation et une coordination importante des procédures.

3. Une infrastructure matérielle, organisationnelle et logicielle

L'infrastructure générale de CAPI PLUS vise à mettre à la disposition de tout concepteur désireux de réaliser une enquête en saisie portable l'infrastructure organisationnelle, matérielle et informatique. Elle recouvre à la fois un réseau de collecte équipé et formé à la réalisation d'enquêtes en saisie portable, une infrastructure de communications et de recueils des résultats, un logiciel général de questionnement, un corps de règles et de méthodes. Les différents postes de travail concernés : ceux des enquêteurs, ceux des gestionnaires d'enquêtes en région et le poste du statisticien combinent ces composants.

1. Le poste de travail de l'enquêteur comprend un module de formation, une messagerie et un menu lui permettant d'accéder aux enquêtes à traiter. Pour chacune d'elle, il dispose d'un carnet de tournée pour obtenir, en amont la liste des logements à enquêter et "entrer" dans le questionnaire, en aval l'état d'avancement de son travail et son carnet de rendez-vous.

2. Le poste de gestion de l'enquête en région permet de récupérer l'échantillon, charger les micros des enquêteurs et recevoir les données. Des programmes standards permettent également pour chaque enquête de suivre l'avancement de la collecte de l'enquêteur et vérifier la qualité des questionnaires à partir de contrôles statistiques paramètres. Ce poste doit permettre également de faire les premiers redressements.

3. Le poste de travail du "concepteur-statisticien" auquel le projet CAPI PLUS fournit un ensemble de programmes utilisables sans modification pour toutes les enquêtes. Il constitue en quelque sorte l'ossature du questionnaire que le statisticien va concevoir. Il comprend :

- un ensemble de questions sur l'unité statistique à repérer qui porte sur les caractéristiques du logement et des indications de repérage. Ces indications sont nécessaires pour établir les pondérations ;
- un ensemble de questions appelé "tronc commun" permet de connaître la composition du ménage et les caractéristiques essentielles de chacune des personnes. Ce tableau de composition du ménage permet de définir la personne de référence. Il est utilisé dans la plupart des enquêtes sur les conditions de vie;
- des questions de validation en fin de questionnaire qui permettent de dresser un bilan de collecte.

Le questionnaire de l'enquête sera "glisser" dans cette ossature.

Restait à choisir le logiciel à utiliser pour construire cette infrastructure.

4. L'expérience de Blaise II

Nous avons rencontré avec Blaise II (dans les versions 2.39 et 2.4) des limites de trois ordres :

1. L'obligation de scinder des questionnaires volumineux en plusieurs "*setups"* Blaise, les limites de taille étant assez rapidement atteintes (4 *setups* Blaise pour l'enquête Budget de Famille). Cette contrainte nous a conduit à réaliser des développements complémentaires spécifiques pour assurer la cohérence entre les différentes parties du questionnaire ;

2. Le nécessaire détournement de la fonction du "*BLOCK APPOINTMENT"* pour permettre l'affichage de carnet de tournée (liste des logements à enquêter, le choix d'un logement dans la liste provoquant l'ouverture du questionnaire) ;

3. Une programmation informatique complexe qui nécessite une formation assez lourde pour des statisticiens.

5. L'Insee choisit Blaise III.

L'annonce de la version 3 du logiciel Blaise, nous incitait à reconsidérer le choix de Blaise II dans la mesure où une reconversion des enquêtes antérieures était de toute manière nécessaire.

Un premier examen des logiciels du marché nous a conduit à étudier l'un d'eux utilisé par les principaux instituts de sondage français. Il semblait pouvoir supporter sérieusement la comparaison avec Blaise. Nous avons donc fait une investigation poussée sur ce logiciel : réalisation d'un questionnaire regroupant les principales caractéristiques des différentes enquêtes de l'Insee.

La force de ce logiciel réside d'abord dans son interface de développement, puisque les questions sont saisies dans la forme où elles apparaîtront à l'enquêteur *(WySiWyg)*. Le mode de programmation des contrôles nous a paru convivial. Il ne semblait pas y avoir de limites pour les gros questionnaires, puisque le programme (<500K) utilise toutes les données du questionnaire comme des données externes (comme le fait aujourd'hui Blaise III). Il fournissait en outre un poste de gestion de la collecte avec des fonctions de contrôles et de tabulations des données sans investissement supplémentaire, conçu plus particulièrement pour CATI.

Mais il n'était pas possible, avec ce logiciel, de remonter dans un questionnaire sans effacer les données saisies. Ce choix est conforme à la pratique des instituts de sondage dans la réalisation des interviews : un questionnaire (souvent court) doit être déroulé sans hésitation du début à la fin, l'enquêteur est très guidé (impératifs de rendement et de rapidité). Dans le cas d'un institut de statistiques, le souci d'obtenir la plus grande qualité possible des réponses au cours de la collecte, pour des questionnements souvent complexes est primordiale. Une plus grande autonomie doit être laissée à nos enquêteurs, et les aller-retours dans le questionnaire sont fréquents et nécessaires.

Blaise III nous a donc finalement paru plus adapté au mode de collecte à l'Insee. D'ailleurs Blaise est utilisé, ce n'est sans doute pas un hasard, par bon nombre d'instituts nationaux de statistiques. Avec Blaise III, les contraintes de taille de questionnaire ont été levées, et la création d'un carnet de tournée est maintenant possible avec Maniplus. De plus à coté de Blaise proprement dit, il existe un ensemble d'outils périphériques qui permet de réaliser l'ensemble des traitements d'une enquête dans le même contexte informatique.

6. Le carnet de tournée en Maniplus :

La version 1.1 de Blaise 3 inclura le nouvel outil Maniplus, dont l'utilité nous paraît essentielle.

Jusqu'à présent, il était nécessaire de réaliser les interfaces des postes de travail avec un outil spécifique : nous avons utilisé Turbopro, puis C, et Clipper. Maniplus offre cette possibilité. Ce nouvel outil nécessite un apprentissage léger, et présente l'avantage d'être évidemment tout à fait

cohérent avec les autres modules de Blaise, et donc de simplifier le développement des applications CAPI, et d'en faciliter la maintenance.

Maniplus nous permet de fournir à l'enquêteur un carnet de tournée qui contient la liste des unités à enquêter. L'enquêteur commence toujours par ouvrir son carnet de tournée: pour chaque unité à enquêter, il visualise l'identifiant du logement, son adresse, des précisions de repérage, un état de son travail (non validé, bon, transmis), un résultat de collecte (logement vacant, résidence occasionnelle, résidence secondaire, impossible à joindre.), un suivi de collecte (refus, abandon, terminé). Ces informations lui permettent de gérer au mieux sa tournée en lui fournissant un aperçu général de l'état de sa collecte. Le choix d'une unité dans ce carnet lui permet d'ouvrir le questionnaire. Maniplus donne une solution tout à fait satisfaisante pour répondre à ce besoin.

Maniplus permet le calcul de variable à l'extérieur du programme de questionnement. Le fonctionnement asynchrone de Blaise rend impossible par exemple le calcul du temps de questionnement (lorsqu'un questionnaire est ouvert plusieurs fois, le temps est recalculé), et le nombre d'ouvertures d'un questionnaire. Maniplus permet cela.

7. Programmer en Blaise : facile ou non

Notre expérience de la programmation en Blaise III est très récente, mais il semble qu'elle reste complexe. Elle nécessite un investissement important en formation, et la mise au point d'un questionnaire est longue et difficile surtout en période d'apprentissage.

Nous avions constaté avec Blaise II deux caractéristiques principales:

1. Le temps d'apprentissage, 2 à 3 mois pour être opérationnel, est lié aux multiples possibilités du logiciel. Les différentes fonctions doivent être programmées dans des langages qui ne sont pas très cohérents (*DEP, MANIPULA, CAMELEON, ABACUS,...*). La séparation des *CHECKS* et des *RULES* complique la tâche par rapport à une programmation classique. Il est obligatoire de faire fonctionner l'analyseur syntaxique pas à pas, ce qui ralentit la mise au point d'un programme. Avec Blaise III, il n'y a pas d'amélioration sensible sur ces points.

2. Le langage est très complet. Si on ne met pas en balance en permanence l'avantage (obtenir un questionnaire convivial) et le coût (temps de programmation nécessaire), on aboutit facilement à un programme volumineux.

Pour conclure, même si des progrès importants ont été faits en Blaise 3 pour faciliter le développement des questionnaires (notamment le mode *Windows*, un paragraphe RULES unique, les INCLUDE), nous pensons que la maîtrise du logiciel nécessite encore un investissement important.

8. Un pôle de compétence Blaise

Dans ce cadre, plutôt que de confier la programmation du questionnaire à chaque concepteur, il nous paraît préférable de spécialiser une unité de développement à laquelle ils pourront faire appel. Ce pôle a quatre missions :

1. l'expertise des versions successives du logiciel et la formation des concepteurs.
2. le développement des questionnaires et la validation des questionnaires développés par des concepteurs.
3. la maintenance des applications et notamment de l'ossature de questionnaire.
4. l'établissement des normes d'ergonomie, d'écriture, de maintenance et de bonne insertion dans CAPI pour le développement.

Cette centralisation du développement permettra de profiter plus facilement des expériences successives de programmation et de "ramasser" la compétence en Blaise III. Il facilitera l'élaboration de standards de développement et la création progressive d'une bibliothèque de programmes utilisables.

9. La nécessité d'établir des normes

Blaise III est un système très souple qui offre la possibilité de personnaliser son utilisation. Cette souplesse est à double tranchant : laisser chaque concepteur ou programmeur définir la personnalisation induirait pour les enquêteurs et les gestionnaires régionaux des difficultés d'utilisation. C'est dans cet esprit qu'il est recommandé, pour chaque enquête d'utiliser l'architecture de questionnaire proposée sur le poste de travail du statisticien. Dans cet esprit, il nous paraît nécessaire de définir progressivement un corps de normes :

1. Des normes de programmation

Tout développeur de programme de questionnement doit respecter ces normes, définies par le pôle de compétence Blaise.

Par exemple:

- Les mots du langage Blaise sont écrits en majuscules.

- Les noms de *FIELDS* sont composés de 8 caractères au maximum. Cette contrainte est nécessaire pour faciliter la production de dictionnaires SAS.

- Les noms de *FIELDS* doivent être significatifs, dans la mesure où ces noms sont affichés sur l'écran de l'enquêteur devant la zone de saisie.

- Un nom de *BLOCK (TABLE)* commence par B (T) majuscule. On note en commentaire le nom du *block* (*table*) auquel se rattache une instruction *ENDBLOCK (ENDTABLE).*

- Un nom de *TYPE* (excepté pour les *BLOCKs* et les *TABLEs*, Cf. ci-dessus) commence par Typ, suivi du nom de la variable décrite par ce *TYPE.*

- Règles d'indentation des *IF, THEN, ELSE, ELSIF.*

-Règle d'écriture des *TYPE,* des *CHECKS.*

-Etc..

2. Une norme de présentation de l'écran de saisie

Le paramétrage de la présentation de cet écran s'effectue par des modifications sur le fichier *MODELIB* (et bientôt dans une section *LAYOUT* du programme), et sur le fichier *DEPMENU* (activer ou inhiber certaines fonctions). Un écran standard a été défini, que les enquêtes devront respecter au maximum.

3. Les touches fonction.

Le fichier *DEPMENU* permet également de définir les touches fonction. Notre souci est d'éviter au maximum les combinaisons de touches, et de ne pas bouleverser les habitudes déjà prises.

Un exemple de normalisation :

DESCRIPTION	TOUCHES
Aide à l'utilisation de Blaise	F1
Aide contextuelle (sur les questions)	F12
Mode édition	F2, Backspace, insert.
Afficher la suite d'une question	F8
Se positionner sur une question précise	F5
Confirmer une réponse douteuse	F3
Passer du champ de saisie à la liste des modalités de réponses	F6
Ouvrir une fenêtre remarque	F4
Interrompre un questionnaire, ouvrir un bloc parallèle	CTRL+ENTER
Annuler l'option sélectionnée	Escape
Reconduire une même réponse à une même question dans un tableau	/

Pour codifier "ne sait pas "	?
"refus de réponse"	!
Pour se placer tout au début, à la fin d'un questionnaire	HOME, END
Pour remonter d'une étape dans les menus	ESC
Pour avancer, remonter, par page d'écran	PGDN, PGUP
Pour se déplacer question par question	← ↑ → ↓
Pour effacer le caractère précédent le curseur	BKSPACE
Pour effacer le caractère sur le curseur	DEL
Pour valider la saisie, une option	ENTER
Relancer le micro	CTRL+ALT+DEL

4. Le langage

Blaise est livré en néerlandais et en anglais. Nous devons livrer des interfaces utilisateurs en français. La possibilité de traduire les messages Blaise dans une autre langue ne lève pas toute la difficulté : les problèmes sémantiques subsistent. Il nous faut d'une part adapter les termes induits par la nouvelle technique CAPI à la culture existante en matière d'enquête, et d'autre part trouver les termes adéquats pour désigner les nouveaux concepts. Ce travail de linguiste est délicat, et doit être mené avec un grand soin très tôt, afin de faciliter pour les enquêteurs et les gestionnaires d'enquêtes l'apprentissage de la réalisation des enquêtes par CAPI.

L'infrastructure CAPI avec Blaise III sera mise en place à la fin de l'année 1995 et testée sur les premières versions de l'enquête logement ; la réalisation en vraie grandeur se fera au quatrième trimestre 1996. Parallèlement, l'enquête trimestrielle emploi est développée en Blaise III pour juin 1996. Il n'est pas envisagé de convertir l'enquête emploi en (Blaise II) avant sa refonte en 1999. Les autres enquêtes auprès des ménages passeront progressivement sur micro-portables. La demande est forte de la part des concepteurs qui y voient des gains en qualité et en délais, comme des enquêteurs qui préfèrent, dans une très grande majorité, ce mode de travail. Mais le passage du papier au micro, pour être rentable, nécessite la mise en place de procédures harmonisées pour les informaticiens comme pour le statisticien dans la conception du questionnaire.

Training Interviewers in the Use of Blaise

Chris Goodger, Office of Population Censuses and Surveys, UK

1. Introduction

Over the past 8 years OPCS has become experienced in instructing interviewers in the use of Blaise as an interviewing tool for both CAPI and CATI surveys. This paper concentrates on the CAPI training programme, in particular on the Initial Training Course for interviewers and on the role played by trainers in the field.

CAI is now the basic data collection tool for many organisations. This paper also looks at the new methodology from a trainer's viewpoint, and considers how CAI can best be implemented to facilitate the training of interviewers.

2. A brief history of CAPI training at OPCS

Following 4 trials between March 1987 and July 1990, CAI was introduced on our Labour Force Survey in September 1990. 154 interviewers already working on the PAPI survey were trained between July and September 1990. The training consisted of a 2 day office-based training course.

In September 1991 the Labour Force Survey quadrupled in size, and OPCS decided to recruit and train a separate field force. Interviewers attended a 6 day course covering survey methods and procedures as well as training in CAPI.

The field force used for all of our other surveys, the General Field Force (GFF), gradually saw more of its work being carried out using CAPI. Today over 90% of our total interviewing is CAPI. On joining OPCS all of the GFF interviewers had undergone a 3 day initial training course covering interviewing technique. As CAPI surveys appeared interviewers were trained both on CAPI and the specific survey. Typically these courses lasted 2 days.

Once all of the current GFF interviewers had been trained on CAPI it was then possible to revise the initial training course so that it also covered CAPI.

3. The Initial Training Course

The main part of the course consists of 4 days of training in our London office conducted by headquarters staff assisted by field trainers. Interviewers complete approximately 7 hours of home study before they come into the office.

About 3 hours is spent working through a training pack covering standard interviewing method, and up to 4 hours carrying out practice interviews using paper questionnaires. We do not send out computers in advance of the course. We found in the course of training interviewers for the Labour Force Survey that it was best not to. At that time over 50% of the interviewers had no previous experience of working with computers, and consequently needed a great deal of support from office staff. Today more interviewers have some experience of computers when they join us, and we are rethinking this policy.

At the moment the only information about CAPI that the interviewer receives before the course is a 3 page leaflet designed mainly to reassure them. The leaflet outlines in general terms how CAI works, highlighting its advantages. Interviewers are assured that they need not be computer-literate or have keyboard skills, and that practical sessions feature heavily in the training.

A maximum of 12 interviewers are invited to each course. This is felt to be a manageable number, particularly for the group sessions.

The timetable for the course is as follows:

Day 1	Duration (minutes)
Doorstep and full introductions	120
Interviewing method	60
Sampling	30
Practice introductions	90
Day 2	
Introduction to laptop	**150**
Dummy interviewing (whole group)	**60**
Coding industry and occupation	75
Practice interviewing (groups of 3/4)	**90**

Day 3

Classification	75
Administration, backing up, data transmission	**90**
Planning of work	30
Practice interviewing (groups of 3/4)	**180**

Day 4

Practice interviewing (groups of 3/4)	**120**
Performance management	45
Allocation of work, claims	90

Practical sessions involving use of a laptop computer have been highlighted. These account for 53% of the time spent in the office.

On the first day the sessions follow up on the study the interviewers have completed before attending the course, in particular the practice interviewing.

The first CAPI training session is on the second day. The highlighted sessions in days 2 to 4 are described below.

"Introduction to the laptop" - 150 minutes.

Laptops are distributed and the interviewers are instructed in the following:

a) Basic laptop operation.

Using mains electricity or battery power, adjusting the screen brightness or contrast, the keyboard, what the function keys do, the floppy-disk drive.

b) Case management.

Loading questionnaires onto the laptop, selecting cases to interview, entering/exiting cases.

c) Basics of CAPI.

Entering different types of answer (single response, multiple-response, numeric, text), changing and correcting answers, soft/hard checks, making comments.

Although the procedures for operating the laptop on battery power are covered, the interviewers use mains electricity during the course. We have found this less disruptive during the course, and it is less common in the field for interviews to be carried out using battery power anyway.

New cases are opened for this first session, with the trainer telling the interviewers what they should enter at each stage. The trainer's laptop is connected to a panel-projector so that the interviewers can follow more easily.

The training questionnaire we use has been designed to cover the full range of question types, and a range of topics representative of most of our continuous surveys.

"Dummy interviewing" - 60 minutes.

The trainer takes the role of respondent, asking each course member in turn to take the role of interviewer. All course members key the respondent's answers. The trainer also keys the answers so that they can be refered to on the projection screen.

The purpose of this session is to consolidate the interviewers' understanding of both CAPI and standard interviewing method.

"Practice interviewing" - 390 minutes.

On days 2 to 4 interviewers are split into groups of 3 or 4 for practice interviewing sessions led by a field trainer. The sessions are similar to the "Dummy interviewing", but because of the smaller groups more individual attention can be given by the trainer.

More advanced CAPI techniques, dealing with complex warnings for example, are introduced gradually. As the sessions progress and interviewers become more comfortable with using the laptop, the emphasis shifts from CAPI to standard interviewing method.

"Administration, backing up, data transmission" - 90 minutes.

The whole group reconvenes for a session covering:

a) Accounting for cases.

Recording the outcome (interview, non-response or ineligible), calls made to the address, reasons for refusal or non-contact etc.

b) Backing up work.

Making a back up onto floppy disk.

c) Data transmission.

Preparing cases for transmission, using the modem.

Training cases (example cases which have been constructed to illustrate training points) are used to demonstrate the procedures for accounting for cases and for data transmission. As in previous sessions with the whole group interviewers use their laptops under instruction from the trainer. Once again the projection panel is used.

4. Training on individual surveys

Training requirements can vary considerably from survey to survey, but for most of our continuous surveys training courses are fairly similar.

Most of the training courses, or "briefings" for these surveys consist of 1 day in the office preceded by some advance study. Having attended the Initial Training Course interviewers are expected to be conversant with CAPI, but briefings do afford interviewers a further opportunity to improve their CAPI skills.

Typically, the pre-course material includes a "dummy" interview, or interviews, on audio tape. Interviewers listen to the tape, recording the respondent's answers onto a training case on their laptop. Training points are made on the tape, and it is common for warnings to be triggered and for answers to require amendment.

The day in the office often features a session similar to the "dummy interviewing" in the Initial Training Course. Part of the interview that is especially complex, or that is very specific to the survey, might be used for this purpose.

5. Training in the field

When an interviewer begins work on a survey for the first time they are accompanied by a field trainer. The trainer spends 2 days at the start of the interviewer assignment and 1 day towards its end with the interviewer.

If on the Initial Training Course the interviewer is found to be weak in some respect, this information is fed through to the field trainer. This means that if the interviewer is not sufficiently proficient in CAPI, for example, the trainer will be able to devote more time to this aspect of the training.

The trainer meets the interviewer either at the interviewer's home or in the work area. Before setting out to interview the trainer checks that the interviewer is properly prepared. As far as CAPI is concerned this means checking that the interviewer has the correct questionnaire on the laptop and 2 fully-charged batteries.

Clearly, at this stage it is very important that the interviewer feels confident about operating the laptop. If they are not, the trainer is able to offer them one-to-one tuition.

The trainer always conducts the first interview. The interviewer follows the interview by keying the respondent's answers into their own laptop. As well as the interviewer having a chance to observe a properly conducted interview, this allows them to practice keying answers in a real interview situation.

When it comes to the next interview, although the interviewer is conducting the interview the trainer also keys the respondent's answers. This can be very reassuring for the interviewer because should they encounter a problem with the laptop, all of the data has been safely stored on the trainer's laptop.

After the interview has been completed and they have left the respondent's house, the trainer will go through the completed interview with the interviewer, checking the accuracy of the interviewer's recording.

Over the last year we have acquired a new tool to assist the field trainers. We are using software that enables us to link the interviewer's and trainer's laptops. The interviewer's "host" laptop is connected by wire to the trainer's "remote" laptop. On the remote laptop screen it is possible to see exactly what is appearing on the host's screen, but only the host's keyboard is operable.

The main advantage is that the person using the remote laptop can see exactly what is being keyed on the host laptop. This can save the time usually spent in going through the interview together after leaving the repondent's house.

For the trainer, because they are not keying the answers into their laptop, it also allows them to pay more attention to the interaction between the interviewer and the respondent.

However, some interview situations can mean the system is not practical. There is a potential hazard in trailing the lead linking the laptops across the room in which the interview is taking place. The link can only be used safely when the laptops can be positioned fairly closely together, so we ask trainers to assess whether or not it is appropriate to use the link once they are in the respondent's house.

6. How CAI can aid the training process

My colleague, Paul Hunter, presented a paper at the last IBUC about "Introducing CAI standards within OPCS", in which he described the advantages of establishing standards. The introduction of consistent practices between surveys can only be beneficial to trainers.

Standardisation of screen layout between surveys is particularly helpful. It is also important that the computer offers as much help as possible to the interviewer. Screen messages should be informative, and appropriate help screens provided. Case management systems should be designed to be intuitive. The interviewer should be allowed to both control and see what is happening.

CAI should make it easier to maintain standards. Having standard Blaise code that can be copied from survey to survey should ensure that core classification questions, for example, are always asked in the same way.

There are areas still to be explored in offering assistance to interviewers through Blaise. One example would be storing instructions or definitions relating to specific questions within the questionnaire, so that they could be accessed by the interviewer during the interview.

I'm sure there are also possibilities for computer based training packages to be designed for interviewers learning the basics of CAPI. Distance learning can be very effective, and a properly produced computer package would make it possible for interviewers to start learning about CAPI at home.

7. Conclusion

Effective training programs are an essential part of the survey process. The move to CAI has meant new challenges for trainers, and it is essential that those designing training programs not only keep pace with the resultant changes in methodology, but also take full advantage of advances in technology.

An Object-Based Approach for the Handling of Survey Data.

James Gray, Office of Population Censuses and Surveys, UK

1. The nature of the Case Management task

1.1. Basic case management tasks on a laptop

The main case management tasks on a CAPI laptop are:

- to receive a batch of work,
- to interview,
- to send back completed cases.

In addition, the laptop needs to take care of data security and backups etc.

There will be a need to extract completed cases from the laptop (usually with respect to a questionnaire variable, eg outcome code, or using the Blaise cleanliness status). An immediate question is whether or not to delete the extracted cases from the Blaise database on the laptop. Do you risk duplication, or do you risk losing a copy?

1.2 Multiple surveys

Where there are several surveys in the field, the situation will be more complicated. The variables that determine whether a case can be sent back may be in a different place on the questionnaire, and may well have a different coding scheme. The nature of the serial number may change.

The answer will be either to make the surveys more similar, by standardisation, or to make the laptop system more flexible.

1.3 Increasing complexity

Different surveys have different needs. This makes it difficult both to standardise surveys, and to create sufficiently flexible systems.

One major complication for some surveys under Blaise 2 is the need for more than one instrument to interview a household. In these cases, the tasks of selecting households to interview, rostering within the household and extracting complete cases from the interviewer workload databases are greatly complicated.

There are also occasions where there is a need for a non-Blaise program to be run as part of the interview task. Again this complicates the job of the laptop system.

1.4 Organisation effects

With an increasing number of surveys, with diverse needs, one soon finds a large number of staff who feel that they have an interest in the laptop programs. A central program that attempts to deal with all the conflicting and changing demands will find itself being forever amended. There is a real danger that the programs will become "over developed" and unstable.

1.5 An uncertain future

The only certainty is uncertainty. The demands of new surveys are constantly changing - the need for multi-instrument surveys, for example, was not predicted until it became necessary.

A difficulty that OPCS had to face when we were redesigning our laptop systems was the need to be able to handle Blaise III as well as Blaise 2, preferably concurrently. At the time, we knew very little about Blaise III, so had to design a system that would deal effortlessly with a comparatively unknown CAPI package.

2. Why Object Orientation is a good approach

2.1 Mapping onto the problem domain

Object orientation enables us to produce solutions that mirror the problems we are trying to solve. The traditional database approach will send data out to the interviewer in a number of interviewer workload-sized databases, and receive it back as a different number of "completed day" databases. With an object oriented approach, one can define each interview as an object. The task then becomes to send out a number of interviews, and receive back the same number of interviews.

One is able to implement a system that closely mirrors intuitive thought processes. If an object is exactly analogous to an old paper questionnaire, it can be handled in the same way.

The interviewers will more easily understand the processes that are occurring.

Object orientation also makes it very much easier to develop event driven systems that respond to the interviewers' demands rather than leading them through menu hierarchies.

2.2 Resilience

In a well designed object oriented system, all the interview objects will be fully encapsulated. They will be totally independent of each other. This means that new surveys can be implemented on the field force's laptops without the danger that they will inadvertently interfere with existing surveys.

By breaking the data down into the smallest components that can be handled independently (the interview), the system is more resistant to faults. For example, if the interviewer switches the laptop off in mid-interview, this could corrupt the dataset. By ensuring the dataset corresponds to only one interview, only that interview is lost, not the entire month's workload for the interviewer.

2.3 Flexibility

By encapsulating everything inside objects, the laptop programs that perform common handling operations on the data are insulated from the nature of the data itself. This means that they can handle very different types of questionnaire without any rewriting.

Thus the laptop handling programs don't care if a survey needs one Blaise instrument or two, or even if extra non-Blaise programs are included. They will handle Blaise III instruments as well as Blaise 2 (and at the same time). They will also handle any other CAPI package, so long as they can be persuaded to load up and enter a specified case from the command line, and can maintain the public interface.

The contents of an object don't have to constitute a survey interview. The same case handling system can effortlessly deal with:

- software installation,
- housekeeping tasks,
- laptop audit,
- pay claims,
- email.

3. Designing an Object Oriented Case Management System

3.1 The level of abstraction - what are the objects?

We have seen that we handle the data in a number of different degrees of aggregation:

- survey month / stage,
- interviewer workload,
- interviewer daily completion.

However, in line with good object oriented design practice, we want to keep the object small. I have been referring to an object as representing "an interview". This enables each interview to be handled separately from all the others in the system. One can send a number of interview objects to the interviewer, representing the workload. The interviewer acts on them and sends them back as separate interview objects.

The problem with that approach, is what to do when we don't know how many interviews we expect back from the interviewer. Examples of this are:

- Exit surveys, where we assign the interviewer to a shift, and interview every *n*th person.
- Household based surveys where the sample point is an address that may contain multiple households.

We have a choice - do we make the object match the sample point, or the interview?

If the object matches the sample point, then the common systems (the Object Handling Systems) will fully account for the sample, and the survey-specific systems (the Interview Management Systems) will roster within the sample point. All interviews for the sample point will be stored and handled within the same object. In the exit survey example, the object would represent an interviewer shift; in the household example, the object is a sampled address.

If the object represents an achieved interview, we need a mechanism to allow for the fact that we don't know how many objects to send to the interviewer. One solution would be to send a large number of empty objects, and ask the interviewer to return unused ones. A better approach would be to generate extra objects as required in the field. The interviewer is sent an object that contains a full class template for the interview-based object, together with methods for rostering within the sample point. On choosing to perform an interview, the interviewer will activate this object, which will then create an instance of the interview class for this survey. The interview will be performed within this new object.

The OPCS approach is usually to assign the sample point to an object. So for our household-based surveys where the sample point is a Post Office delivery point (address), the object represents an address. The survey-specific systems account for households within the address. This means that all households at an address must be fully accounted for and completed before any can be returned to the office. This represents good field practice, and reflects the rules applied to paper-based surveys, but wouldn't be appropriate for all circumstances.

For some of our systems (eg handling of interviewer pay claims), we will spawn new objects in the field from a class template.

So the object contains the data that make up a sample point, plus attributes that apply to the data, plus methods that enable an interview to take place.

3.2 Method of encapsulation

There is a need to hold within the object:

- interface to public methods,
- public attributes,
- the questionnaire data itself (private).

Most object oriented development systems will provide tools to create classes and objects and will provide the public interface, and private data. The difficulty is that we actually care about the way the data are stored; we are dealing with Blaise data, and need to store the data in *.D00, *.DRF, *.KEY etc. files. Internal data storage formats would be fine if we were writing an entire system from scratch, but what we are trying to do is put an object wrapper around Blaise interviews.

The encapsulation tool therefore needs to be able to handle DOS files, either directly or indirectly. One way of dealing with this would be to hold a pointer within the object, and hold the data itself separately. This however violates one of the first principles, that an object should be entirely self sufficient.

OPCS adopted a file-based approach both for the private data, and for the public interface. The data are kept as native Blaise files. The public attributes are kept in various attribute files, and the interview method is a batch file. All the files for an object are held together within a PKZip archive file. Thus one object is fully encapsulated within one Zip file.

This use of PKZip to encapsulate the objects gives platform and language independence, and allows great flexibility in the nature of the stored information and methods. The price for this is a certain processing overhead when loading an object, or especially, when scanning a large number of objects.

A file-based encapsulation tool, where each object is a file, enables standard DOS file and directory management features to be used to handle the data. An added bonus with PKZip is the inclusion of encryption within the package.

3.3 Classes and inheritance - standard classes and objects

The Casebook laptop systems, as designed by OPCS, are very flexible in their use. An object can represent a variety of different entities. OPCS have developed a number of different standard classes, from which objects can be generated.

The root class, from which all others inherit, represents the minimum necessary for our field-based systems to handle the object. The root object contains just one attribute and one method:

- Attribute - INFOLINE. This is a caption to appear on the laptop.
- Method - METHOD. The default method is a batch file, called METHOD.BAT.

Inherited from this root class, are three main classes:

- Basic survey class. This will contain attributes controlling transmission status of an object, and encryption information.
- Questionnaire installation class. The Blaise programs are not carried within the objects, but called from the interview method. The questionnaire installation objects will install a copy of the questionnaire program onto the hard disk. This will contain additional attributes declaring the amount of free disk space needed for a successful installation.
- Interviewer workload installation class. When sending stints of work to interviewers by disk, all interview objects will be sent together wrapped up in an installation object. The method will install the objects to the correct place. An additional attribute is one that prevents the installation being run twice (thus overwriting started interviews with unstarted ones).

The basic survey class itself gives rise to two other main classes:

- Multi-instrument survey class. This contains additional details in the method to handle multi-instrument rostering.
- Training case class. This contains attributes that enable the interviewer to delete the object when no longer needed.

An individual survey going into the field will generate a set of identical questionnaire installation objects. It will define its own specific survey class (inherited either from the base survey class or the multi-instrument survey class), and will instantiate this as a number of interview objects, each representing one sample point. The survey will also generate a number of interviewer workload installation objects, each containing one interviewer's allocation of interview objects.

3.4 Polymorphism - defining the common public services and attributes

As described above, the basic class contains only one service and one attribute. In fact, there is only one public service defined for all the classes in use at OPCS; this is the default method, which is implemented as the batch file METHOD.BAT within every zipped object. The actions of this method will vary from class to class (this is what is meant by polymorphism in the object oriented context). For interview objects, the default method will conduct an interview (though the internal means of achieving this will differ from survey to survey). For installation objects, it will copy a set of files from one place to another. Typically, the batch file will call on routines written in Clipper or Blaise/Manipula, and that have been previously installed in the appropriate survey directory on the laptop.

The caption attribute, called INFOLINE is a line of Ascii text held in the text file INFOLINE.TXT. The object handling systems have no interest in the layout of the text line - the laptop will merely order the captions alphabetically and present them to the interviewer for action. In practice, the lines are typically

made up of a three character survey identifier, followed by a stage or month, then a serial number. Additional identifying information follows (usually taken from a comments question on the Blaise questionnaire), then completion and outcome information. All this information is of use to the interviewer, and being ordered in this manner ensures that sample points for interview are sorted by serial number within stage within survey. However it was decided early on that the common systems would have no interest in the contents of the caption; there is freedom for the class designer to include whatever information, in whatever order is appropriate.

Similarly it was early decided that there was no need for the common systems to "know" what the survey, stage or serial number are for an individual object (it may not even refer to a survey at all). Hence there are no common attributes currently defined for these items of information.

Information on readiness for transmission is held within another text file, TRANSTAT.TXT. This contains one line, which holds two items of information:

- The state of the case, either clean or dirty. This is indicated with a "C" or a "D" single character at the start of the record.
- If dirty, the reason. This is a short text string to be displayed to the interviewer if an attempt is made to transmit the case before it is ready. Examples are "No work done on this case" or "DONECODE not coded YES for all households"

Different survey classes may use different techniques for assigning values to TRANSTAT. Typically, they will refer to one or more questionnaire variables (checking all cases at the sample point), or they may look at the Blaise cleanliness status.

The two attribute files mentioned, INFOLINE.TXT and TRANSTAT.TXT are held in separate one-line files for ease of maintenance, particularly as they will need to be updated during interviewing.

All the other public attributes are kept within a single properties file, CASEBOOK.DAT. This consists of a number of lines, in arbitrary order, of the form:

attribute=value

The file is not mandatory, and attributes will hold a default value if they are not mentioned. There are many different object properties that can be set within this file - for example whether the object can be deleted by the interviewer, disk space required, any flag files that must exist before running (eg to check that a questionnaire program has been installed). In addition, extra information is stored to help manage data encryption.

4. An Object Oriented Case Management System in real life - OPCS' experience

4.1 How OPCS designed and built Casebook and InterCom

System design

Our initial need covered the redesign of our laptop and communications systems so that they could:

- send and receive data to and from the interviewers by modem,
- cope with the increasing complexity and diversity of our surveys,
- cope with the increasing number of surveys, and survey teams with an interest in the laptop.

In addition, we had the brief to ensure that the laptop system would not need radical redesign when Blaise III was implemented, and should preferably run both versions of Blaise concurrently.

The solution was a three-part system, *InterCom*. This is an object handling system to cover laptop and office systems.

- The *Scatter and Gather* system provides the "post office" facilities to collect and distribute data via the interviewer post boxes.
- The *2Way* facility handles the transmission and reception of objects.
- *Casebook* is the laptop system that gives the interviewer full handling facilities for the interview objects.

Casebook was designed with the two main aims, for the interviewer, of visibility and control. In the older laptop systems, the computer decided which cases were to go back to the office. The interviewer had to guess which had been transmitted, and which had not (this of course was more difficult for multi instrument surveys). With Casebook, we put the interviewer back in charge. The interviewer selects cases for transmission; they are checked for cleanliness, and if they are ready to go, they are moved to the laptop's out-tray. The interviewer can always see exactly what is happening.

Implementation tools

Although we were designing an object oriented system, we decided early on not to use an object oriented development language.

The key point is that we were designing and implementing an object oriented system, not a series of object oriented programs.

As discussed above, we chose a language-independent encapsulation tool, PKZip. Having defined the standard classes and their attributes and methods, the system itself is built from three main programs reflecting the components of InterCom described above. Thus the laptop system is fully encompassed within the Casebook program. Casebook was written largely as an event-driven program, but was not written using object-oriented development tools, nor is its internal construction object-oriented.

Designing and implementing the object handling system is only half the work. The survey classes need detailed design and implementation work. For these, Manipula is used for extracting management information from the Blaise interview data; external Clipper programs are used for rostering within the sample point.

4.2 The split between common object handling systems, and survey specific systems

Having encapsulated the interviews and defined what constitutes the public interface, and what is private, we are able to define the data-independent tasks that will be handled by the common parts of the case management system.

The common object handling systems are responsible for:

- storage and deletion of objects,
- backup,
- security,
- transmission,

as well as presenting the object captions to the interviewer and initiating the interview method. The common systems only have access to the public interface of the objects. They are not allowed to look at the private data.

The survey specific interview systems (called by the interview method) are responsible for:

- rostering within the sample point,
- running the interview,
- any internal integrity checking,
- maintenance of the public attributes.

4.3 Casebook and InterCom in action.

OPCS introduced the new laptop systems for all our General Field Force CAPI surveys in April 1994. This coincided with the conversion of several of our remaining PAPI surveys to CAI.

As the systems are a lot more resilient and controlled than the previous ones, the number of cases that require additional checks to trace has dropped to zero.

We now find that we can rely on receiving cases back from the field very quickly. Because the interviews are fully encapsulated, and never broken into constituent records in the field, the data are very much more reliable; the number of structural errors being reported by our checking programs has now dropped to zero.

The object oriented systems work reliably and easily. The only remaining difficulties are at the interfaces with more traditional systems. Creating the interview objects from a sample file is slower than desirable. However this is more than compensated for by the large increases in speed, reliability and flexibility of the new systems.

The flexibility of the objects has shown itself many times - we can send programs and non-standard data to our interviewers, and handle them totally within our standard case management system.

4.4 Where we go from here

The new systems are working well in the field, but development has not stopped. There are several areas for further work.

The area that has the most immediate potential is extending the system to cover office-based work, particularly editing. The editor's task is essentially similar to that of the interviewer - to receive cases, work on them and pass them on. Again, the natural system design should be around the "completed questionnaire" rather than around databases of record types. OPCS are developing a system, CASEFLOW, that will take survey objects from the field and route them through to the survey editing teams.

For CASEFLOW to fully replace the existing systems, we need to be able to extract appropriate management information from the interview objects as they pass through. An extension of the public interface to cover common information, such as standardised outcome codes, would make the systems more flexible and reliable.

The systems already implemented have been firmly based on the work of the face-to-face interviewer. It would be advantageous to extend the system to cover telephone interviewing, but call scheduling usually relies on selection of cases from a single large dataset. One solution would be to extend the objects' public attributes to cover the information required, and to extract these attributes to form a call-scheduling dataset that contained pointers to the interview objects. This could enable Blaise call scheduling to manage the interviewer interface of the object-based system. Aside from the advantages already experienced by the face-to-face interviewers, it would enable interviews from a variety of surveys and stages to be scheduled together within the same day-batch.

The move towards Windows will mean new interfaces to all our systems. However it seems that the basic system design and object encapsulation method will continue to work. This is important, as it will enable us to migrate slowly, and to enable Windows-equipped interviewers to use the same Windows interface to access older DOS-based questionnaires alongside their Windows ones. As more surveys and interviewers move to Windows, we may find that attributes and methods are more appropriately stored in DLLs.

5. Conclusion

The object oriented systems we have built in OPCS spring from a very simple set of ideas. They have been implemented using robust, but "low-tech" design techniques and tools. However they have proved themselves extremely powerful. They have already delivered real savings in cost, reliability, flexibility and speed. These benefits are expected to increase as the system is extended to cover telephone interviewing and office-based systems.

Performance and Design

Jean-Pierre Kent, Statistics Netherlands

1. Introduction

The Data Entry Program of the Blaise III package has features specifically designed to ensure short response times even for very large and complex data models. This article describes the performance aspects of an interviewing program. It presents the main lines of the mechanisms designed to optimize response times and explains some of the basic rules that a data model designer should follow to allow these mechanisms to work efficiently.

Writing this article has contributed to identifying some problems of Blaise III 1.0 that had not been caught by beta testers or by production users. These problems have been solved in Blaise III 1.1. There is no guarantee that the facts presented in this article work as stated in any release of version 1.0.

2. Response times in interviews

For the Data Entry Program (DEP), response time is the time needed to react to each of the user's input or edit actions. The system may need this time to calculate a new route, impute fields or create new question and answer texts. In this paper I assume that the DEP is being used in interviewing mode (dynamic routing is turned on).

2.1 Data Consistency

The DEP of Blaise is built in such a way as to guarantee the consistency of the values whatever the interviewer does and in whatever order. Whenever values previously entered are modified under dynamic checking, the state of the whole form is recomputed to reflect the new situation.

2.2 Time in the data entry / editing process

The Blaise Data entry program is asynchronous. In simple words, this means that the time of occurrence of some of the actions defined in the RULES is unpredictable. For instance if you write

```
A := 5
B := A
```

then you know that B will always be 5, because B:=A will always be executed after A:=5. If however you write

```
A.ASK
B.ASK
```

there is no way to predict the order in which the values of A and B will be supplied or corrected. Take a look at the following example:

```
Age
IF Age < 15 THEN
   School
ELSE
   Job
ENDIF
```

School and Job could be simple yes/no questions, but they could also be blocks with lots of questions and sub-questions. We can imagine a situation in which the interviewer has to correct the value of Age after filling in the answers of some of the questions about School. This can cause some of the entered values to become irrelevant, and lead the system to backtrack into the other branch of the route.

2.3 What should be performed?

We want to be sure that whenever a value is changed, all the RULES that depend either directly or indirectly on this value are executed. It is, however, not a trivial task to determine the dependency relationships between values and rules. Due to the potential complexity of conditions and loops nested in one another, this information changes during the interview and cannot be computed at preparation time. Unless some sort of run-time mechanism takes care of tracking down what has to be done, the only way to ensure nothing relevant is skipped is to perform all the rules after every modification of the contents of the form!

This, in fact, is what the older versions of Blaise do: every time the cursor leaves a field during a CAPI-mode interview, the whole model is recomputed, and the interviewer cannot start editing another field before this work has been done.

2.4 Data model complexity

When survey designers had to take the limited memory of XT computers into account, they were forced to define small models whose rules could be executed very fast. As computer memories grew, designers tended to build larger and larger models, until they reached the internal limits of Blaise 2. It is in this context that speed became a relevant aspect for Blaise.

One usually assumes that performance is a hardware issue. A program that is slow on today's machines will be fast on tomorrow's, so speed is only a question of technology.

This assumption, however, does not always hold. Execution time depends not only on hardware, but also on the complexity of the task at hand. For a Blaise program, this is the complexity of the rules (in earlier versions: the combined complexity of the route and of the check). This, in turn, is a function of the size of the data model.

2.5 Exponential growth

If the complexity of the rules were a linear function of the size of the model, there would hardly be a problem: as memories become larger, processors also become faster, so performance of Blaise models could be expected to remain stable. The number of consistency rules, however, tends to grow faster than the number of fields.

Consistency rules are usually a combination of two or more fields; so one can theoretically write one consistency rule for every possible combination of two or more fields. Although such an extreme case will never be encountered in real-life data models, it remains true that the number of rules tends to grow exponentially as a function of the number of fields.

3. Lazy checking

One of the prerequisits of the specification of Blaise III being the possibility of using all available memory (up to 16mb), it was clear that something had to be done to prevent an explosion of processing times. The system had to be designed to reduce computing activity to a minimum. It had to be capable of detecting useless work and refusing to execute it. This is what we call lazy checking.

3.1 Level of processing

For many reasons the block was seen as the best unit of administration for lazy checking. By block in this article we mean a block field: this is a field (defined in a FIELDS or AUXFIELDS paragraph) whose type is a block.

In this section we will give an overview of the mechanisms that make lazy checking possible. This information is crucial for understanding how data model design can contribute to the performance of the Data Entry Program.

3.2 A block functions independently

While the rules of a data model are being performed, each instance of a block plays its part independently of other blocks. If you know about object oriented techniques, see a block as an object exchanging messages with other objects. If you don't, just see a data model as a play in which blocks are the actors. The actors talk to each other, ask questions, react to questions by supplying answers or taking actions. Note that the data model itself is a block. So whatever is said here of blocks is also true of the data model.

It is important not to see the rules of a block as a procedure being called by the rules of another block: it is the sole responsibility of the block itself to decide whether its rules should be performed, basing its decision on the information available to it.

3.3 On or off the route?

One of the important concepts for understanding the lazy checking mechanism is that of "being on the route". The conditional rules (rules specified within an IF ... ENDIF construction) allow the definition of multiple potential routes through the questionnaire, one of which will become active at interview time, when values can be computed for the conditions of the IF rules and for the bounds of the FOR rules. Any field whose ASK, SHOW or KEEP method is part of the active route is said to be on the route. This notion is very important in the present context, because a block only talks to those of its sub-blocks that are on the route. The others are systematically ignored.

Although routing is primarily an interviewing concept, with dynamic routing turned on, this mechanism is also active while in data editing mode, when dynamic routing is turned off.

3.4 Detecting changes

Although communication between blocks takes place mainly while the rules are being executed, blocks are also active in another phase, when values are entered or edited.

It is the responsibility of every block to keep track of any change taking place in one or more of its fields. Whenever such a change takes place, this activates a process in which the block marks itself as changed, and communicates this information to its owner. This will cause its owner to mark itself as changed and pass the message further up.

This goes on until the message reaches the data model. During an interview, the data model's owner is the DEP. The DEP will react to changes by asking the data model to perform its rules.

3.5 Selective execution of rules

The rules section is executed as one compound rule, computing the values for conditions and activating the correct branches of conditional checks and routes. This causes consistency rules to be evaluated and error messages to be generated if necessary. It is through the execution of its rules that a block decides which of its sub-blocks are on the route. During this process there is no difference between the ASK, SHOW and KEEP methods; all they do here is to say: "this field is on the route".

A block performing its rules will send every one of its sub-blocks that are on the route a message telling it that its turn has come. For clarity we will first concentrate on the simplest case, considering blocks with no parameters.

The receiving block can react to its owner's message in two different ways: it can either trigger its own rules or ignore the message. The rules will only be triggered if the block is marked as changed.

By ignoring its owner's message, a block not only ensures that its own rules are skipped: it also avoids any form of communication with its own sub-blocks, so that all blocks owned directly or indirectly by it will be left out entirely.

After executing its rules, the main block (the data model) unmarks itself as changed and tells all of its sub-blocks to do the same. This action will be repeated recursively by all the blocks that are marked as changed, allowing every block to ignore its owner's message next time the rule are executed.

A block can be marked as changed if the interviewer or respondent changes any of its field values. But this can also be the effect of an imputation carried out by another block within the same check of the data model. For example the statement

```
Household.HHSize := Household.HHSize + 1
```

will mark the block Household as changed.

3.6 Lazy checking in the example model

To illustrate the lazy checking mechanism, let us have a look at the following data model:

```
DATAMODEL Evaluation
   BLOCK BComment
      FIELDS SayIt: STRING[80]
      RULES SayIt.ASK
    ENDBLOCK

    BLOCK BReadability
      FIELDS Level: 1..10
      RULES Level.ASK
    ENDBLOCK
  FIELDS
    Readability: BReadability
    Comment: BComment
  RULES
    Comment.ASK
    Readability.ASK
  END
```

This trivial model has two blocks and a two-level hierarchy, which is sufficient to illustrate the issues at hand.

When a new record is created, all blocks are initially marked as being off the route. This changes as soon as the rules of the data model are executed. The first rule (Readability.ASK) does the following: it tells Readability that it is on the route, which causes it to mark itself as changed, because its route status has changed; then control is passed to Readability, which triggers its own rules. The effect of Readability's rules is to put the field Level on the route. Now Readability passes control back to Evaluation. The next rule in Evaluation is Comment.ASK. Comment and SayIt will go through the same process as Readability and Level.

What happens if the interviewer enters a value for Readability.Level? The Readability block will enter the new value for Level and mark this field as changed; then it will mark itself as changed and pass this information up to the Evaluation block. Evaluation will mark itself as changed and because this is the main level the DEP will detect this and will ask Evaluation to perform its rules. This time, however no change has taken place within the block Comment, so when its time has come Comment decides to skip its rules. In a similar way the rules of Readable will be skipped when the interviewer enters a value for Comment.SayIt.

3.7 Taking advantage of the hierarchic structure

If a data model has a hierarchic structure, which is usually the case for interviews designed with Blaise, the mechanism described above is very effective in reducing computations. A change will cause work to be done in just one branch of the tree: in the block in which a change has taken place, and in the owners of this block. The number of blocks triggering their rules is thus a function of the depth of the tree and not of the overall size of the data model. If the model is made of independent blocks, this should compensate for the potential combinatoric explosion of the number of checks.

4. Communication through parameters

The mechanism sketched above assumes that block ownership is the only form of dependency between blocks. In most models, however, it is necessary to establish dependencies of another nature.

4.1 Block dependencies

The syntax of Blaise allows the designer to make use of values residing anywhere in the data structure. Whenever this possibility is used in the rules of a given block, this block becomes dependent on all the blocks it draws upon.

If for instance we write the following rule in a block called BChild:

```
Age < Father.Age
```

then all instances of BChild become dependent on Father.

The syntax of Blaise also allows a block to compute a value for a field of another block. For example:

```
Father.Age := Age + Difference
```

Although the syntax allows information interchange between blocks from all parts of the model, the only lines of communication supported by the system are vertical. Now we will explain how information is

passed over ownership lines from the block in which it resides or is computed to the block in which it has to be used or stored.

4.2 The vertical communication channels

We have seen that whenever a block receives control from its owner, it will only trigger its rules if it has detected some change in the values that it manages. If, however, it accesses information from other blocks, this mechanism will not suffice. If any of the values it accesses has changed this can lead to a different result, so the rules have to be recomputed.

In order to be able to detect such a change, a block has to keep track of imported values over time. The fields managing values accessed from other blocks are called import parameters. The designer is free to define import parameters himself, and it is often useful to do so. However it is always possible to let the system generate those parameters internally.

The passing of import parameters plays an important part in inter-block communication. Before passing control to one of its sub-blocks, the owner passes it the values for all its import parameters. The receiving block compares each parameter with its previous value before storing it, and marks itself as changed if one of the values is different. This ensures that any change in one or more of the values accessed by a block will cause the rules of that block to be executed.

4.3 Example of a generated import parameter

Let us return to the model outlined under 3.6 and expand the rules of Bcomment with an unconditional check rule:

```
RULES {BComment}
  Sayit.ASK
  IF Sayit = "Good" THEN
     Readability.Level > 5
  ENDIF
ENDBLOCK
```

Now BComment has become dependent on a value that it does not own. This has consequences both for the preparation of the data model and for inter-block communication at run time. At preparation time, the system will create an import parameter for block BComment. At run time Evaluation will extract the value of Readability.Level and pass it to Comment every time the rule Comment.ASK is executed, causing Comment to mark itself as changed every time the value it receives has changed from the previous run.

The effect of adding that rule, in terms of execution complexity, is that while the rules of Readability are executed only when Readability.Level changes, those of Comment are triggered every time any of the two fields of the model changes.

4.4 Imputing an extraneous field

What happens if a block, instead of accessing an extraneous field, imputes it a value? The block will be expected to return the same value every time, whether it executes its rules or not. In order to be able to impute values without computing them, a block needs fields to keep track of all imputed values. These fields are called export parameters.

4.5 Example of a user-defined export parameter

Now, instead of using the value of Readable.Level in block BComment, let us admit we want to modify the value of an extraneous field passed through a parameter. To do this, we will add a PARAMETERS section in the block, and add a rule performing the imputation:

```
BLOCK BComment
    PARAMETERS
        EXPORT LevelParam: INTEGER
    FIELDS
        Sayit: STRING[80]
    RULES
        Sayit.ASK
        IF Sayit = "Good" THEN
            LevelParam := 8
        ENDIF
ENDBLOCK
```

Now we are left with the task of telling the main block Evaluation that Readability.Level is the destination field for the value computed by Comment for its parameter. So the rule Comment.ASK becomes:

```
Comment.ASK (Readability.Level)
```

Let us now see how inter-block communication takes place in this new situation. Whenever the interviewer enters a value for Comment.Sayit, Comment will be marked as changed and so will Evaluation. This will cause the rules of Evaluation and BComment to be triggered, computing a new value for Comment.LevelParam. This value will be passed back to Evaluate, which will store it in Readability.Level. If this value differs from the value previously stored in that field, Readability will mark itself as changed, which will cause it to perform its rules when its turn comes.

4.6 Timing aspects

Note that the mechanism outlined in the previous paragraph heavily relies on the order specified in the RULES of Evaluate. If we change this order:

```
Readability.ASK
Comment.ASK
```

then a change in Comment will never trigger the rules of Readability: by the time the imputation of Readability.Level takes place, Readability has already been called, at a time at which it was not marked as changed. When Readability is called, it skips its rules; it will not get another chance after the imputation takes place.

This feature is potentially dangerous: a block receives a value and is denied a chance to check it and compute the consequences. However, if you know that the imputed values are always correct, and that neither the routing of the block nor the values it manages depend on them, you can take advantage of this and put blocks on the route in an order that minimalizes checking activity due to imputations.

4.7 Import or export?

Although import and export parameters are both used by the system to keep track of values over time, their function is totally different. Imported values have an influence on the RULES, and a change in imported values contributes to triggering them. Setting and comparing the values of IMPORT PARAMETERS is the first step in the communication taking place between a block and its owner. EXPORT PARAMETERS have no such influence. They are used to enable a block to return the correct values, whether or not it decides to trigger its rules. Returning the values of EXPORT PARAMETERS is the last step in inter-block communication.

It is often necessary to access the value of a field before modifying it. In such a case we need a TRANSIT parameter. TRANSIT PARAMETERS perform both the functions of IMPORT and EXPORT PARAMETERS. This is why a TRANSIT parameter internally needs two fields: one to keep track of the imported value in order to detect changes, and one to keep hold of the exported value. TRANSIT PARAMETERS perform both functions of detecting changes in accessed values and returning imputed values. So they are more costly, both in terms of memory use and of execution time, than IMPORT and EXPORT PARAMETERS.

4.8 Example of a generated TRANSIT parameter

Let us modify again the definition of block BComment:

```
BLOCK BComment
    FIELDS Sayit: STRING [80]
    RULES
        SayIt.ASK
        IF Sayit = 'Good' AND Readability.Level < 5 THEN
            Readability.Level := 5
        ENDIF
ENDBLOCK
```

This will make sure that whenever SayIt has the value 'Good'. Level will be at least 5.

At preparation time the system will add a transit parameter to the data structure of BComment. At run time Evaluation will extract the value of Readability.Level and pass it to Comment, ask for the return value and store it back in Readability.level. Both Comment and Readability will compare the values supplied by their owner and mark themselves as changed if necessary.

4.9 Export parameters are always user-defined

Wherever the designer would define export parameters, the Blaise system will always create transit parameters. This is because it can never be sure that the value of the field is not used before it is computed. So it is always wise to define export parameters yourself: this will spare memory (the block will not need a field to keep track of the value of the parameter before it is called), communication time (the block will not receive the value of the parameter from its owner), and execution time (a change in the value of the parameter will not cause the block to trigger its rules).

4.10 When does a block need a parameter?

In the cases sketched above, Evaluation transfers the value of Readability.Level to Comment through Comment's import or transit parameter. But where does Evaluation get it from? Evaluation can access it directly, because it owns it. The rule is: a block has direct access to its own (aux)fields and those of all the blocks that it owns either directly or indirectly. It can access them both to read their values and to change them. The only thing it cannot do with fields of its sub-blocks is put them on the route. But whenever a block needs to access a field that it does not own, it has to count on its direct owner to pass it as a parameter.

5. The impact of arrays

Although an array can be seen as a list of fields of one given type, in which each element is treated separately, they often behave as a whole, and as such they can have a heavy influence on performance.

5.1 Arrays and routing

The elements of an array can be routed in two different ways: either separately, in direct route rules:

```
Person [1].ASK
Person [2].SHOW
Person [3].KEEP
```

or as a group, within a loop rule:

```
FOR i := 1 TO 3 DO
   Person [i].ASK
ENDDO
```

These two examples use constants, either for the indexes in the direct route, or for the bounds of the loop. This results in elements 1 to 3 of the array being routed unconditionally: whatever happens during the interview, all three elements are always on the route.

In real-world models, however, arrays are usually larger, and a limited number of elements are expected to be on the route for each interview. This can be expressed by using FIELDS or AUXFIELDS for the bounds of the loop:

```
FOR i := FirstPerson TO LastPerson DO
     Person [i].ASK
ENDDO
```

At preparation time the system will define the layout to display all Person fields having indexes ranging from the lowest possible value of FirstPerson to the highest possible value of LastPerson. At run time, only those comprized between the actual values of FirstPerson and LastPerson will be on the route.

5.2 Arrays and parameters

Parameters and arrays interact in two ways to affect performance: first, if a block has parameters and is used for the definition of an array, the parameters will be replicated for all elements of the array; second, accessing a field of a block that is part of an array can cause Blaise to generate too many parameters.

5.2a: Parameters in an array of blocks

The examples in the rest of this article will all be taken from the data model in the appendices. Let us take a look at the question text of the field Name in the block BPerson. It refers to the local variable i, which is part of the table THouseHold. This causes Blaise to generate a parameter for i in the block BPerson. This block, however, has 10 instances: Person[1] to Person[10]. So the value of i will influence the behaviour of all ten blocks of the Person array.

Although the value of i changes all the time, this change is not visible for the individual Person blocks: whenever i has, for instance, value 2, HouseHold will communicate with Person [2]. So i is always 2 for this block. The block, however, does have to receive this value, compare it with the old value, and store it, and this can take time if there are any INPUT and TRANSIT parameters, and if the array is large.

5.2b: accessing a field in an array of blocks

The other point is not trivial: if a block is part of an array, its fields can only be accessed through and indexed reference to the array. If the index is a constant, Blaise knows at preparation time which element to access, and it can generate one single parameter. If, however, a field or an expression is used for the index, neither Blaise, at preparation time, nor the owner of the accessing block, at interview time, can determine which field is relevant: this forces Blaise to generate as any parameter fields as there are elements in the array of blocks.

Take a look at the question texts in the block BEducation of the example in the appendix. They use ^HouseHold.Person[i].Name to display the name of the person being interviewed. When Blaise is preparing this data model, there is not enough information to ensure that the owner of instances of BEducation can make the correct choice. The only solution to this dilemma is to generate parameter entries for all the names in the Person array. So BEducation, as well as BActivity, ends up with 10 Name parameters.

The effects of 5.2a and 5.2b can be compounded, if a block that is part of an array accesses fields from an array of locks. In the example, the series of 10 Name parameters is present 10 times in Education and 10 times in Activity. The owner of these two arrays, Info, does not own those Name fields, so it also has to receive their values from its owner, the main block Survey. This gives us a total of 10 * (10 + 10 + 1) = 210 parameters for passing the Name of the respondent to the question texts.

If you want to see how many parameters are generated in a given data model, you can ask the Structure Viewer to show them. First you need to modify the structure viewer options through the menu item OptionsöStructure Viewer: then go to the Structure Pane and activate the entry for Internal Parameters. This will ensure that the viewer will integrate this type of information in the Structure Pane. The tree structure in Appendix 2 shows the data structure of the example in Appendix 1, including internal parameters.

6. How design affects performance

The preceding sections have presented the lazy checking mechanism, outlining its interaction with the three main features affecting its effectivity: routing, the passing of parameters, and arrays. We will now build upon this information to show what has to be done to reduce computing activity to a minimum.

6.1 Interaction of the three features

The mechanisms of routing and parametrization have been designed in such a way as to ensure optimalization of lazy checking. The interaction of these two features do not lead to problems if you take a few elementary facts into account. We will first concentrate on these simple rules, before examining the impact of arrays.

6.2 Routing and parameters

There are two basic rules: try to minimize block dependencies as much as you can, and build truly hierarchical models.

6.2a: Keeping blocks as independent as possible

Whenever a block can compute a value by itself, it should not count on other blocks to provide it. It is, for instance, not a good idea to store the value of STARTTIME in a field of the datamodel, and access this field from all other blocks. The execution of the function wherever it is needed is more efficient than its value being passed through parameters. Although this value never changes, and will never be the cause of unnecessary execution of the rules, it has to be compared before the system can decide that it has not changed.

It once occurred that a developer submitted a model with very poor performance, asking us what to do about it. We found out that the question text of every single field was defined as "^QText", and RULES sections imputed the value of QText for every field on the route. QText was a local variable of the main block. No wonder this was slow: every block was receiving the question text of the latest routed field, and returning the question text of its own last routed field. The blocks were wasting time exchanging irrelevant information!

Before accessing a value in another block, always ask yourself if it can be computed locally. This should be possible if the value is not dependent on information stored in other blocks.

Block dependencies are often due to a computation being performed at a level that is too low. Let us return to the example given under 4.8: a value for Readability.Level is computed from within the block Comment. This causes the generation of a TRANSIT parameter. This, in fact, is not necessary: the computation can take place in the common owner of the two blocks involved. What we must do is restore the original rules of BComment, and place the computation in the RULES of the main block, which then become:

```
RULES { Evaluation }
  Comment.ASK
  IF Comment.SayIt = Good AND Readability.Level < 5 THEN
    Readability.Level := 5
  ENDIF
  Readability.ASK
END
```

Now take a look at the Survey example in the Appendix. The RULES THouseHold have two checks to verify that no more than one person is declared as Head or Spouse. This is the right place to perform such checks. If we had placed them within the block BPerson, the incrementation of the variables HeadCount and SpouseCount would have caused the creation of 2 * 10 = 20 TRANSIT PARAMETERS to access them.

6.2b: building truly hierarchical models

Always try to find a good balance between the width and the depth of the hierarchical tree. If you have 1000 instances of block, it is not a good idea to declare an array indexed from 1 to 1000 like in:

```
BlockInstance: ARRAY [1..1000] OF BlockDef,
```

because the owner will be communicating with all 1000 instances, which is not efficient. It is better to nest arrays in one another, like so:

```
BLOCK BLevel1
    BLOCK BLevel2
        BLOCK BlockDef
        .
        .
        ENDBLOCK
    FIELDS
        Instance: ARRAY [1..10] OF BlockDef
    ENDBLOCK
FIELDS
    Instance: ARRAY [1..10] of BLevel2
ENDBLOCK

FIELDS
    BlockInstance: ARRAY [1..10] of BLevel1
```

Now each block owns only 10 blocks. In order to reach an instance of BlockDef, communication has to take place at three levels, so in total 30 blocks will receive a message, instead of 1000.

6.3 Routing and arrays

Arrays are usually defined for the storage of information about a variable number of instances. Very few records will need them all. To make sure only relevant instances are on the route, you need to be careful when choosing the type of the fields with which the bounds are defined.

The following example will illustrate the importance of this:

```
FIELDS
    Count "Number of persons to interview": 1..3
    Person: ARRAY [1..10] of BPerson
LOCALS
    i: INTEGER
RULES
    Count
    FOR i := 1 TO Count DO
        Person [i]
    ENDDO
ENDBLOCK
```

This construction will cause instances 1 to 3 to be displayed, because 3 is the maximum possible value of Count. It will also cause instances 1 to 3 to be routed only if their respective indexes are smaller or equal to the run-time value of Count.

But what happens to instances 4 to 10? For reasons that will not be discussed here, Blaise generates an unconditional KEEP rule for every field not explicitely routed. So instances 4 to 10 are always on the route. This was probably not the effect aimed at.

This can be corrected either by defining Count in the range 1..10, or by defining the index range of the array as 1..3.

Blaise wil not force you to harmonize the ranges of the loop bounds with those of the array index: you might want to process different chunks of the same array under different conditions and in different loops. So use of the correct ranges is left to the responsibility of the designer.

6.4 Extracting parameters from arrays

Let us try to minimize the number of parameters generated for HouseHold.Person[i].Name in the question texts of the BActivity and BEducation blocks of the model given in the appendix (see why under 5.2b).

Every instance of BActivity and BEducation will need just one instance of Name, and the designer can specify which. So what we must do is define a parameter in both blocks, and use it in the question texts:

```
BLOCK BEducation
    PARAMETERS
        Name: STRING
.
.
BLOCK BActivity
    PARAMETERS
        Name: STRING
    FIELDS
        Level "What is the highest education followed by ^Name?": EduLevel
```

Now we need to modify the RULES of BInfo, to pass a value for the defined parameters:

```
RULES
   FOR i := 1 TO HouseHold.HHSize do
     Education [i] (HouseHold.Person[i].Name)
     Activity [i] (HouseHold.Person[i].Name)
   ENDDO
```

Note that STRING is not a very satisfactory type definition for the parameter: each instance of the parameter will be 255 characters wide by default. To avoid this, the parameters have to be of the same type as the fields accessed. This means that the type definition has to be supplied at a level accessible both by the block owning the field referred to and by the block owning the parameter. In the example, the common owner of HouseHold.Person [i].Name and of the parameters defined for the names is the main block Survey. So to enable the definition of explicite parameters, you need to define their type in Survey. The definition could take the following form:

```
TYPE NameStr = STRING [22]
```

Now NameStr can be used for the type definition of both the Name field in BPerson and the Name parameter in BEducation and BActivity.

Explicite parametrization of the BInfo block is not necessary: it will have to access all the Name fields anyway in order to supply every instance of its sub-blocks with the value of the corresponding Name field. So we might as well let Blaise do the job.

The resulting number of Name parameters has now fallen from 210 to 30: 10 in BInfo, 1 in every instance of BEducation, and 1 in every instance of BActivity.

7. Appendices

7.1 Appendix 1: the Blaise III example's source

This example was built with pedagogical aims. The THouseHold table is meant as an example of how to design an efficient model. The BInfo block is just the opposite: it gives an example of what not to do. The necessary corrections are supplied in paragraph 6.4.

```
DATAMODEL Survey

   TABLE THouseHold
       BLOCK BPerson
           FIELDS
               Name "What is the name of person ^i?": STRING [22]
               Birth "When was ^Name born?": DATETYPE
               Married "Is ^Name married?": (Yes, No)
               Relationship "What is ^Name's relationship to the head of the
                            household?": (
                                Head   "^Name is head of the household",
                                Spouse "^Name is the head's spouse",
                                Parent "^Name is one of the head's parents",
                                Child  "^Name is one of the head's children")
            RULES
                Name Birth Married Relationship
                IF AGE (Birth) < 16 THEN
                  Married = No "^Name is too young to be married!"
                  Relationship = Child "^Name is too young to be anything but
                                        a child!"
                ENDIF
       ENDBLOCK {BPerson}
       LOCALS
           i: Integer
           HeadCount, SpouseCount: Integer
       FIELDS
           HHSize: 1..10
           Person: ARRAY [1..10] OF BPerson
       RULES
           HHSize
           HeadCount := 0
           SpouseCount := 0
           FOR i := 1 to HHSize DO
             Person [i]
             IF Person [i].Relationship = Head THEN
               HeadCount := HeadCount + 1
               HeadCount = 1 "There can be only one head of the household!"
             ELSEIF Person [i].Relationship = Spouse THEN
               SpouseCount := SpouseCount + 1
               SpouseCount = 1 "The head can have only one spouse!"
             ENDIF
           ENDDO
   ENDTABLE {THouseHold}

   BLOCK BInfo
       BLOCK BEducation
           TYPE
               EduLevel = (None, PrimarySchool, HighSchool, University)
           FIELDS
               Level "What is the highest education followed by ^Household.
                      Person [i].Name?": EduLevel
               Grade "What is the highest grade obtained by ^Household.Person [
                      i].Name?": EduLevel
           RULES
               Level Grade
               Grade <= Level "^Household.Person[i].Name cannot have a
                               ^Grade grade without having followed that
                               level of education!"
       ENDBLOCK {BEducation}
       BLOCK BActivity
```

```
            FIELDS
                Occupation "What is ^Household.Person[i].Name's main
                           occupation?": (School, Work, Unemployed, Retired)
                Income "What is ^Household.Person[i].Name's income?": 0..1000000
            RULES
                Occupation
                IF Occupation = School THEN
                  AGE (Household.Person[i].Birth) <= 18 "^Household . Person [i
                           ].Name is too old to be going to school"
                ELSE
                  Income
        ENDBLOCK {BActivity}

        LOCALS
            i: Integer

        FIELDS
            Education: ARRAY [1..10] OF BEducation
            Activity : ARRAY [1..10] OF BActivity
        RULES
            FOR i := 1 TO HouseHold.HHSize do
              Education [i]
              Activity [i]
            ENDDO
        ENDIF
    ENDBLOCK {BInfo}

 FIELDS
    HouseHold: THouseHold
    Info     : BInfo
 RULES
    Household.ASK
    Info.ASK
 END
```

7.2 Structure of the "Survey" example

```
BSurvey
🖑🕐HouseHold:  THouseHold
¤ 🖑🖘HHSize: 1..10
¤ 🕘🕐Person[1..10]: BPerson
¤  🖑🖘<i>: INTEGER
¤  🖑🖘Name: String[20]
¤  🖑🖘Birth: (Date)
¤  🖑🖘Married: Enum(2)
¤  🕘🖘Relationship: Enum(4)
🕘🖘Binfo: Binfo
  🖑🖘<HHSize>: 1..10
  🖑🖘<Name>[=10]: String[22]
  🖑🖘<Birth>[=10]: (Date)
  🖑🖘BEducation[1..10]: BEducation
  ¤ 🖑🖘<Name>[=10]: String[22]
  ¤ 🖑🖘<i>: INTEGER
  ¤ 🖑🖘Level: EduLevel
```

⌑ 🕓↳Grade: Edulevel

🕓↳BActivity[1..10]: BActivity

  🖇↳<Name>[=10]: String[22]

  🖇↳<Birth>[=10]: (Date)

  🖇↳<i>: INTEGER

  🖇↳Occupation: Enum(4)

  🕓↳Income: 0..1000000

Interviewer Interface of the CAPI system of Statistics Finland

Vesa Kuusela, Statistics Finland

1. Introduction

The CAPI system of Statistics Finland (SF) was technically outlined already in the First Blaise Users Conference (see [1]) although the field interviewers had no computers then and the CAPI organisation did not exist yet. The reported experimental system proved be useful shortly after the conference when it was decided to equip the interviewers with laptops. The telecommunication part of the CAPI system was basically implemented according to those plans. However, in the first plans the rest of the interviewer's workstation software, i.e. the user interface[1], was only vaguely outlined.

An inherent requirement for a CAPI information system is that it is reliable, that is the data must not be in danger under any circumstances. Besides the data security, the user interface should also be functionally and ergonomically well designed. An error in the design of information systems in previous years has been that the end user has been considered as something of minor importance. Such an attitude is in many respects inconsiderate. In making the user interface for the CAPI system in SF it was clearly outlined that the interface should give support to the end users' work. As we all agree, interviewers play a very important role in the data collection process and their commitment is crucial. If we, who work in the survey industry, commit ourselves to the Total Quality Management (see [2]) we should be concerned about the tools and instruments interviewers have, as well as of the survey questionnaires.

In the first place, the interface that the interviewers are obliged to use daily directly affects on their motivation and performance. If they find the usage of the computer too difficult or clumsy, or in the worst case, the software does not seem reliable enough, interviewers will begin to worry about coping in the interviewing. Interviewing thus becomes tense and rigid. It affects the quality of the surveys in many ways if interviewers are forced to use new techniques that they do not like or which they cannot cope with.

Secondly, training interviewers to use an interface that is not user-friedly is very hard and expensive. A complicated structure and functionality that does not follow the process of interviewing conveniently are difficult to understand and hard to remember. After the training period the use of the interface should be transparent so that it does not draw attention from the main task, interviewing.

---

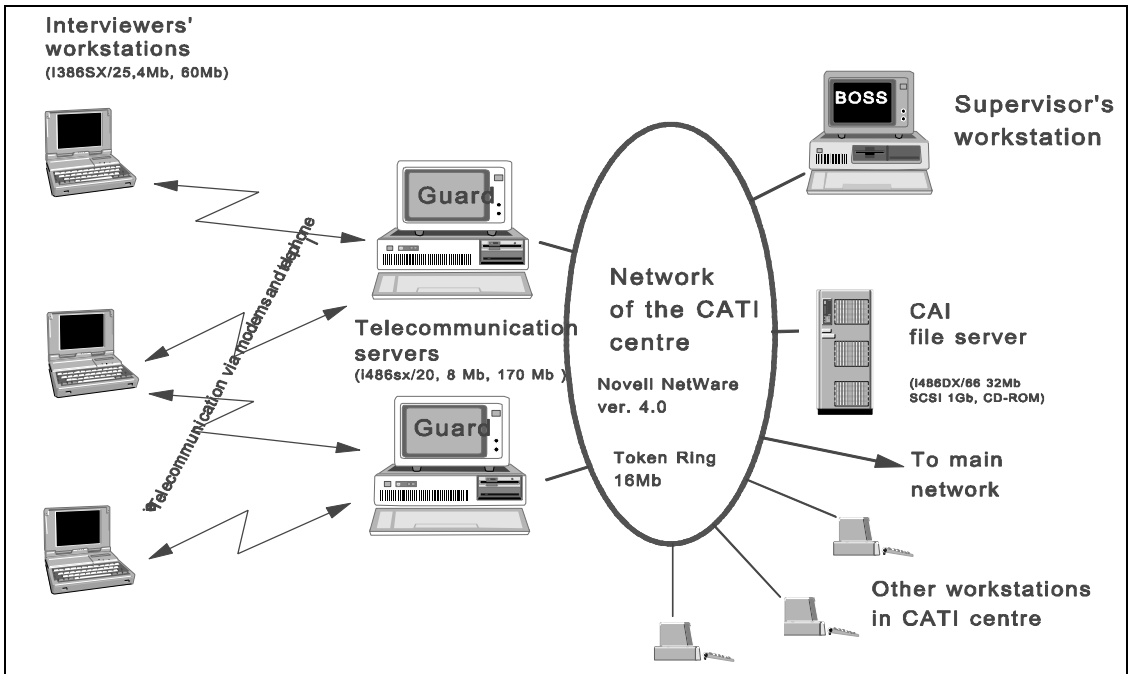
[1] Throughout this text the concept of *user interface* or the *human-computer interface* is used in a wide sense meaning the software which connects the end user to the information system.

Recently designers have become aware that the user interface affects the quality and reliability of the system in every environment. Consequently, the user interfaces have been studied for some time and at the moment there are numerous postulated principles for a user interface that are based on these studies and experience. A number of good books dealing with this subject are also available (see eg. [3] or [4]). However, the requirements for a well-designed interface arise partly from the end users', in this case interviewers', abilities. Thus, the knowledge must be applied in the local environment.

In this paper I will describe the design principles of the user interface and its functions and use.

2. The CAPI Information system

The CAPI information system in SF is composed of two computer programs communicating with each other: the interviewer's workstation software and the supervisor's software



(including the telecommunications software) at the main office of Statistics Finland. To understand some parts of the orchestrations, it is necessary to know how the system should function. The technical description of the information system is shown in figure 1.

The 150 field interviewers' workstations[2] are connected via external 2400 baud modems to

the telecommunications servers (called GUARDs) at the main office. So far two servers have been sufficient. (However, adding new telecommunication servers to the system is quite an easy task.) GUARDs have access to the file server via Token Ring/Netware. All survey data are stored in the file server connected to the main network via a bridge. The whole survey information system is supervised from a network workstation running a specific software called BOSS (Basic Operating Software for Supervisors).

GUARDs are dedicated only to telecommunications, that is to distributing questionnaires and to receiving data. They run 24 hours a day, enabling interviewers to send data anytime. The interviewer is the active party in the initiation of the transmission system and GUARDS wait passively for the calls made by interviewers' workstations. However, after the connection is established, the roles of the client and the server are changed: the GUARD sends data to and/or captures data from the interviewers' workstation. That is, GUARDs cannot make a contact under any circumstances by themselves, and sending files from the workstation is not actually possible. That is, the transmission of data is managed by the GUARD, not the workstation. Traffic in both directions may take place during the same session: if a post packet ready waiting for an interviewer when he or she returns data, the packet will be delivered without a separate request.

When a new survey is launched, an interviewer has to carry out some tasks before interviews can start. Interviewers are usually notified of each upcoming survey by letter. A description of the survey is included the letter, and sometimes also a paper questionnaire, as well as the names and addresses of those to be interviewed. The letter is a signal to the interviewer to fetch the questionnaire using the Get Questionnaire option (see picture 2) and then practice it with real or imaginary interviewees.

Interviewers should transmit data once or twice a week. Due to time restrictions, the case management was solved using a very simple method. All data files containing all cases interviewed so far are transmitted as such. The solution causes many problems but, on the other hand, data security is guaranteed. The design of an adequate case management would have been at least as labourious as the design of the other parts of the interface.

2. Design principles of the user interface

The entire CAPI organization had to be established in less than one year and about three months were allocated to the design and programming of the user interface. There was not thus much time to find and test sophisticated solutions. The design principle can be described as 'quick and sturdy' but the ergonomy and the functionality of the interface were not compromised.

However, the demands for a user interface originate from many different things and some of them are contradictory. The result is always some sort of a compromise between some aims. A list of the aims and principles applied in the programming is presented below.

The user interface was made into *a platform where new surveys are easy to implement*. This was important because the Department of Interview Services at SF also carries out small scale surveys for other organizations, in addition to the large scale surveys of SF. In practice that may mean that an interviewer may be involved in several (up to four or five) surveys during one month. Getting a new survey quickly to the interviewers and back again to be handed over to the customer was an essential feature required from the system.

Another aim was to achieve *easy maintainability*. A prominent feature of today's information technology is that it evolves fast and sometimes unpredictably. Therefore, the software was designed into a platform that can be changed easily for instance to meet the needs of Blaise III or changes in telecommunications. This led to a modular structure. Most features of the software may be changed separately without touching the others.

Another structural decision was to apply the *tree structure* instead of a net structure. The selection was based partly on functionality reasons and partly on the fact that the tree structure is easier to make. And, on the other hand, there is no real need for horizontal movements in a survey interface.

As mentioned earlier, the user interface should be *transparent* so that an interviewer does not notice she or he is using it. This means that using it must be easy.

One of the most determining issues in planning the interface was the observation that most of the interviewers of Statistics Finland had little or no experience in using computers. Even a typewriter was unfamiliar to one third of them. Accordingly, the major task was to develop a 'housewife proof' user interface that, on the other hand, was versatile and reliable. To meet this demand, a group of interviewers tested the software several times before it was released. Tests and training led to a few new decisions, e.g.:

- Most functions are activated by 'pressing the key'. There are no written commands and most functions are 'automatic'.

- No function keys (except F1 for help). The selection of menu functions is done by using arrow keys to highlight the desired function, and then pressing Enter opens it. Function keys were used in the first versions but their use appeared to be difficult and error-prone. So they were discarded.

- Most selections must be confirmed before execution or continuation. This arrangement contradicts the so-called principle of the easiest path, i.e. that the default option in a submenu should be the one selected most often. In the Interviewer's menu the default option is generally return to the previous menu. The confirmation of a selection was found very important in the first version of the user interface. For instance, people who are unfamiliar with the use of a computer keyboard frequently press the key for too long, which actually results in several key strokes. Without

confirmation (and default return) keys pressed too long cause unexpected situations and errors in many cases.

- A close resemblance to the Blaise interface is retained as far as possible. Although the interviewer's menu is a totally different program from Blaise, inexperienced users do not easily understand the difference. In any case, it is more ergonomical, and better in many ways that all parts look and feel the same.

Interviewers had a two day supervised training course at Statistics Finland and right after that unsupervised training in their homes for one week. The supervised training was based on a manual which was delivered at the beginning of the training. The manual 'Interviewer's ADP Manual' is about 60 pages and covers, e.g. the user interface, Blaise, and maintenance of the hardware.

Reasonable disclosure control. As mentioned earlier, an inherent requirement of a system is data security. It should also be made a prominent part of the system. Conveying the message is necessary for the sake of interviewers' comfort, so that they can be certain that data are safe at every stage of the process. On the other hand, ambitious measures to protect data would make the everyday use of the system too complicated. The implemented disclosure control is a compromise consisting of several small parts.

For instance, all data stored in the system are protected against outsiders. When the laptop is turned on the operating system will be loaded from the hard disk. Booting from a diskette requires changes in the setup of the computer. Vital options are protected by a password. In addition, the logic of the telecommunications makes breaking into the system difficult.

3. The user interface and the interviewer's menu

The functions of the user interface are divided in two menus. The most frequently used options are in the main menu, that is

- interviewing
- telecommunication
- back-up.

The Utilities menu contains most of the maintenance functions and some reserve functions, such as:

- diskette operations
- transaction logs
- change of the password or telephone catalogue
- preparing and erasing surveys.

3.1. The main menu

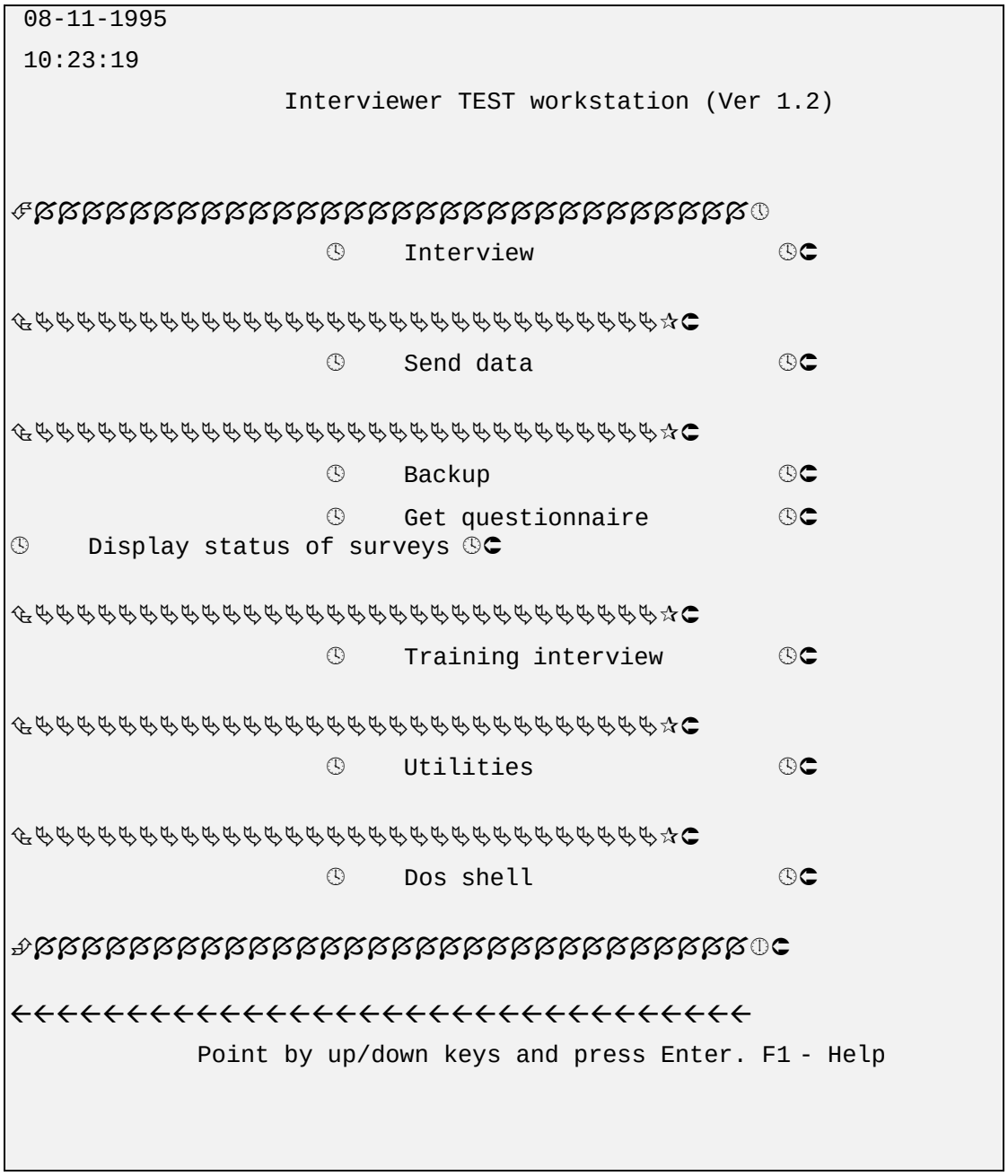
The main menu of the user interface, the Interviewer's Menu is shown in figure 2. At the top is the code of the interviewer ('TEST'). The options of the menu are not in logical order because in this way the probability of an incorrect choice is smaller.

Interviewing

When an interviewer turns on the laptop, the first screen will be the Interviewer's menu (see fig. 2) and the cursor is on the Interview option. The activation of the Interview option brings up a screen with a list of all active surveys. Selection of one survey opens the corresponding Blaise instrument. If the same interviewee should also answer to another survey, the change of questionnaires is possible only through the list of surveys.

Interviewers are provided with the possibility of practising with any survey anytime. The Training Interview option has proved to be indispensable. A new survey becomes more familiar by carrying out some interviews with real or imaginary subjects, and previous (concluded) interviews can be checked without the danger of accidentally changing the

markings. All training interviews will be erased after the training session.



Telecommunication

There are two tasks in which telecommunications are needed: the Get questionnaire option and the Return data option. In both telecommunications options the interviewer may select either of the two modes: whether the contact will be made right away or at a specified time later, i.e. at night. After the initiation, the telecommunications is automatized as far as possible. All possible error situations (?) have been taken into account and treated adequately. An interviewer does not need to follow how things are proceeding.

The questionnaire is fetched automatically. The telecommunications program establishes the connection, gets the zipped source code of the questionnaire, unpacks the packet, and checks

the integrity of the received packet. If the received packet was corrupted during transmission, the workstation asks the telecommunications server to retransmit it. After the packet is safely received, the telephone connection is terminated. However, the user interface software continues in a stand alone mode, without any confirmation from the user. The next phase is to prepare a ready-to-use instrument of the source code. This phase consists of making a directory for the survey, syntax check with Blaise and compilation with the Pascal command line compiler. If there was a message attached to the packet it will be displayed on the screen after the task is completed.

When interviewers *Return data* they have to choose the survey(s) and when the contact will be made. The user interface software automatically packs all the predefined files, dials the telephone number (at the specified time) and sends the packet to the telecommunications server GUARD. GUARD checks the integrity of the received packet, and if it is OK terminates the connection. Otherwise, GUARD asks the workstation to send the packet again. When the transmission is completed and the call terminated, GUARD unpacks the received file and moves the data to the interviewer's directory in the file server.

Back-up

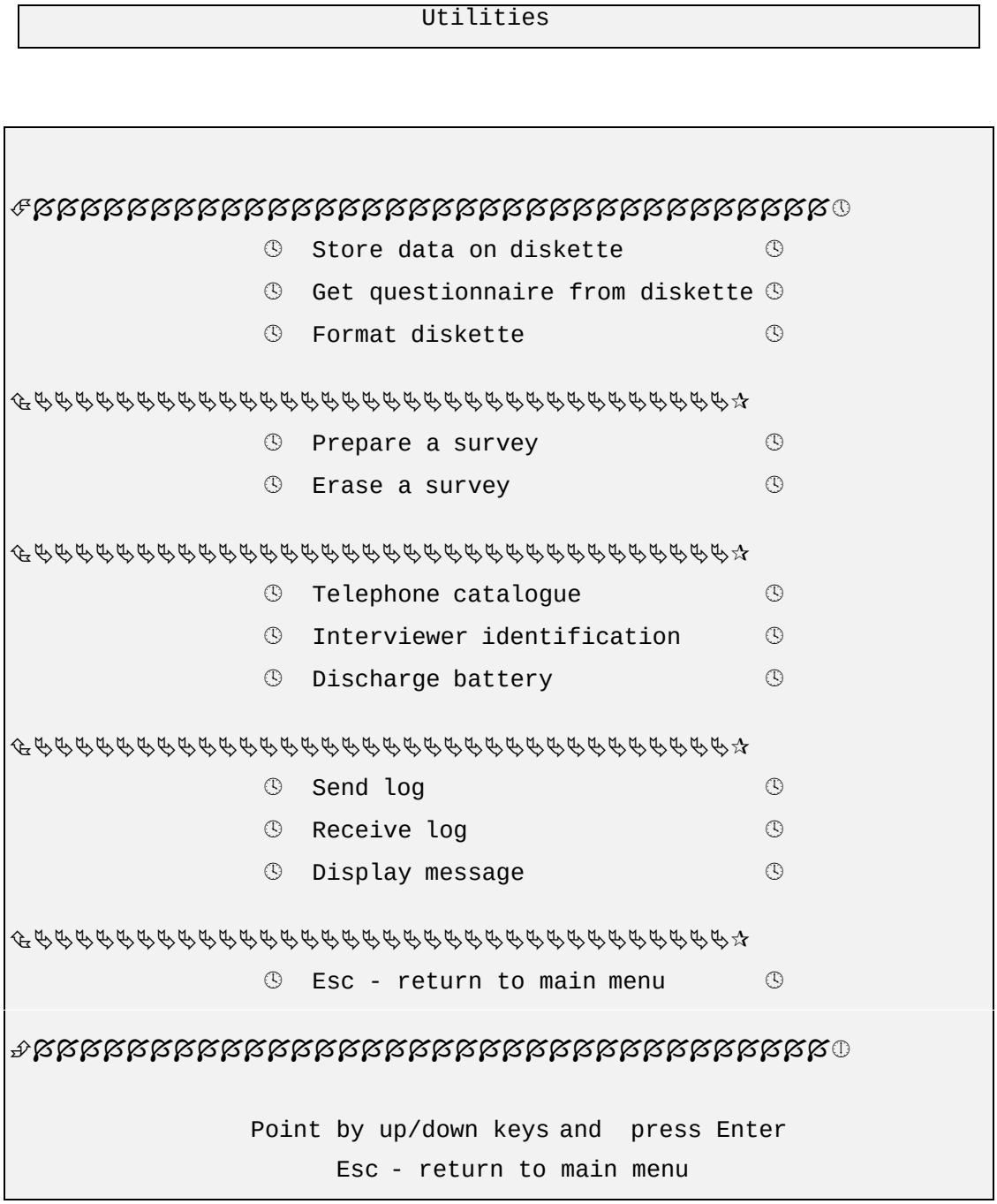
Interviewers are advised to make a full back-up of the surveys every time they receive a new questionnaire, and a differential backup after each day they have made interviews (except the days when they have returned data). Back-ups are made by the DOS back-up program on diskettes.

3.2 Utilities Menu

The Utilities menu (see fig. 3) contains functions mainly intended for maintenance and some reserve functions for telecommunications. It is presumed that these functions are only needed occasionally, not in everyday work.

The Store data on diskette option and Get questionnaire from diskette option are meant for situations in which telecommunications is inapplicable for some reason. At the program level they function in the same way as the telecommunications part. Preparation of a survey means syntax check and compilation of a questionnaire. This option is used in case the instrument is corrupted either during the initial preparation or later.

The Erase survey option erases a survey from the computer's disk. However, interviewers usually do not erase surveys. The cleaning of computers is undertaken by batch files which come with the packets sent by the supervisor. Batch files are ordinary DOS bats and a great variety of DOS operation can be performed with them, e.g. change the configuration of the computer.



In the Telephone catalogue there are the numbers to telecommunications servers. The numbers are added, deleted or changed with this option. Each workstation has a unique identitification corresponding to the interviewer's identification. The computer's id can be changed with the Interviewer identification option.

Transactions are recorded in corresponding logs. The Send log contains the names of files and time and date when the packet was sent. The Receive log contains the same data on incoming packets. With the Display message option the interviewer can read the last message received.

5. Further development

At the moment it is apparent that the requirements set for the user interface have been met for the most part. Some minor changes need to be made in the next version. Overall, interviewers have been quite satisfied. The greatest flaw is the unsophisticated case management system. However, plans are ready for a case management system capable of dealing even with rotating panels surveys.

Naturally, the implementation of Blaise III brings some minor changes to the software. For instance, since compilation is no longer needed, both Blaise and Pascal are redundant. A foreseeable change is brought by new laptops with a colour screen. Colours make the user interface much more efficient but simultaneously colours make the design more complicated.

The need for a more versatile messaging system has been expressed. At the moment, only the supervisor may send a message to all (or a group of) interviewers and only the last message is stored in interviewers' computers. Maybe in the future there will be a person to person mail system where personal feedback in both directions is possible. However, any plans for that have not been made, so far.

Graphical User Interface and mouse

Graphical user interfaces are coming to all systems. Evidently the CAI-systems are no exceptions. I do not share the enthusiasm, however. Especially the use of the mouse is likely to be a problem. The use of the mouse for an inexperienced user even on a surface designed for that purpose is difficult because the accurate movements of hand require exercise and a steady hand. If the mouse needs to be used on an irregular surface it will not work properly. Inexperienced users have found the use of a mouse very stressing and error-prone. Some users have even developed a tennis elbow by using the mouse.

There are many different new types of pointing devices in new laptops but they are not as practical as the conventional mouse. The pen-like pointing devices may be the type which is as good as the keyboard and in some cases even better, e.g. in doorstep interviews. However, the entering of survey data does not require a mouse to become faster or more accurate or even more ergonomical.

References

[1] Kuusela Vesa, Merisalo Antti (1992): *A Prototype System of Data Exchange for Statistics Finland's CAPI system.* In **Essays on Blaise**; Proceedings of The First International Blaise Users Meeting. Statistics Netherlands, Voorburg.

[2] Pietilä Pentti, Niemi Hannu (1995): *Total Quality and Computer Assisted Interviewing; Frames, definitions and Tools for Planning the Total Quality System.* In this book.

[3] Laurel, B, ed. (1990): *The Art of Human-Computer Interface Design.* Reading, Mass.: Addison-Wesley.

[4] Schneiderman Ben (1987): *Designing the User Interface. Strategies for Effective Human-Computer Interaction.* Reading, Mass.: Addison-Wesley.

Poster Session

Role of Manipula Programs in Support of a Blaise III v1.05 Institutional CATI Study

Patrick Murphy, Battelle/Survey Research Associates USA

1. Overview of CATI Study

The Business Responds to AIDS Benchmark Study (BRTA) is funded by the U.S. Centers for Disease Control and Prevention (CDC). Battelle/SRA is responsible for collecting data from companies in the United States. Participation is voluntary. There are five primary objectives of this data collection effort: (1) to provide a baseline with which trends in business activity regarding HIV/AIDS prevention and education can be tracked; (2) to determine the nature and extent of current AIDS workplace policies and education programs; (3) to estimate the overall impact of the Business Responds to AIDS program which was initiated by the Office of HIV/AIDS at CDC in December 1992; (4) to determine the adoption and diffusion of the major components of the Business Responds to AIDS program; and (5) to understand the factors that influence the decision to adopt and implement an AIDS workplace policy, educational program, or philanthropic effort.

Battelle/SRA assisted the CDC investigators in the development and testing of the survey questionnaire which includes questions on a company's demographics, worksite policies, education programs, and philanthropic activities. The survey questionnaire was programmed into a computer assisted telephone interview system (CATI) using Blaise III by Battelle/SRA staff. The CATI system consists of a call scheduling system and a data entry program. On average, the CATI interview takes about 20 minutes to complete. An average of six to eight interviewers simultaneously access the CATI system using a Novell network. Approximately 4,000 firms will be called. To date, the interview has been administered in English, Spanish, and Cantonese.

The BRTA project is an institutional CATI survey which contacts businesses Monday through Friday from 9AM to 5PM local time. The time zone field feature of the Blaise III call scheduler was implemented in this study to compensate for time zone differences across the United States. The survey is divided into eight sections. If the initial respondent cannot provide answers for a section, s/he is encouraged to provide a reference to another person at their company who may be able to answer those sections. Therefore, there may be more than one respondent per firm. To manage this fact, a dual key is assigned to each case, FIRM_ID and PERS_ID, to identify the firm and the person responding to the questions, respectively. The initial respondent is assigned a PERS_ID of '1' and subsequent respondents from the same company are given PERS_IDs of '2', '3', etc. When a respondent gives a referral, the first respondent's data are removed from the main Blaise data base and are saved in another area. A blank FORM is inserted into the main Blaise data base for the referral person. (The term "FORM" is Blaise III terminology for a unique record in the primary study data base. For this study, the primary data base is called CATIVER.~BD.) In this way, only one person from

each firm is in the Blaise data base at any given time, ensuring that only one person per firm will be included in the day batch by the Blaise call scheduler. Another criterion specified by the client was that only one person per firm is allowed to answer a section of questions. Therefore, a method was devised to pass information from the previous respondent's FORM to the current respondent's FORM regarding which sections are still available to be answered by the current respondent.

The multiple respondent feature of the CATI system, customized appointment setting features, and a customized refusal conversion system were all implemented using Manipula programs. This paper describes the structure of the CATI system that Battelle/SRA developed and how Manipula programs were used to implement the study requirements.

2. Structure of CATI Questionnaire System

The CATI questionnaire system consists of three Blaise program files: two questionnaire modules and one CATI module for controlling the flow of logic through the questionnaire and for establishing the CATI blocks.

The two Blaise questionnaire modules are BRTABLOK.BLA and ANSTYPES.BLA. The BRTABLOK.BLA file contains the study question texts in BLOCK format. The RULES sections for each of the BLOCKs are included in this file. The ANSTYPES.BLA file contains answer types for enumerated questions such as (YES, NO) and (HUMAN RESOURCES, FINANCES, CORPORATE POLICY).

The Blaise CATI module, CATIVER.BLA, performs several functions. First, it initializes the system as a CATI system by telling Blaise III to use CATI scheduling data structures. Second, it establishes CATI blocks such as NONRESPONSE and APPOINTMENT. A special block, BIDENT, contains identification information (e.g., name, phone number, time zone, comments) on the respondents. It also has a field that indicates which questionnaire sections are available for the respondent to answer. Third, it contains code to address the special requirements of this study. The most important of these requirements is that each of the sections of questions contained in BRTABLOK.BLA is answered only once by each firm. An IF-THEN-ENDIF logical construct surrounds each section of questions to ensure that each section of questions is only answered once. A further requirement of this study is that multiple people from the same company are allowed to answer different sections (BLOCKs) of the questionnaire but the same section cannot be answered by more than one person. This was accomplished by passing a variable to a new FORM identifying the sections that were unanswered in the old FORM. The IF-THEN-ENDIF logical construct surrounding each BLOCK of questions evaluates this variable to make sure that the BLOCK has not been answered by a previous person at the company before it allows another person into the BLOCK. This logic ensures that a BLOCK of questions is only answered once by any one person.

3. Structure of the Manipula Program System

Manipula programs are used to manipulate data as they enter the CATI system, as they reside in the system, and as they exit the system. For the Business Responds to AIDS project, the majority of Manipula programs are interfaced with the Blaise III call scheduler.

Each subsection that follows describes how Manipula was used in support of this project.

Sample data

A Manipula program was written to load the sample data into the primary study data file, CATIVER.~BD. The program that performed this function is SAMPLE.MAN.

SAMPLE.MAN A block titled "BIDENT", containing the identifying information for each member of the sample, was used as a "DATAMODEL" for the input data in SAMPLE.MAN. The data structure in BIDENT was modeled after the structure of sample data provided by the client. An ASCII file called SAMPLE.DAT contained the sample information formatted according to the BIDENT data structure.

Sample data are read into the Blaise data file, CATIVER.~BD, by using simple INFPUTILE and OUTPUTFILE statements pointing to the BIDENT data model and the CATIVER data models, respectively. An initial appointment of "Monday through Friday, 9AM to 5PM local time" is made for each sample member. The appointment is made in the "Manipulate" section by assigning values to the CATI data structure "catimana.catiappoint" fields.

Referral cases

A special requirement of the BRTA project is to allow more than one person from a company to answer the survey sections. Additionally, only one person from a company is allowed to answer any one section. For example, assume John Doe from Acme Manufacturing Company answered Sections A, C, and E of the questionnaire. He recommended that the personnel director, Jane Doe, answer Sections B, D, and F. She must not be allowed to answer Sections A, C, or E because the first respondent already did.

Before each section of the questionnaire is asked, the CATIVER.BLA program checks to see if that section has been answered by a previous respondent. If not, the questions in that section are routed to the screen for the interviewer to see. The first question in each questionnaire section is "Can you answer questions regarding [the subject of this section]?" If the respondent says yes, then the rest of the questions in the section are routed to the screen. At the end of each questionnaire section, there is a question that states "This section is done. Press 1 to continue." When the interviewer presses '1', that section is marked as complete and is added to the variable which lists the completed sections for that firm. If the respondent states that s/he cannot answer questions contained in that particular section, then the CATIVER.BLA program evaluates the next section of questions. At the end of the interview,

the respondents are asked to provide a the name of a person at their company who can answer the questions in the sections that have been skipped. Because the end of the interview has been reached, the DIALRESULT for this attempt call is 1 (Completed Interview).

SPAWN.MAN The Manipula program SPAWN.MAN scans each FORM in CATIVER.~BD to see if the interview is complete ("complete" is defined as "the respondent has been asked to complete all the sections, even if all the sections were not answered") and if the respondent gave a reference to another person at their company. If these two conditions are met, the FORM enters the "spawning" process. The first step of the "spawning" process is for the referral person's identifying data to be written to an ASCII file called SPAWN.DAT. These identifying data follow the structure defined in the BIDENT block, including the name, title, telephone number, and any specific comments about the referral person. Because a new FORM will be generated for these data, a new set of key data (FIRM_ID and PERS_ID) items must be generated for the referral person. The FIRM_ID for the referral person remains the same but the PERS_ID is incremented by one. For example, if the first interview was with FIRM_ID=10022 and PERS_ID=1, then the referral record would have the FIRM_ID=10022 and PERS_ID=2.

SAVEREC.MAN After the SPAWN.MAN program is run, a Manipula program called SAVEREC.MAN is executed. The purpose of this program is to remove the FORMs from CATIVER.~BD for all cases in which a new case is being "spawned". The SAVEREC.MAN program searches CATIVER.~BD for cases in which the interview is complete (DIALRESULT=1) and a reference was given. (These are the same criteria that the SPAWN.MAN program uses for creating a record for SPAWN.DAT.) These FORMs are moved from CATIVER.~BD to a data file called SAVEREC.~BD which has the same format as CATIVER.~BD. The purpose of this procedure is to ensure that only one FORM per firm will be in CATIVER.~BD at any given time.

ADDREC.MAN The last steps in the "spawning" process occur in the Manipula program ADDREC.MAN. The data in the ASCII file SPAWN.DAT are used to create new FORMs in CATIVER.~BD. The program reads the data in SPAWN.DAT, using the BIDENT block information as a DATAMODEL, and writes them to CATIVER.~BD.

Summary To summarize the "spawning" process, data for referral respondents are written to an ASCII file called SPAWN.DAT using the BIDENT block as a DATAMODEL. A list of the unanswered sections for that FIRM_ID is also written in SPAWN.DAT. The existing data for firms which gave referrals are copied to SAVEREC.~BD, a Blaise data file with the same structure as CATIVER.~BD. New forms, with the same FIRM_ID and an incremented PERS_ID, are written to CATIVER.~BD using SPAWN.DAT as the source of input data.

Refusal cases

For this study, the BRTA coordinator wanted only one interviewer to conduct refusal conversions. Additionally, he wanted all calling statistics on refusal conversion to be kept separate from all other calling statistics. These criteria were met by setting up a separate area

on the network for refusal conversions. FORMs for respondents who refused to participate were moved from the main CATIVER.~BD data base to the REFUSAL.~BD data base. This process of moving the FORMs and performing other necessary details was carried out by the Manipula program REFUSAL.MAN.

REFUSAL.MAN REFUSAL.MAN was executed to identify and extract FORMs from CATIVER.~BD whose respondents refused to participate. If the last dial result for the FORM was '5' (REFUSAL), the FORM is moved from CATIVER.~BD to the REFUSAL.~BD data file in another subdirectory. The REFUSAL.~BD data file has the same format as the CATIVER.~BD data file. The REFUSAL.MAN program changes the last dial result from '5' (REFUSAL) to '4' (APPOINTMENT) for each FORM so that the call scheduler will include those cases in the day batch. If the last dial result is left at '5', the Blaise call scheduler considers the case 'done' and will not include the case in a day batch. The study manager chose to set a period appointment for these cases, Monday through Friday, 9 AM to 5PM local time. All of the calling statistics will be kept in the REFUSAL.~TH file in this subdirectory, allowing separate calling statistics to be maintained.

Not yet called cases

FORMs are loaded into the CATIVER.~BD data base 500 at a time by the program SAMPLE.MAN. The day batch size used for this study is 200. An average of six to eight interviewers spend approximately six hours each per business day going through the day batch. The result of these parameters is that not all the FORMs get called over a two to three week period. Once a case is started, the appointments set for it are given higher priority (by the call scheduler system) than cases not contacted. Approximately three weeks after a sample of 500 cases is loaded, a significant portion of these cases may not have been contacted. A Manipula program NOCALL.MAN was written to resolve this problem.

NOCALL.MAN The Manipula program NOCALL.MAN scans CATIVER.~BD for cases in which the CATI variable "number of calls" is zero. For cases with "number of calls" equal to zero, the program makes a period appointment for that case from 9AM to 5PM the next day. This will change the priority of the case from default (0) to medium (2).

Appointmentexpired cases

If a case has a period appointment (e.g., Monday through Friday, 9AM to 5PM) and the case is never called during that period, the call scheduler priority for that case drops to default, the lowest priority. For the purposes of the BRTA study, a case which has not been called during its appointment period should have a higher priority than the original appointment, not a lower priority. This was accomplished by a Manipula program called NODEFAULT.MAN.

NODEFAULT.MAN The Manipula program NODEFAULT.MAN scans the CATIVER.~BD data file for cases which have an expired appointment period. If a case had a period appointment and the DATEEND of the appointment had passed, then the program makes

another appointment for the case for the next day, 9AM to 5PM local time. This changes the call scheduler priority of the case from default (0) to medium (2).

3. Conclusion

The Manipula programming language, provided with Blaise III and integrated into the system interface, is a powerful tool for customizing the Blaise III call scheduler to the particular needs of a project. The BRTA project used Manipula programs to:

- load sample data,
- "spawn" new FORMs,
- segregate refusal FORMs,
- make appointments for firms that haven't been called, and
- make appointments for firms with expired appointments.

By gaining skill with the Manipula programming language, the CATI programmer can say "Yes" to many more requests from CATI study managers.

Appendix

```
{
       SAMPLE.MAN
       ADD SAMPLE DATA TO DATA FILE
}


uses brtacati "cativer"

   datamodel address
      block bident
         fields
            firm_id: integer[5]
            pers_id: integer[2]
            firm_name : string[30]
            firm_phone: string[14]
            contact : string[25]    {firstname+lastname}
            address : string[30]
            city    : string[16]
            state   : string[2]
            zipcode : string[5]
            timezone : string[3]
            sic      : string[6]
            industry : integer[1]
            empsize  : string[1]
            size     : integer[1]
            abl      : integer[1]
            abinum   : string[9]
            employ   : integer[5]
            title   : string[20]
            location: string[30]
            sectflag: string[16]
            sectdone :array[1..8] of integer[1]
            callername: string[16]
            refcmnt: string[80]
      endblock

      fields
```

```
         ident : bident
   endmodel

inputfile
   infile: address ('SAMPLE.DAT', ascii)

outputfile
   outfile: brtacati ('cativer', blaise3)
   settings makenewfile = no

manipulate
   outfile.catimana.catiappoint.appointtype := 3
   outfile.catimana.catiappoint.datestart := sysdate
   outfile.catimana.catiappoint.timestart := totime(09,00,0)
   outfile.catimana.catiappoint.dateend   := sysdate+7
   outfile.catimana.catiappoint.timeend   := totime(17,00,0)
   outfile.catimana.catiappoint.whomade   := 'WAVE4'
   write(outfile)
```

```
{
       SPAWN.MAN
       FORMATTED FOR CATIVER   FINAL VERSION
       GENERATE NEW CASE, FOR REFERRAL
 }

uses cativer 'cativer'

   datamodel address

      block bident
         fields
            firm_id: integer[5]
            pers_id: integer[2]
            firm_name : string[30]
            firm_phone: string[14]
            contact : string[25]    {firstname+lastname}
            address : string[30]
            city    : string[16]
            state   : string[2]
            zipcode : string[5]
            timezone : string[3]
            sic      : string[6]
            industry : integer[1]
            empsize  : string[1]
            size     : integer[1]
            abl      : integer[1]
            abinum   : string[9]
            employ   : integer[5]
            title   : string[20]
            location: string[30]
            sectflag: string[16]
            sectdone :array[1..8] of integer[1]
            callername: string[16]
            refcmnt   : string[80]
      endblock

      fields
         ident : bident
```

```
   endmodel

inputfile
   infile  : cativer ('cativer', blaise3)

outputfile
   outfile : address ('spawn.dat', ascii)

manipulate   { cases with interview dial result ;
               the interview is done for this one}

   if catimana.caticall.regscalls[1].dialresult = 1 then
         if infile.blkheard.refask=1 then       {gave reference}
            outfile.ident.pers_id := ident.Pers_id + 1
            outfile.ident.callername := 'SPAWNED'
            outfile.ident.contact  :=   infile.blkheard.refname
            outfile.ident.title    :=   infile.blkheard.reftitle
            outfile.ident.location :=   infile.blkheard.refdept
            outfile.ident.firm_phone := infile.blkheard.reftel
            outfile.ident.refcmnt :=    infile.blkheard.refcmnt
            write(outfile)
         endif
   endif
{
 SAVEREC.MAN
 FINAL VERSION
 delete case from cativer database
 save case to saverec database
}

uses cativer 'cativer'

updatefile            {update because we want to read and write}
   updtfile  : cativer ('cativer', blaise3)

outputfile
   saverecs : cativer ('saverec', blaise3)
   settings makenewfile = no
```

```
    manipulate

       { cases with interview dial result }

 if (catimana.caticall.regscalls[1].dialresult = 1) AND
(blkheard.refask=1) then
       write(saverecs)
       delete(updtfile)
    else
       write(updtfile)
 endif
```

```
{
  ADDREC.MAN
  final version
  add spawned records in tmpspawn system to cativer system
}
uses cativer 'cativer'

   datamodel address

      block bident
         fields
            firm_id: integer[5]
            pers_id: integer[2]
            firm_name : string[30]
            firm_phone: string[14]
            contact : string[25]    {firstname+lastname}
            address : string[30]
            city    : string[16]
            state   : string[2]
            zipcode : string[5]
            timezone : string[3]
            sic      : string[6]
            industry : integer[1]
            empsize  : string[1]
            size     : integer[1]
            abl      : integer[1]
            abinum   : string[9]
            employ   : integer[5]
            title   : string[20]
            location: string[30]
            sectflag: string[16]
            sectdone :array[1..8] of integer[1]
            callername: string[16]
            refcmnt:   string[80]
      endblock

      fields
         ident : bident
   endmodel
```

```
inputfile
   infile : address ('spawn.dat', ascii)

outputfile
   outfile  : cativer ('cativer',blaise3)

   settings makenewfile = no

manipulate
   outfile.catimana.catiappoint.appointtype := 3
   outfile.catimana.catiappoint.datestart := sysdate
   outfile.catimana.catiappoint.timestart := totime(09,00,0)
   outfile.catimana.catiappoint.dateend   := sysdate+7
   outfile.catimana.catiappoint.timeend   := totime(16,00,0)
   outfile.catimana.catiappoint.whomade   := 'SPAWNED'

   write(outfile)
```

```
{
       REFUSAL.MAN
       FORMATTED FOR CATIVER, FINAL VERSION
       COPY REFUSAL RECORDS TO SEPARATE SUBDIRECTORY
}
uses cativer 'cativer'

   datamodel address

      block bident
         fields
            firm_id: integer[5]
            pers_id: integer[2]
            firm_name : string[30]
            firm_phone: string[14]
            contact : string[25]    {firstname+lastname}
            address : string[30]
            city    : string[16]
            state   : string[2]
            zipcode : string[5]
            timezone : string[3]
            sic      : string[6]
            industry : integer[1]
            empsize  : string[1]
            size     : integer[1]
            abl      : integer[1]
            abinum   : string[9]
            employ   : integer[5]
            title   : string[20]
            location: string[30]
            sectflag: string[16]
            sectdone :array[1..8] of integer[1]
            callername: string[16]
            refcmnt: string[80]
      endblock

      fields
         ident : bident
   endmodel
```

```
     updatefile
        updtfile  : cativer ('cativer', blaise3)

     outputfile
        outfile : cativer ('k:\cai\brta\refusal\refusal', blaise3)
        settings makenewfile = no

     manipulate   { cases with refusal dial result ;
                    the interview is done for this one}
        if catimana.caticall.regscalls[1].dialresult = 5 then
           outfile.catimana.caticall.regscalls[1].dialresult := 4
           outfile.catimana.catiappoint.appointtype := 3
           outfile.catimana.catiappoint.datestart := sysdate
           outfile.catimana.catiappoint.timestart := totime(09,00,0)
           outfile.catimana.catiappoint.dateend   := sysdate+
           outfile.catimana.catiappoint.timeend   := totime(1,00,0)
           outfile.catimana.catiappoint.whomade   := 'Refusal'
           write(outfile)
           delete(updtfile)
        else
           write(updtfile)
        endif
     {
       NODEFAULT.MAN
       FINAL VERSION
       IF AN APPT PERIOD IS OVER, MAKE NEW APPT SO CASE WON'T BE
DEFAULT
     }

     uses cativer 'cativer'

        datamodel address

           block bident
              fields
                 firm_id: integer[5]
                 pers_id: integer[2]
                 firm_name : string[30]
```

```
            firm_phone: string[14]
            contact : string[25]    {firstname+lastname}
            address : string[30]
            city    : string[16]
            state   : string[2]
            zipcode : string[5]
            timezone : string[3]
            sic      : string[6]
            industry : integer[1]
            empsize  : string[1]
            size     : integer[1]
            abl      : integer[1]
            abinum   : string[9]
            employ   : integer[5]
            title   : string[20]
            location: string[30]
            sectflag: string[16]
            sectdone :array[1..8] of integer[1]
            callername: string[16]
            refcmnt:   string[80]
      endblock

      fields
         ident : bident
   endmodel

updatefile
   updtfile  : cativer ('cativer',blaise3)

manipulate
 IF updtfile.catimana.catiappoint.appointtype = 3 THEN
   IF updtfile.catimana.catiappoint.dateend <= sysdate THEN
      updtfile.catimana.catiappoint.appointtype := 3
      updtfile.catimana.catiappoint.datestart := sysdate+1
      updtfile.catimana.catiappoint.timestart := totime(09,00,0)
      updtfile.catimana.catiappoint.dateend   := sysdate+1
      updtfile.catimana.catiappoint.timeend   := totime(17,00,0)
      updtfile.catimana.catiappoint.whomade   := 'no deflt'
   ENDIF
```

```
ENDIF
 write(updtfile)
```

```
{
  NOCALL.MAN
  final version
  make medium appointments for cases with 0 calls
}

uses cativer 'cativer'

   datamodel address

      block bident
         fields
            firm_id: integer[5]
            pers_id: integer[2]
            firm_name : string[30]
            firm_phone: string[14]
            contact : string[25]    {firstname+lastname}
            address : string[30]
            city    : string[16]
            state   : string[2]
            zipcode : string[5]
            timezone : string[3]
            sic      : string[6]
            industry : integer[1]
            empsize  : string[1]
            size     : integer[1]
            abl      : integer[1]
            abinum   : string[9]
            employ   : integer[5]
            title   : string[20]
            location: string[30]
            sectflag: string[16]
            sectdone :array[1..8] of integer[1]
            callername: string[16]
            refcmnt:   string[80]
      endblock

      fields
         ident : bident
```

```
   endmodel

 updatefile
   updtfile  : cativer ('cativer',blaise3)

 manipulate
   IF updtfile.catimana.caticall.nrofcall=0 then
      updtfile.catimana.catiappoint.appointtype := 3
      updtfile.catimana.catiappoint.datestart := sysdate+1
      updtfile.catimana.catiappoint.timestart := totime(09,00,0)
      updtfile.catimana.catiappoint.dateend   := sysdate+1
      updtfile.catimana.catiappoint.timeend   := totime(17,00,0)
      updtfile.catimana.catiappoint.whomade   := 'NOCALL'
   ENDIF
write(updtfile)
```

Development of a Complex Longitudinal Computer-Assisted Questionnaire for Studying Infant Feeding[3]

James M. O'Reilly, Research Triangle Institute USA[4]

1. Introduction

We describe here the computer-assisted interviewing (CAI) systems developed for a longitudinal research project. The research project studies the infant feeding practices of a representative sample of women enrolled in the Special Supplemental Food Program for Women, Infants, and Children (WIC) of the U.S. federal government.

Of particular research interest are events and influences affecting the mother's decision to attempt to breastfeed, continue breastfeeding, terminate breastfeeding, and feed the child prepared foods. Researchers are interested in the timing of major changes and possibly associated factors such as work and schooling activities, interpersonal influences, and feelings and opinions. Previous research usually addressed these matters with a survey conducted about a year following the child's birth. A key problem is that many mothers cannot recall accurately detailed information about these events months later.

To address these problems, the Food and Consumer Service (FCS) of the U.S. Department of Agriculture specified that the 1994-1995 WIC Infant Feeding Practices Study would be conducted by monthly interviews with mothers or caretakers of the sample infant. The much briefer recall period would enhance accuracy in reporting events, activities, and subjective influences. A central virtue of this approach is that in the interview immediately following a significant change in feeding practices, detailed questioning can be conducted on circumstances surrounding the change at a time when recall would be less difficult.

Longitudinal data collection designs are superior to cross-sections for looking at changes over time. They are less often used, however, because of the greater cost and complexity. Cost is higher because of the multiple contacts with respondents. Complexity is greater because each interview after the baseline must include mechanisms to recall previous responses and conditions in order that change can be detected in the current interview and acted upon.

FCS also recognized that the WIC design would benefit significantly from computerized administration. A computer-assisted interview (CAI) approach can better handle the complexities of a longitudinal study--asking the proper question for each interview, instantly identifying that a key change has been reported, and asking appropriate questions about the change.

Working with FCS, Battelle developed the questionnaire and the CAI systems to conduct the study. The field work began in August 1994 and will be completed in late 1995. While other multi-wave surveys have been conducted with computer-assisted systems, we believe the WIC system breaks new ground in terms of the number of interviews (10), the rapid, monthly cycle, and the degree of integration of the longitudinal design into the CAI instrument. For a general description of the study design, development, and field experience see Williams et al. (1995).

2. Study design

An important issue in designing the study was that many of the low-income WIC mothers would be difficult to contact by telephone. Therefore a standard computer-assisted telephone interviewing (CATI) approach from a centralized calling facility would be problematic. One alternative was a centralized CATI interview followed by in-person interviews of a

supplemental sample of WIC mothers without telephones. A major difficulty of this approach is the substantial costs of interviewing the non-telephone households, since traveling interviewers would be required.

As with all longitudinal research, a critical design issue was retaining subjects in the sample through each cycle. Even a relatively small attrition rate per cycle may result in a final, cumulative response rate that jeopardizes some of the research goals of the study. With a sample population of younger, lower income women, many of whom may change living arrangements over the course of a year, retention seemed likely to be especially challenging.

Our experience in other demanding data collection situations suggested strongly that the key to retention would be establishing a close rapport with the respondent through one-on-one interactions with the same inteviewer over the entire duration of the study. The relationship between the interviewer and respondent could build through the course of the survey. As well, the interviewer could make use of her/his knowledge of the respondent and her household to solve problems of recontacts and tracing in later rounds.

Battelle proposed an innovative design in which there are be two roughly equal subsamples--one subsample of cases for CATI-only interviewing and one subsample of cases to be interviewed by CATI or by CAPI, if no telephone contact is possible. This design is able to produce the required estimates of the national WIC recipient population in both telephone and non-telephone households. An essential feature of this design was conducting the CATI work from interviewers homes, most of whom lived in or near the sampled areas.

Exhibit 1: Study sample design:

	Followup Mode		
	CATI-only	CATI-CAPI	Total
Sites	23	19	42
Participants per site	15	19	
Allowance for non-response	7	8	
Total participants selected	22	27	
Total participants (all sites)	506	513	1009

The design allows the same interviewers to conduct the CATI and CAPI interviews, providing them with sufficent work to retain their participation over the year-long interview period. The alternative of a centralized CATI system, supplemented by a local CAPI interview force to handle non-telephone households, provided many fewer work hours per site and inhibited establishing a close rapport with respondents.

The distributed CATI-CAPI design increases significantly the importance of the systems design for the field work. It must be capable of managing and monitoring interviews and interviewers spread across the country, and handle scheduling, case transfers, and many other functions that are taken as a given in a centralized CATI environment. Exhibit 1 describes the study sample design.

3. Questionnaire structure

The 329 substantive questions were organized into 31 sets which share similar content and timing of questioning. These are shown in Exhibit 2.

Exhibit 2: Description of WIC instrument question groups

1 Demographics of mother, child, household
2 Child and birth
3 Government support
4 Schooling
5 Work history
6 Work plans
7 Maternity leave
8 Child care and feeding at child care
9 Health problems during pregnancy
10 Prenatal care
11 Health problems following birth
12 WIC voucher use
13 Information/advice received from WIC office on
14 Knowledge of WIC feeding recommendations
15 Details on baby's birth and initial feeding
16 Baby's health and eating over past 7 days
17 Current breastfeeders--detailed feeding
18 Current breastfeeders--feeding problems
19 Current breastfeeders--feeding at work/school
20 Stopped breastfeeding since last time
21 Stopped breastfeeding since last time & currently formula feeding
22 Currently formula feeding
23 Breast and formula feeding
24 Formula feeding--detailed feeding
25 Currently feeding neither breast or formula and did formula last time
26 Feeding during past 7 days
27 Discussions how you intend to feed you baby with
28 Ever been told you should breastfeed by:
29 Prior breastfeeding influences
30 Informationi on breastfeeding from other than WIC
31 Agree/disagree about breastfeeding attributes, pluses, minuses

4. Distribution of questions across interviews

Three hundred and twenty nine substantive questions is not an especially large number for a survey. Yet, if asked entirely in one sitting, the interview would be an hour or longer. A virtue of the fully implemented longitudinal CAI design is that only the appropriate questions are asked for each interview. Exhibit 3 shows the counts of questions normally asked in the 11 wave.

Exhibit 3: Number of substantive questions "normally" asked per wave

Prenatal	94
Birth screener	7
MONTH 1	190
MONTH 2	161
MONTH 3	135
MONTH 4	101
MONTH 5	101
MONTH 6	116
MONTH 7	88
MONTH 9	101
MONTH 12	109

The number of questions which individual respondents are asked in a wave is substantally less than that shown in Exhibit 3. For example, all post-natal waves ask questions about why the mother stopped breastfeeding only in the month following that event. Similarly, the series of questions on formula feeding or feeding the baby other foods only apply to a subset of women in any month. As a result, the average length of the monthly interview is about 15 minutes. We believe enlisting and retaining cooperation in the study has been enhanced by the ability to structure the interview this way.

Orchestrating these questions is challenging, however. Each must be asked in the proper waves for the appropriate person. Special situations must also be handled correctly, such as making up key questions when a prior round interview was missed. Exhibit 4 shows part of a question matrix report developed so that project staff can determine which question are to be asked in which wave. For each question is shown

- whether it is normally asked in each of the wave PN, BS, M1 to M12 (Prenatal, birth screener, month 1 to month 12),
- whether the question is to be asked at the next wave if the earlier wave was missed (Ask Next) and
- whether the question is only to be asked of the natural mother not a caretaker (Mom Only).

Waves marked with an 'x' means to ask the question. Those with an 'xn' mean not to ask in subsequent waves if missed.

Exhibit 4: Section of Questions-Asked Matrix report

QUESTION	PN	BS	M1	M2	M3	M4	M5	M6	M7	M9	M12	Ask Next	Mom Only
BABYDOB		x										y	
TWINS		x										y	
BABHOSP		x										y	

MOMHOSP		x						y	y
DOBMOM	x							y	
RACEMOM	x							y	
BORN_US	x							y	
COUNTRY	x							y	
YRSINUS	x							y	
MRTLSTAT	x			xn			xn	y	
HHSIZE	x			xn			xn	y	
HSEHOLD	x			xn			xn	y	
HHREL	x			xn			xn	y	
BABYSDAD	x			xn			xn	y	y
HHONWIC	x							y	
MOINCOME	x							y	
FSPGRM	x		x		x		x		
AFDCPGRM			x						
MAIDPGRM			x						
MOMSMK	x							y	y
EDUCCOMP	x							y	
CURRSCHL	x		x	x		x	x		
HISTSCHL			x	x		x	x		
HRSSCHL	x		x	x		x	x		
PLANSCHL	x								y
CURRWORK	x		x	x		x	x		
PRENWORK	x								
HISTWORK			x	x		x	x		

The 329 questions are asked normally a total of 1095 times (3.32 times per question). Normal means asked in a specified wave. A number of more important questions are asked out of the 'normal' sequence when a prior wave is missed. These are asked in the next interview. Combining normal and make-up question occasions, the questions may be asked in a maximum of 1695 occasions (5.15 times per question).

Another important factor affecting which questions are asked is whether the baby's caretaker is the natural mother or someone else. Some 208 questions are addressed only of the natural mother and the remaining 121 are asked of either the natural mother or other caretaker of the baby.

5. Tracking respondent information across the study

A key part of the design is dynamically tailoring the interview questionning based on information collected in previous months. For example, mothers are asked each month if the baby was immunized since the last interview. Once she responds 'Yes', the question is not asked in the subsequent wave. Many other questions which are dependent on previous responses relate to infant feeding.

Perhaps the most important for this study are those to women who are breastfeeding. Each month mothers are asked, how was the baby fed for the last 7 days--breastfeeding only, formula only, both, or neither.When a previously breastfeeding-only mother reports any other response, then a substantial series of questions are asked about the types of other feeding and when and why the change took place. Fourteen substantive questions are tracked from month to month to control these queries. These items are:

Baby vaccinated
Date of vaccinations
Current infant feeding method
Baby ever drank from a cup
Baby held a cup while feeding
Baby spoon fed
Baby fed from an infant feeder

Baby fed self
Mother's relatives breastfed
Mother's friends breastfed
Baby fed cereal,
Baby fed fruit,
Baby fed vegetables,
Baby fed meat

6. Developing the WIC System

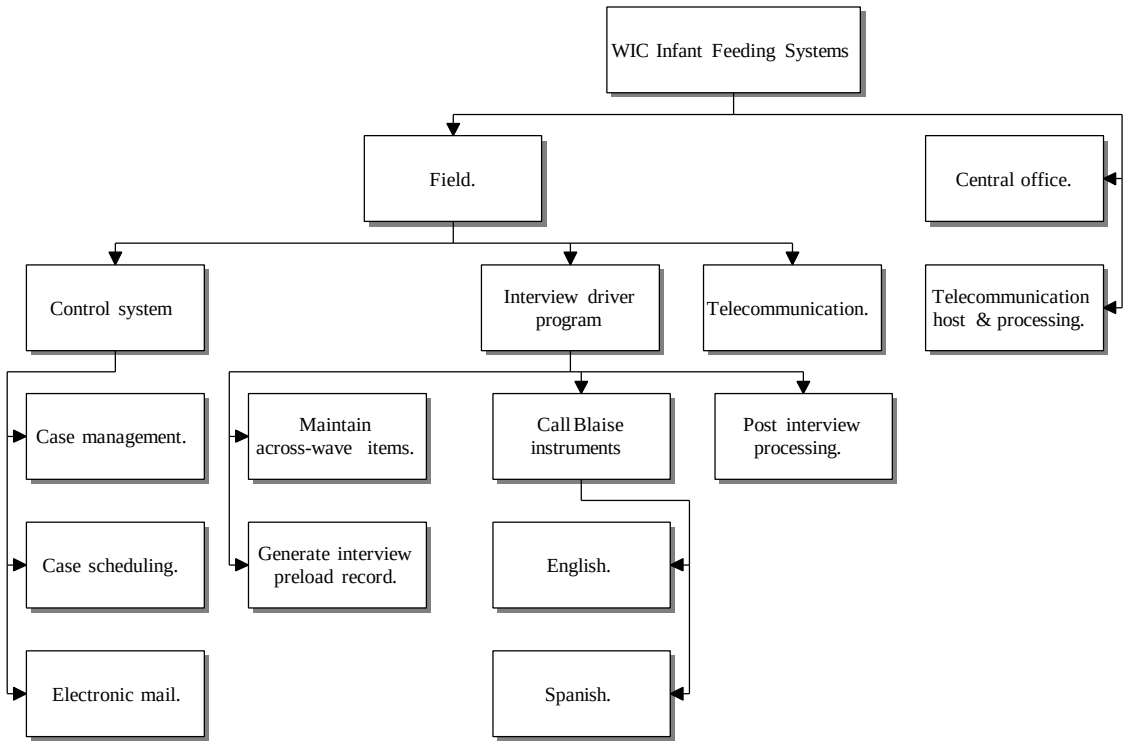
6.1 Systems design

In developing of the WIC CAI system a number of key characterics of the study had to be kept carefully in view:

1. The study is an original. No prior model would be available to follow.
2. The questionnaire was being created from scratch. One could expect changes even late in the process.
3. The system must function as either CATI or CAPI at locations anywhere in the U.S without any local support.
4. The sample population of younger, less educated, low-income women would be difficult to locate, track, and contact by phone. The ability to transfer cases from location to location, to specialized interviewers, and to conduct interviews from the central project office must be provided.
5. The interview cycle would be tight and demanding. Interviews for a given wave must take place in a narrow window of 15 days on either side of the baby's monthly birthday.
6. Cycling interview information from wave to wave so that changes are identified properly is crucial.
7. A distributed processing model is needed, rather than one depending on centralized processes to organize and manipulate monthly data and prepare it for the following month.
8. Daily automated telecommunication of data and mail from interviewers to the project staff is essential for data security, to transfer cases, to provide coordination and support.

Exhibit 5 displays the structure of the WIC project information systems. The central office and central telecommunications host system will not be discussed in detail here. The field system runs on the Toshiba 486/33 Model 1950 laptop PC in the interviewers home. The control system was written in Foxpro by staff of the Battelle/SRA St. Louis office. It's three major functions are general case management, case scheduling, and electronic mail. All are important, but for this project case scheduling is particularly critical because of the narrow window when cases can be interviewed for a wave.

Exhibit 5: Diagram of WIC project systems



6.2 Field telecommunications system

The field telecommunications system was a custom product developed in the Battelle/SRA Durham office. This is a general purpose program designed to handle the full range of automated data communications functions needed for field survey research. Nightly each interviewer selects the automated telecommunication function in the control system. At a preset time between 11pm and 7am each interviewer system calls into the telecommunications host. It uploads all new and changed data and mail files and downloads any files from the host system which the project staff wants the field system to receive. These may be mail messages, cases being transfer to the interviewer, commands for the field system to transfer a case out, or even new software systems. Because this system is completely automated and allows the central office to monitor and control all field system comprehensively and conveniently, managing a demanding and dispersed field study such as this is made much easier.

6.3 Design and development of CAI system

Key considerations in designing the Computer-Assisted Interviewing (CAI) system were:

1. Simplicity of design--because changing and revising the CAI instrument would be required late in the process with minimal time to do the work.
2. Capability of implementing a complete translation into Spanish after the English version become final.

The major programming issue was whether to develop a single comprehensive CAI instrument which included all the intricate logic to handle the myriad contigencies across all

11 waves, or develop 11 separate and more simple CAI instruments, one for each wave. Our first tack was to follow the orthodox systems wisdom--

- keep it separate, simple, and modular;
- avoid complex, comprehensive approaches.

However, after some initial design efforts and early prototyping, we concluded that a single complete instrument could be constructed. And by doing so we would avoid the burden of trying to maintain and change 11 separate versions of slightly different instruments.

The comprehensive approach meant there would be a single instrument and data structure for all waves. The essential unit of observation is a person-wave. Having the data record for all waves the same significantly simplified construction of the analysis file and the analysis itself.

6.4 Blaise Version 2.5 Development System

The CAI development software used is the Blaise system from Statistics Netherlands. We believe the use of the Blaise system was critical to the success of this work. It's conceivable that other systems might have been made to do what we accomplished in Blaise. But, it is very hard to believe they could have done it as well, as quickly (3-4 months), or for the 350 hours of programmer time expended.

One crucial characteristic of the overall questionnaire allowed us to develop essentially 11 separate, but related questionaires within one general Blaise instrument. The questions were always asked in the same order. This means that, for example, for Q43 and Q44, For any of the 11 interviews, any combination of these questions might be asked. But if they both are asked in a round, Q44 is always asked after Q43.

This is important because Blaise 2.5 has strict, structured IF-THEN-ELSE logic for control questionning. No 'GO-TO' statements to jump back against the question flow is permitted. If for one wave, Q44 could be asked before Q43, while in others it is the reverse, then using Blaise 2.5 would have required significant 'work-arounds', tricks, and other efforts.

6.5 Spanish version

Interviews are conducted in English or Spanish. To develop a Spanish version of the instrument our strategy was to wait until the English version of the questionnaire had reach a final state. Once the English CAI instrument was fully programmed, tested, and accecpted , and then we used Blaise tools generate an ASCII version of the questionnaire from the Blaise code, This version was translated into Spanish and the translated text transferred into the Spanish version of the Blaise instrument.

The structure of the Blaise language separates the question definitions--the text, fills, and answer definition--from the questionnaire process--the question order, skips, and related dynamic elements. This was a significant help with the Spanish version. We had to replace the question definitions. But the logic of the instrument was identical for both versions. As well, for interviewing there was one Blaise data file for an interview, whether it was conducted in English or Spanish.

6.6 Questionnaire logic

To understand how the questioning is handled in the Blaise instrument so that different sets of questions are asked in different waves and asked in the next appropriate wave is a wave is missed, Exhibit 6 shows the logic used to control when a set of 30 agree/disagree questions are asked. These questions, BOTLEASY (#350) to DIARRHEA (#379) are asked only of the

natural mother. They are to be asked normally in the prenatal and Month 2 waves. But if either of those wave are missed, then they are asked at the next interview.

Exhibit 6: Example Blaise code

```
if (((WAVE = PN) and (SKIP_PN <> 'Y')) or (WAVE = M2) or (MkUpM2 =
yes)) and
   (RisMom = YES) then
   BOTLEASY  ;  { 350 }
.
.
.
   DIARRHEA  ;  { 379 }
endif;
```

The Boolean expression in the if statement says: if the current wave is Prenatal and the Prenatal wave is not flagged to be skipped, or the current wave is Month 2 or the Month 2 is flagged for makeup (MkUpM2 = YES), and the respondent is the mother (RisMon = YES), then ask the questions. Otherwise don't ask. So a key element was the careful contruction of the Boolean expression which determine whether question are asked. Compared to the 'GO-TO' flow controls which would have be required in many other CAI languages, this was not difficult to do in the first place or, just as importantly, to understand weeks later as the instrument was changed.

6.7 Managing interviews and passing data between monthly interviews

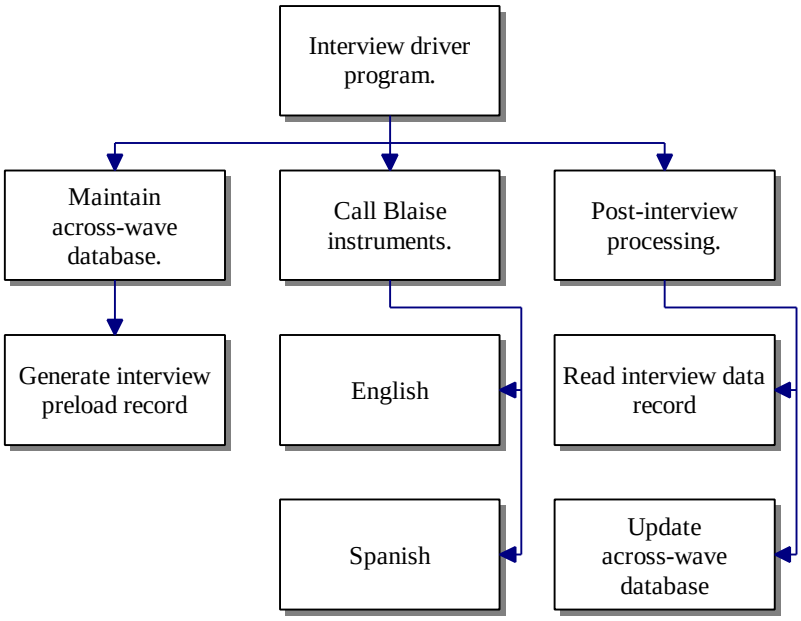
Blaise 2.5 is a powerful, robust, structured computer-assisted interviewing system. It is able to handle relatively large surveys. But it was never designed to manage very long interviews of more than one hour. Also it was not designed to manage complex multi-wave longtitudinal surveys.

However, it is reasonably straight-forward to extend the Blaise system to accomplish these more complex and elaborate surveys. This is possible because of the systems' strong development architecture for interview instrument programming, its data management system, and its DOS system tools for very quickly reading and writing Blaise interview data files and importing external data into the interview process. Together these provide a rich tool set for creating long and complex interview applications.

We did this previously in a very complex and lengthy CAI study, the 1993 National Survey of Family Growth pretest (O'Reilly, 1993). In that study, the instrument exceeded a number of fixed Blaise limits. As well, the instrument included a specialized event history module programmed in Foxpro and an Audio Computer-Assisted Self-Interviewing (ACASI) section. The solution was to develop seven major sections as separate Blaise 2.5 instruments. The overall instrument was managed by a custom Foxpro for DOS driver program which called each section. The driver also generated external Blaise data files of relevant responses from earlier sections, which were accessed by the current Blaise interview. This architecture worked successfully during the 1993 NSFG pretest (Lessler, et al., 1994).

A similar approach was used for managing the WIC longitudinal interviewing process. Each Blaise interview is unique for a study person and interview month. During each interview the instrument must access information about the study person's previous interview history to ask correct questions. This was done using a combination of a Foxpro "driver" program and Blaise tools, as shown in Exhibit 6.

Exhibit 7: Diagram of interview processes.



The major steps are:

- The Foxpro driver program maintains a database record for each case with information about interview status for each wave and responses for all interview data fields which must be passed to subsequent wave.

- When an interview is to begin, the driver program generates an ASCII data record for the case with the key previous interview data fields. Then it calls the Blaise data conversion program which reads the ASCII data record and generates a Blaise 'external' data file which the interview program will read.

- The drivers starts the Blaise interview program (in either English or Spanish), passing as parameters the case ID, type of interview (new or a restarted) and the names of the LIPS data file in which the interview data will be stored.

- Following the interview, when control returns to the Foxpro program, it calls a Blaise program to generate an ASCII copy of the Blaise interview data file, reads the ASCII file, and updates the case's Foxpro data record for information from the latest interview.

This approach worked well. On the notebook PC's, even though often the system was doing many steps to move from process to process--writing data files, calling DOS programs, running Blaise, etc.--the speed was quite adequate.

7. Conclusions

The 1994-95 WIC Infant Feeding Survey has now been in the field for one year and will be completed in a few months. Definitive results on how well the design and systems performed are not yet available. Yet, substantial preliminary statistics on the interviewing, data, and related areas all point to a very successful outcome. Wave-to-wave retention rates have been very high. Overall cooperation of this normally difficult to interview population have been quite strong. See Williams, et al. (1995) for detailed results.

In terms of the information systems, the conclusions look similarly positive. We believe much of the overall success of the survey is a function of the innovative distributed CATI/CAPI design. Gratifyingly, the systems described here were flexible, powerful and robust enough to implement the challenging design. The Blaise system was able to render effectively virtually every features of the questionnaire the substantive experts required. The complex web of questionning based on the child's caretaker, previous interviews missed, previous interview responses and other variables was accomplished to near perfection.

The system also succeeded in managing the overall data systems of some 20 distributed interviewers and a central office so that cases are interviewed at the appropriate times, transferred to other interviewers when necessary, and are available for a central office telephone interview in the case of an incoming call from the respondent. The custom Battelle field telecommunication systems worked successfully managing the nightly automated calling.

We are pleased that this systems design, built around the strength, integrity, and extensibility of the Blaise 2.5 system, has succeeded in supporting this important research.

References

Lessler, J.T. Weeks, M.F., O'Reilly, J.M. 1994 "*Results from the National Survey of Family Growth Cycle V Pretest*". A paper presented to the Annual Meetings of the American Statistical Association. Toronto, Canada. August 1994.

O'Reilly, J.M. 1993. "*Lessons Learned Programming a Large, Complex CAPI Instrument*." A paper presented at the International Blaise Users Conference. London, England. October 1993.

Williams, R.L., O'Reilly, J. M., Vesper, E. 1995. "*Design and Implementation of a Longitudinal Infant Feeding Survey*". Paper presented to the Annual Meeting of the American Statistical Association. Orlando FL, August 1995.

The 1995 June Area Frame Project

Mark Pierzchala, National Agricultural Statistics Service USA

Blaise III (version 1.05) was used in the 1995 June Area Frame (JAF) Survey which is NASS's largest agricultural survey. This survey interviews about 50,000 farmers during the first two weeks of June, with official estimates based on these data released at the end of June. One state (Indiana) used Blaise for CAPI and for Interactive Editing while two other states (Pennsylvania and Wyoming) used the system only for Interactive Editing after data were collected on paper questionnaires. The project was successful though in CAPI two bugs were found that had to be overcome early in the collection period. In preparing for this large survey, design challenges were met so that the instrument could be used in multiple modes for 45 different versions of the survey. Performance of the instrument for CAPI and CADI was good on 486 machines. Both enumerators and editors liked the functionality and useability of the system. The Manipula utility which was used for reports and for data movement was very slow though its functionality (feature list) is greatly expanded over previous versions. Budgeting constraints will limit the implementation of CAPI for other states, however, the instrument could be used in all states in 1996 for editing. NASS views Blaise III as having great functionality and great potential but needing to mature. This survey helped Blaise III in that maturation process.

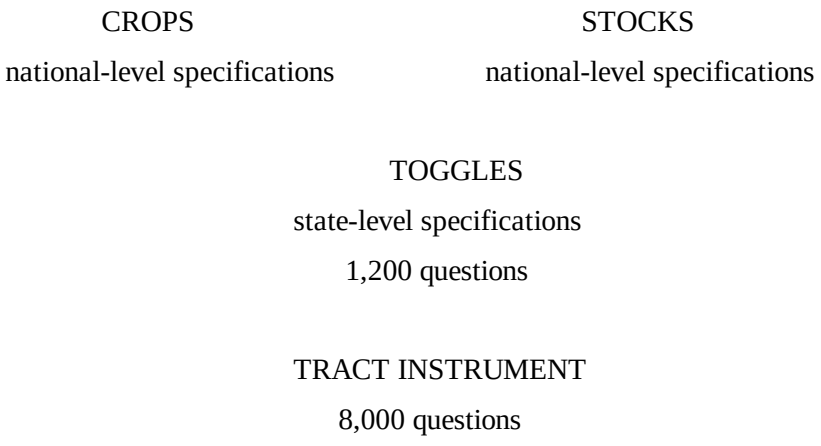
1. Design of the Instrument

The first design challenge is that the June Area Frame Survey can be viewed as a multi-level rostering situation of 4 (or 5) levels. The first level is the segment, usually a one-square mile area of land, which is the Primary Sampling Unit (PSU). Within the segment are up to 78 tracts (land within the segment operated by one operation), within each tract are allowed 35 crop fields (for Indiana CAPI, but 85 for Pennsylvania and Wyoming IE), and each field can have up to 3 crops within a crop year. Within each crop 2 utilizations are allowed but this is not really treated as a 5th level in our design. The tract is the unit of interview as the segment serves only as a unit of administration. Thus the main instrument is a tract-level instrument while a much smaller instrument is used for the segment-level data. The tract instrument will read the segment data when it needs segment-level information and the segment instrument will read the tract data when it is necessary to generate a progress report on the acreage covered by tracts within the segment. The rostering aspect of this instrument did not prove to be much of a challenge as Blaise III blocks, tables, and arrays handle this kind of situation easily.

In its current manifestation, the tract instrument contains about 8,000 overall potential questions and the same number of edits plus many computations and routing instructions. Test versions of the instrument approached 27,000 questions and a like number of edits but this was unnecessarily large. Blaise III never had any capacity limits so one problem we had in Blaise 2.5 (regarding capacity) was indeed solved. Instrument performance was good on a 486 laptop with no noticeable slowdown occurring no matter what the length of the interview. However, performance of Manipula for reports or file handling was very slow.

A related design challenge was that one operator (farmer) could operate land in 2 or 3 PSUs. It is desirable to collect all information from the farmer at one time, and without repeating common farm-level questions for each tract form that must be filled out. So parallel rosters were set up for crops fields, cattle, hogs, and land values which ask information for the tract only. Thus where one roster of 35 fields would normally suffice, there were 2 parallel rosters of 35 fields each to allow the interviewer to ask all pertinent questions as if they all belonged in one form in the first place. This worked well for interviewing but raised questions of data management and increased the size of the instrument just to handle a rare situation.

The second major design challenge was the multiple version aspect of the survey. The JAF survey has 45 versions, almost one for each state. In a previous conference I demonstrated a way to generate versions of questionnaires in a related survey. For the JAF however, a variation on this method was used. Rather than generate separate instruments we decided to use one instrument that could be driven 45 different ways based on an external file of specifications. However this involves much more than just setting flags for whether a question should be asked or not. The external file must itself be based on national-level specifications especially for crops and grain stocks. The following schematic explains:



TOGGLES is the central source of specification for one state. When in TOGGLES a crop (or stock) can be chosen. Meta data from the CROPS (or STOCKS) data base is then transferred into the TOGGLES data base. This prevents a lot of retyping of the same information. Information in TOGGLES includes question-by-question toggles, item codes and SAS Variable names for appropriate questions, some question wording, soft edit limits, and for the crops, a state-specific screen design for interviewers in CAPI. In all, there are about 1,200

potential ways that specifications in TOGGLES can vary from one state to the next. This method worked well; the only limitation was having enough memory to handle the external files. In fact, for Indiana where the laptop computers had only 4 Mb of memory, the external files were too large and we ended up generating a state-specific instrument where these meta data were incorporated.

Under this method of instrument production, a question (or code) in the crops or stocks part of the questionnaire can have one meaning in one state and another meaning in a second state. Thus the instrument itself does not hold all meta data. When reading data into or out of the instrument, or when flashing questions on the screen for the interviewer, meta data from the external TOGGLES must also be used. Again, this worked well in Blaise III. The external data models CROPS, STOCKS, and TOGGLES are all Blaise III data models. They are set up so that non-programmers can state the specifications without having to get into the source code. One advantage of this was seen in Pennsylvania when it turned out that specifications were wrongly stated. The state personnel (who had never before seen Blaise III) were able to go into the CROPS and TOGGLES data models and correct specifications which then allowed them to proceed without the instrument being re-prepared.

A third design challenge was the merging of CAPI and CADI considerations into one data model. Blaise III goes a very long way into making this possible and in fact has no deficiencies in this area. However, there is still considerable thought that must go into accommodating both modes. It is one thing to do post-collection editing in CADI after all data are collected in CAPI, and quite another to do post-collection editing in CADI when data are collected by CAPI in one state and by paper in another. There are considerable differences between CAPI and paper modes of collection and the way that data are handled after collection. For example, in CAPI we rarely use the empty attribute for a question. All questions on the route are to be asked and if the answer is zero then a zero must be entered. However, when data are key punched from paper, zeros are not typed, a blank then represents a zero. The upshot is that we had to program a considerable number of edits that would require an entry in CAPI but allow an empty in CADI. Also in CAPI there are a considerable number of screening questions for each section that are not item code questions. Routing instructions must then be written to accommodate CAPI where the screening questions are necessary and item code data where the screening questions are not entered. Despite these differences in modes, it was still very worthwhile to do both in one system. Well over 90% of the code was applicable to both modes and thus saved considerable resources that would otherwise have to be spent programming the same thing in two different systems. The ability to switch dynamically between modes made testing much easier than in previous versions of Blaise.

A fourth design challenge was to maintain good performance during interview. Blaise III has the ability to maintain speed no matter how long the interview and at the same time enforce all appropriate edits and routes. However, this ability works better for instruments which have structure that is common to rostering situations. We were worried that a large instrument with many blocks at the data model level would be slow in Blaise III. However, on a 486 laptop there was never any problem. We did take two preventative measures from the start based on performance research conducted in December 1994. We declared all LOCALS at the lowest level possible and we reduced the number of parameters being passed into blocks from external files by combining separate bits of information into one string (which was then picked apart as needed). The concern with performance in Blaise III stems from the way it administers parameters which then decides which blocks have to be checked. If there are a lot

of parameters, then the parameter administration can take a long time. NASS instruments typically generate hundreds of parameters. This is no problem in well-structured instruments (where the structuring limits the number of parameters administered at any one time), but is a problem for instruments where there are many blocks on the data model level. We expect Blaise III version 1.1 to eliminate this problem by allowing parameters that do not have to be administered.

A fifth design challenge was to manage the survey totally by electronic means both on the laptop and on the LAN in the office. This challenge was not met. We were expecting to use the newly enhanced version of Manipula called ManiPlus which would have allowed point and shoot access to forms and the like. However, ManiPlus is a version 1.1 product which was not available at the time we went into the field. Thus laptop management was largely done both by paper and by regular Manipula generating reports from DOS BAT files. We have seen ManiPlus in alpha versions and expect this utility to cover all of our laptop needs as well as some for editing. For example, we would like to do away with the segment-level instrument next year and let ManiPlus handle its administrative functions.

2. Blaise III CAPI in the Field

The CAPI part of the test met most expectations once two bugs were overcome. As interviewing software Blaise III performed well. The 35 interviewers in Indiana were able to handle a variety of difficult situations including non-response, handling two tracts at once for one operator, error correction, and the large field table roster. The majority of the interviews were collected outdoors, sometimes in adverse conditions. In addition to the computers the interviewers had large aerial photos where they had to draw off the farmer's land and fields, so logistics were difficult. Interviewers persevered despite these challenges and the fact that farmers were often impatient as they were behind in their planting. We were relatively pleased with this aspect of the project.

However the bugs initially caused a lot of consternation for both interviewers and office staff. The first bug was in Manipula which would sometimes blow up in a runtime 216 error when copying forms with comments. The fix was to read data from Blaise III format to ASCII, transmit them, then read them back into Blaise III. This worked but reduced the overall efficacy of CAPI because subsidiary information was lost. Comments, DontKnows and Refusals, and error suppressions did not survive the Blaise to ASCII to Blaise dance (comments were read out to a separate ASCII file).

The second bug concerned a part of the grain stocks section. The data would be collected as expected and saved when the interview was closed. However, if an interviewer brought the data up for review this part of the instrument wouldn't be on the route. Thus when the form was saved a second time that part of the grain stocks information would be lost (temporarily for the most part, we had backups). We don't know the extent of the problem here, but most interviewers did not normally review forms, they just sent them. An ad-hoc patch was put in the instrument to prevent this from happening.

Both problems were discovered late on Thursday, June 1. Solutions were conceived and executed on Friday, June 2. On the positive side, NASS discovered it could deal with adversity, even modify procedures, during the operational survey where the data collection period lasts only 15 days. However, NASS does need to think more in advance about how to handle such emergencies and how to test to avoid them.

NASS had a variety of problems associated with hardware. The subnotebook computers came from 3 different purchases; as a result we had Zeos, NEC, and AMS laptops (laptop models don't stay in production very long). The Zeos computers were 2 years old and by the second week of the survey 6 of 15 had broken down. It is possible that 4 pound subnotebooks are not as reliable as 6 pound subnotebooks. The one-year-old NEC and new AMS computers held up better but the NEC battery life was often disappointing. In fact some interviewers had to switch to paper when all 3 of their batteries were dead. Charging batteries was difficult because none of the interviewers had external battery chargers. The computer could be used to collect data or it could be used to charge batteries, but it could not be used for both at the same time during a personal interview.

The NEC and AMS computers had 4 Mb of RAM; the Zeos had 8 Mb. The smaller amount of memory was barely sufficient for the instrument. In fact it was difficult to recall some forms for review after they had been closed. They could be recalled, but only after Blaise III had been 'exercised' by bringing up 2 or 3 other forms first. NASS will have to buy more RAM and battery chargers.

3. Interactive Editing in Blaise III

Interactive editing proceeded well from a Blaise perspective. Several minor updates in the instrument were needed but that was a NASS problem. Data were read directly from an 80 column item-code format into the Blaise III data set and out again with Manipula. Cameleon was used to set up a considerable part of the Manipula programs that did this. The new editing interface was very well received by the users. On occasion Blaise would have to read information into the form from a double external file read. The first would obtain an ID from the list frame, the second would obtain stratum codes associated with the ID. The first file was about 2000 lines deep, the second about 90,000 lines deep. This double read of external data was executed almost instantaneously by Blaise. Pentium machines were used to handle the batch activities; otherwise 486 machines were used by most editors. All 45 of our offices have equipment and staff in place to handle the interactive edit; thus we expect to see a move to make this fully operational. The instrument has already been written to handle all states and all we need are state-level specifications which are entered into the TOGGLES data model. Also needed is a lot of testing by operational personnel.

Overall the experiment was very successful. This article is being written on June 26 while the survey ended June 16. Thus several weeks of review is still necessary to completely evaluate our use of Blaise III in the June Area Frame.

Total Quality and Computer-Assisted Interviewing; frames, definitions and tools for planning a total quality system

Pentti Pietilä and Hannu Niemi, Statistics Finland

1. Introduction

Total quality management (TQM) and computer-assisted interviewing (CAI) could well be compared to the quality of a fuel injection system and the total quality of a car. The aim of this paper is to give a broader view and metaphorically speaking to discuss about the car - that is, to discuss about total quality management, which gives a frame for CAI. A car will not work if the fuel injection system is faulty and the same applies to CAI as a subsystem of a survey study.

Total quality management is essential for the survival of the survey industry regardless of whether the sector involved is the private, public, or non-profit one.

2. Computer-assisted interviewing

CAI is estimated to be one of the main data collection methods in survey research in the near future. Its "competitors" are mail surveys and administrative records. Interviewing offers some very significant advantages. It is for example not "a second hand shop". The data obtained are always new. Almost anything can be asked. Furthermore, the results are obtained quickly and in this respect CAI offers major advantages. The disadvantages are the costs in data collection and the respondent burden.

However, interviews form only part of a survey research process. As a simplified model one can distinguished the following subprocess in a complete survey research:

- survey design
- sampling
- questionnaire design
- management
- data collection
- data processing

- analysis
- dissemination.

The use of computers in interviewing has an effect on all of the subprocesses.

3. Definitions of total quality management

The survey question: "What does the word quality mean?" would probably bring as many answers as there are respondents. It has been claimed that TQM is no longer applicable and that by the year 2000 quality will be self-evident. Quality thinking started from process control, but later evolved into a much broader concept. This may be one reason why quality thinking has survived so long - it changes form continuously like an amoeba.

David Garvin [2] has presented five points of view on quality:

* The transcendent view of quality is synonymous with innate excellence, a mark of uncompromising standards and high achievement. This viewpoint is often applied to the performing and visual arts. It argues that people learn to recognize quality only through the experience gained from repeated exposure. However suggesting that managers or customers will know quality when they see it offers little practical guidance.

* The product-based approach sees quality as a precise measurable variable. Differences in quality reflect differences in the amount of some ingredient or attribute possessed by the product. Since this view is entirely objective, it fails to account for differences in individual tastes, needs and preferences.

* The user-based definition starts with the premise that quality lies in the eyes of the beholder, equating quality with the maximum satisfaction. This subjective, demand-oriented perspective recognizes that different customers have different wants and needs.

* The manufacturing-based approach, in contrast, is supply based and is primarily concerned with engineering and manufacturing practices. It focuses on conformance to internally developed specifications, which are often driven by productivity and cost containment goals.

* Value based definitions define quality in terms of value and price. By considering the tradeoff between performance and price, quality comes to be defined as "affordable excellence". This perspective makes clear that one can have a high quality motel as well as a low quality five star hotel.

Basic principles of TQM are:

* Customer orientation
* Internal customer-supplier partnership
* The process view
* Suppliers as a part of the system
* Total employee involvement
* Continuous improvement
* Quality measurement and management system
* Employee training
* Vision and strategic goals.

4. Levels, frames, and tools in TQM

Quality can be discussed on many different levels of an organization:

* Managing the whole organization by mission, vision, strategic goals, management structure and shared values

* Managing different activities and processes such as data collection, data analysis and reporting, where products are considered, production, information technology, research and development, customer relationships.

* Managing different subprocesses (for example a survey)

* Personal level, where personal skills, attitudes and knowledge are key factors.

Different tools are needed at different levels. On the two higher levels the use of concepts related to quality awards are very useful. Deming [1], Malcolm Balridge and the European Quality awards are well-known sources that offer useful information on strategic quality thinking. The structure of the European quality award is appended in appendix 1. Bench marking - comparing your organization with similar organizations - can also give a stimulus to work for better quality on the organisational level. Quality assurance matters are covered extensively in the ISO standards. An example of ISO 9001 standards used in a survey research organization is presented in appendix 2, and the concepts used in these standards are "converted" to the survey research framework. Level three involves working with specific surveys where a new si planned or old surveys are reengineered or renewed. A very interesting model for these purposes was developed by the US Bureau of Labor Statistics (see appendix 3).

The main principles of quality improvement include:

Customer orientation

Quality is related to customer demands. Deviations from expected quality result in customer dissatisfaction. Customers and their needs must be identified. The means of measuring customer satisfaction is necessary. Martin Collins [4] has pointed out that loving care instead of meeting contract standards is important in developing the customer relationship management.

On the other hand, excessive customer orientation can be damaging. Customers are not necessarily aware of all the better quality options available. The purpose is to reach a level of quality where the customer is surprised in a positive way.

The process view

All the products and services, as well as their quality characteristics, are produced within a process. Process thinking is a dominant feature in TQM. The use of characterization in terms of functional organizations is less important whereas the framework of horizontal processes and process owners is more relevant. Business is organized around key customers. The building blocks of all the processes consist of subcontractors, internal suppliers and customers.

Processes consisting of subprocesses within different departments contribute and add value to final products. An internal customer supplier interface must exist between subsequent subprocesses, as with external customers. These internal customers must be identified. Every process must have a process owner, making the organization very flexible. A new customer or a new product is reason enough for reorganisation. Organizational levels also become thinner and the personnel becomes more customer-driven than driven by their superiors.

Continuous improvement

"Right first time" is one of the slogans frequently used in TQM. It implies that error prevention - avoiding faults happening in the first place - is much cheaper than detecting faults and correcting them afterwards. By not doing things right first time we:

* waste time
* cause trouble to others

* put ourselves under pressure
* put customer satisfaction at risk

No matter how much improvement is made, our competitors will continue with their quality improvement efforts and our customers will expect quality that is still better. Small steps to improve the efficiency of our processes and to meet new customer needs are taken continuously. Major changes are followed by some reengineering and big steps to fulfill the requirements must be taken.

Occasionally problems arise. A stepwise approach to problem solving increases the possibilities in problem solving:

* define the problem
* look for root causes not special (random) causes
* choose a solution
* implement the solution
* check that it worked.

Once the solution to the problem has been found, the following questions should be asked:

* will it solve the problem?
* will it solve it forever?
* will it prevent the problem renewing itself?

If you do not know where you are going, all the roads will take you there!

Total employee involvement

Employees throughout the organization are encouraged to be responsible for quality. Most people would rather do a high quality job than a poor quality one. Hence each employee needs to know how she or he adds value in the process and how quality is measured.

Suggesting quality improvements is essential. In order to help improve quality it is necessary to ensure that:

* the right tools and training are available
* systems are designed to help and not hinder
* guidance is offered
* documents are available.

Working with one another is a lot better than working against one another. The best teams have good individual players, but the winning team is created by playing well together. Working in teams helps to:

* achieve things individuals alone cannot
* make best use of all our skills
* make better decisions
* get more enjoyment from work.

It is also very important not to separate those who plan and those who actually do the work. Everyone must be offered a chance to develop his/her own work.

Employee training

Quality training is necessary for everyone. By learning to use different quality tools we can contribute in quality and it is easier to understand personal influence in quality. Special skills in problem solving and teamwork as well as customer relationship management skills are all essential.

An organization needs not only qualified staff, but also employees that are willing to improve their skills continuously.

Management, communication and recognition

A clear vision, shared goals and values are necessary, and constancy of purpose for product and service improvement should be created. TQM puts emphasis on managing processes, people and customers. Taylorism as a management system is outmoded. Employees cannot do their jobs better if massive inspections are held or if work is divided into small segments.

Work standards should be eliminated, as should management by objective. The responsibility of supervisors must be changed from sheer numbers to quality. Barriers between staff areas should be broken and everyone should be allowed to be proud of his or her job.

5. Considerations between TQM and CAI

Quality is not for sale - you cannot buy it. Similarly you cannot teach it, but you can learn it. Every organisation must make its own quality system to fulfill the special needs and goals of that specific organisation. However, it is possible to suggest some useful points when planning a total quality system for CAI operations.

Work done in CBS Netherlands by Keller and Betlehem and the Blaise team on the integration of data collection, processing and dissemination and planning and developing the Blaise system for these operations offers very significant advantages in becoming leading survey organizations in the total quality sense in the future.

Some advice in planning quality:

* Avoid the creation of a cati centre, which uses the methods of Taylorism and reflects offices of the past, where a supervisor sitting high up knows everything and interviewers sitting down in straight lines only know how to read questions and how to type the answers.

* Do not rely on massive inspection and control once a survey has been completed. Try to avoid errors in planning and programming.

* Identify your CAI service as a system and focus on internal and external customers and make a quality agreement with each customer, and make an alliance with your subcontractors.

* Lead different processes instead of functional units.

* Do not separate planning and doing. Let everyone be responsible for quality and be proud of it.

* Try to reach the quality award level as a strategic quality standard. Following the principles is more important than winning the award.

* Create a quality system for CAI-operations by defining subsystems and evaluating, developing, implementing and documenting them. Use internal auditing and possible certification to make sure you follow the documents on different levels. A quality system is never ready and continuous improvement and development is necessary.

* Cooperation between BLAISE users in different countries offers crucial benefits in reaching excellent quality with fewer resources. The basic quality system for CAI can be implemented in cooperation. Setting a high quality standard for CAI is a worthwhile challenge. Achieving that level would almost be equivalent to winning a quality award!

References

[1] Deming Edwards, (1986): *Out of the crisis*,

[2] Garvin David, (1988): *Managing quality*.

[3] Colledge & March (1993): Quality management; Developing of a Framework for a Statistical Agency, ASA, April 1993.

Papers presented by

[4] Martin Collins: *Survey quality; The user's view*

[5] Lorna Hall: *Quality certification in the survey research industry*

[6] Cathy Dippo: *Integrating the concepts of survey measurement and process improvement*

at the *International Conference on Survey Measurement and Process Quality*, April 1995 Bristol, UK.

Selective and Automatic editing with CADI-applications[5]

Frank van de Pol, Willem Molenaar, Statistics Netherlands

Summary

When designing a CADI-machine for a business survey, the implicit approach with Blaise is to edit every record with the same effort, even though some firms contribute vastly more to publication figures than others . Canadian, Swedish and Dutch research into the effectiveness of this approach has shown that in the cases under investigation at least 50% of the edits have virtually no effect on publication figures. This calls for methods to skip ineffective editing activities.

One way is to select records which have a high risk of containing influential errors, the critical stream, for Blaise-editing. The remaining records, the non-critical stream, may remain unedited. The part of the non-critical stream that has Blaise status 'dirty' or 'suspect' may also be 'cleaned' by a routine for automatic editing.

The art of selective editing is to devise a powerful and yet practical formula to determine the risk that a record contains influential errors. This involves taking account of inclusion probabilities, non-response probabilities, size of the publication cell and, most important of all, a benchmark to determine whether an observed score may be in error, like the cell-mean (or median) for that variable.

Partly drawing on experiences from official statistics in other countries like Canada, the US and Sweden, the principles of automatic editing will be briefly dealt with.

1. Introduction

This paper describes editing business survey data in a CADI (computer assisted data input) situation. Business survey data most often are collected with a survey by mail. The paper forms are returned and the data have to be entered and edited. With the introduction of Blaise-CADI, data entry and data editing have been integrated into one process, thus avoiding unnecessary wait queues due to division of labour. This advancement has a drawback, however, which is that with most Blaise data entry applications a copy of the original, unedited data is no longer saved, thus obstructing research on the effects of editing. For one survey, the Annual Construction Survey (ACS), we changed the production process to obtain unedited data.

Data were entered 'heads down', without changing any entry. Every day the newly entered records were saved and added to the file with unedited data, and their status was set to 'old'. Subject specialists would edit 'old' records only, using Blaise-CADI machines which contain almost 100 messages for errors or suspect conditions. According to these criteria, most records needed correction. In this way two versions of the data were obtained, one with ***un***edited data and one with edited data (euphemistically called: 'clean'). By comparing these two data files one can monitor data editing quality. We used these files to evaluate the effects of data editing.

2. Some edits have more effect than others

Boucher (1991) and Lindell (1994) compared unedited data with clean data and found that, for each variable, 50% - 80% of the edits had virtually no effect on the grand total. We reproduced their analysis for twelve ACS variables, two of which will be shown graphically: total number of employees and total production. Figure 1 shows these figures for three classes of business: contractors in residential and non-residential buildings (top), contractors in civil engineering (soil, roads, bridges; middle) and other building completion work (bottom). The lines go from no editing (on the left) to complete editing (on the right). Edits are sorted from the largest errors (left) to the smallest errors (right).

Figure 1 shows that data editing had little effect on the estimates. The level on the (unedited) left side of each graph is almost the same as the level on the (edited) right side. For only one of the 12 variables we looked into, a strong editing effect was found, namely trade benefits (not in a figure). Editing reduced this amount from about 2% of the production value to less than 1%.

Figure A. Edit effect on number of employees (left) and total production (right)

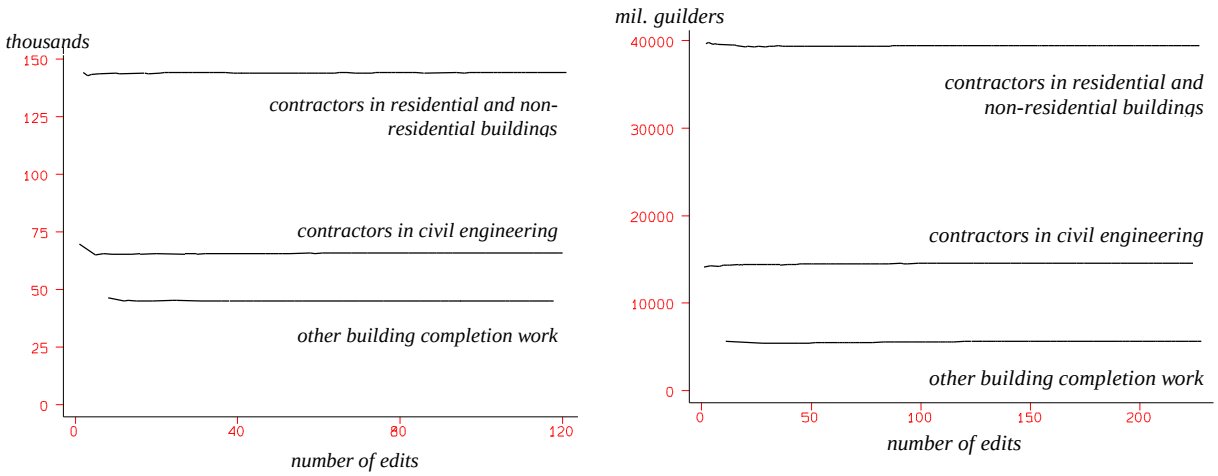


Figure 2. Edit effect on number of employees, vertical axis stretched

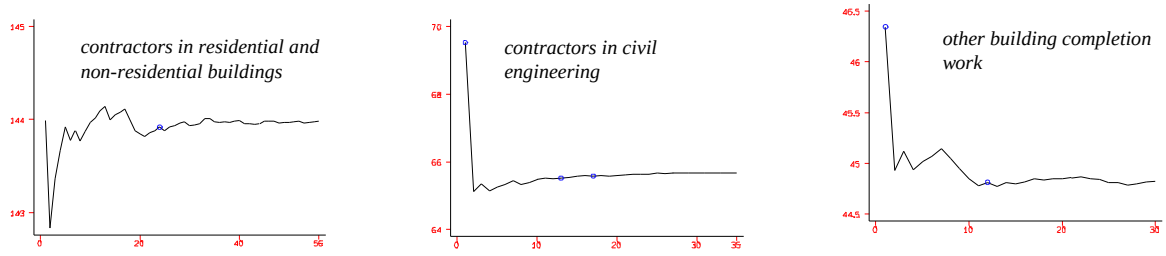
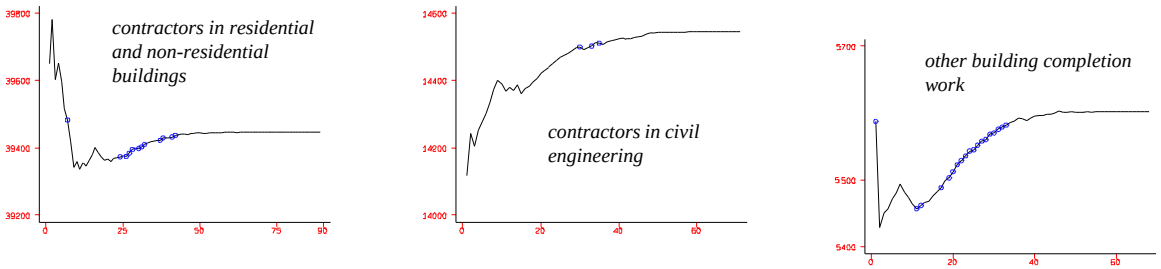


Figure 3. Edit effect on total production, vertical axis stretched



Zooming in to each of these lines (figures 3 and 4) it shows that the small effect that does exist is merely due to the first 10 to 50% of the edits, the largest ones. Like Boucher and Lindell, we find that at least the last 50% of the edits has virtually no effect. This does hold for all the 12 variables that we had at our disposal, even for trade benefits.

Can we know in advance which records hold the more influential errors? Some suggest that only large firms generate influential errors. To test this hypothesis edits on small firms (<10 employees) have been highlighted with a dot in figures 2 and 3, if they occurred in the left half of each line, that is the half of the biggest errors. If small firms would not have to be edited the lines in these figures would not show any dots, but these lines do show dots, sometimes even at the leftmost point, as the largest error. So it shows that not only large firms, but also small firms generate influential errors. Therefore we developed something more sophisticated to pinpoint the records with influential errors.

3. Which forms will have to be edited manually (with Blaise) ?

When a large part of the edits has little or no effect on the publication figures, the question arises: 'How can we avoid the effort of non-effective edits?'. For this aim a 'Safety-Index' is conceived which should predict whether a record needs editing or not. The index was inspired on a paper by Hidiroglou and Berthelot (1986), who worked on a survey with one target variable. The ACS, however, has several target variables.

Traditionally, important tools with ACS editing were ratios of key entries like 'number of employees', 'number of mandays worked', 'production', 'labour costs' and 'trade benefits'. For these ratios upper and lower limits were defined. Moreover, the ACS involves tests on the correctness of the sum of detail-entries. Such a test can also be written as the ratio of the computed sum and the reported total, which has to be 1.

For our safety-index we aim at summarizing these ratios (j = 1, ... J). The index should be sensitive for the effect of an error on publication estimates of totals. Hence it should also take the sample-inclusion probability and the response probability into account. In a previous paper it was argued that the difference of a ratio $r_j = y_{nj} / y_{dj}$ (n for numerator and d for denominator) from its median, $med(r_j)$, can be intepreted as a crude estimate of an error (Van de Pol, 1994). In the present paper we propose to make a symmetrical deviation measure by *dividing* each ratio by its median, $r_j / med(r_j)$, and taking the largest one of $r_j / med(r_j)$ and $med(r_j) / r_j$, that is

$d_j \equiv d(r_j) = \max\{[r_j / med(r_j)] , [med(r_j) / r_j)]\}$ or, equivalently,

$d_j \equiv d(r_j) = \exp\{abs[\log(r_j) - \log(med(r_j))]\}$.

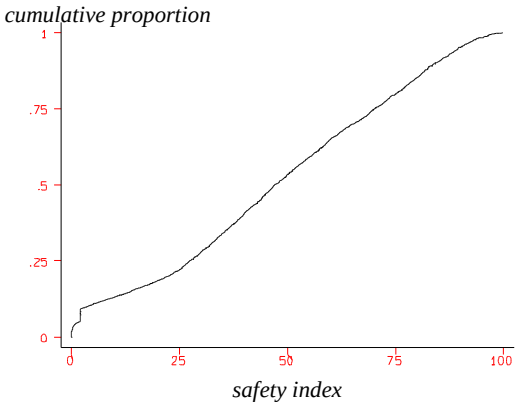
Latouche and Berthelot (1992) proposed other symmetrical measures for the case that more measurements, y_{nj} and y_{dj}, of the same firm are available from a previous time point.

The effect that an error in a ratio has on a publication total (numerator or denominator) depends on the contribution of that firm to the total concerned. Because we have to deal with unedited data, we considered several indicators for this contribution, in order to eliminate erroneous ones. We chose four highly correlated variables, x_1 = 'number of employees', x_2 = 'mandays worked', x_3 = 'net production' and x_4 = 'net production minus production via subcontractors'. All these indicators were transformed to standard deviation 1. Next, to obtain a robust mean of them, the smallest and largest value were eliminated and the average of the two middle values was taken as the contribution of the present firm, denoted by i (importance). If many entries were empty (the importance was zero) the importance was estimated from the firm size class, a 9-category variable which is known from other sources than the ACS.

Taking the product of the inclusion probability and the response probability, π, into account, a 'risk-index' can be formulated as

$$RI = \frac{i}{\pi} \sum_{j=1}^{J} e^{w_j \ln(d_j)} / J ,$$

with w_j a weight for deviation component j. For w_j we initially took the inverse of the standard deviation of $\ln(d_j)$. Weights were trimmed to improve the index's ability to predict all types of errors. The standard deviation of $\ln(d_j)$ can be obtained from last year's clean data. Median ratio's were obtained from unedited data, but can also be taken from last year's data, when most of current year's data is not yet received.



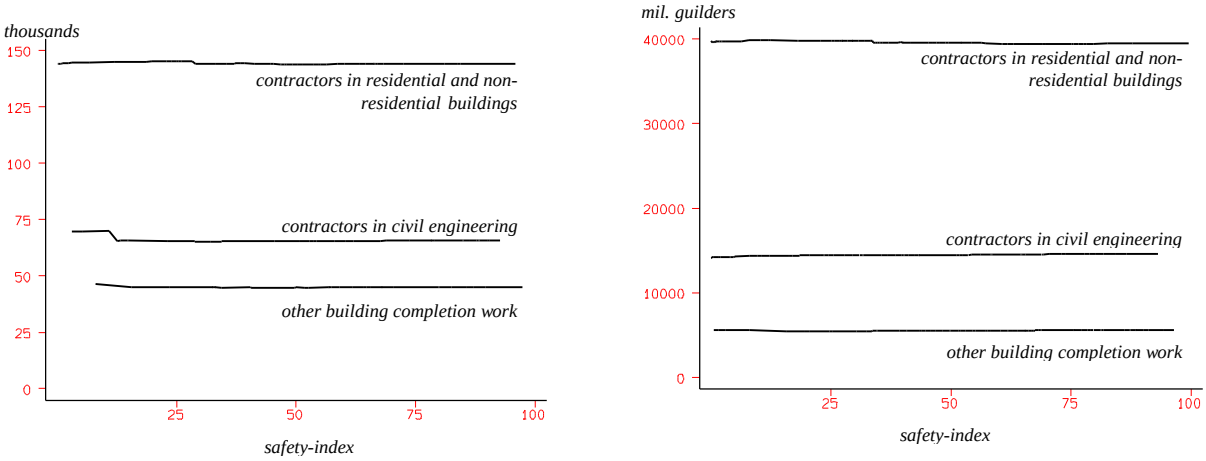
Next, because of peculiarities of Blaise 2, which will be described in section 4, we turned the risk-index into a 'safety-index', with value 0 indicating the highest possible risk, and value 100 the lowest possible risk, SI* = 100 - 100 RI / (1 + RI) . Moreover the index was transformed such that its value approaches a uniform distribution between 0 and 100. This has the advantage that editing all records between a certain value, say z, corresponds to editing approximately z% of the records. The safety-index is

SI = 100 - 100 c RI / (1 + c RI) .

Figure 4. Cumulative distribution of safety index

With $c = 1.6$ we got close to a uniform distribution, as is shown by the cumulative distribution plot in figure 4. With a uniform distribution it should show a straight diagonal line.

Figure 5. Edit effect on number of employees (left) and total production (right); edits



sorted by safety-index

Of course the size of each error is the best risk-index we could think of, but this size is unknown for unedited data. The safety-index however proves a sufficiently effective substitute in predicting which records have to be edited. This can be seen in figure 5, which is a repetition of figure 1, this time not with the number of edits (ordered by error size) on the horizontal axis, but with the safety-index instead. For brevity we do not show the same graphs for the other 10 variables that were inspected. The safety-index is as effective in predicting errors for those variables.

An overview of methods for rationalizing the editing of survey data is given by Granquist (1994).

4. How to give shape to selective editing with Blaise

Suppose we could assign the value of the safety-index to a key, with which Blaise gives access to records in order of the size of the key. Then subject specialists could edit records in order of error risk by requesting a record without any key-specification. The record with the highest error risk (lowest safety-index) would pop up first, the record with the one-but-highest error risk would be edited next, etc.

In Blaise version 2, records can be accessed with two keys, the primary 'key' and the 'selector' key(s). The primary key has to be unique and the safety-index is not, so this key cannot be used. Fortunately the selector key does not have to be unique, so we can use the selector key. When using the CADI machine, the subject specialist could use the 'locate forms' option. If he does not specify a selector value, Blaise will show the record with the lowest selector value. Data access in reverse order is not possible. This is why we formulated the risk-index as a safety-index: records with the lowest safety-index will show up first.

In order to access data in order of error risk, we must have the safety index computed first. When entering data in a CADI-machine, subject specialists should be able to tell the Blaise-program that all data of a record have been entered, and that the safety-index has to be computed. Ideally, we would like to have a function key for this (or a graphical button on the screen). Such functionality is not available, though. In Blaise 2, we used the <shift-F3> key instead, which invokes the evaluation of all error checks.

The computation of the safety-index has been linked to a dummy-variable, whose default-value is true. At the same time when Blaise error-messages are generated, the safety-index pops up on the screen. Records with a high safety-value, for instance larger than 50, should not be edited, even when the conventional Blaise-error messages tell that some details are

wrong. Correcting these details would not have any effect on publication figures, so it would be inefficient to correct them. After all, correction of 1000 records more would take another man-year. Only when the record is not safe, that is the index is lower than 50, the subject-specialist should edit the record.

Of course data entry can be carried out by people who are less highly qualified than subject specialist. In that case the data typist will evaluate the record with <shift-F3> and then save it, without editing. The subject specialist can access the most risky record later, by using the 'locate forms' entry of the main menu. As explained before, if he does not specify a selector key, the most risky record will pop up first.

At the Netherlands ACS forms used to be edited in order of day of reception. The most recently received record was edited first, because in case a respondent has to be called for clarification the respondent will bring back to memory more easily which figures he entered. Moreover respondents should not be demotivated by asking them questions many months after they returned the questionnaire. Therefore we presently try to take the best of both approaches. Recently entered forms are listed on paper in order of their safety-index and subject specialists will select high-risk records from that list. Then they will select the paper form from the pile of forms, which are sorted on day of reception, look up the firm-identification number, and call for the form on their CADI-machine via 'process old form'.

5. What to do with the forms that seem not to need editing

When a risk-index is used for prioritizing edits, some records will remain unedited, though they contain some slight inconsistencies. Moreover, in case of a long questionnaire, not all variables will be incorporated in the risk-index. One way to deal with remaining errors is to apply automatic editing. Automatic editing, not to be confused with computer assisted editing as is offered by Blaise, removes inconsistencies from the data. Fellegi and Holt (1976) developed a method to trace which field(s) cause a series of error messages. Their criterion is, in short, that as few fields as possible should be altered to remove the error messages. Once the erroneous fields have been designated, they can be replaced by imputed values.

In practice, efficient algorithms for this approach are hard to devise. A Windows- program which uses the Fellegi-Holt algorithm, like Lince (Informatica Comunidad de Madrid SA, 1993), will not handle much more than 30 error messages. An alternative algorithm, as is implemented in GEIS (Kovar and Whitridge, 1990), will only be fast for small datasets. If edit checks are restricted to ratios of two variables at a time, a simple and fast algorithm emerges. This algorithm has been implemented in SPEER (Winkler and Draper, 1994). At Statistics Netherlands we presently do research into these methods, which will hopefully become of use for all Blaise-users.

As these fully automated methods should not be used with very many error messages, we looked for simple additional methods. One possibility is to apply 'deterministic' corrections. Deterministic corrections can be applied in those cases where always the same correction is to be made in specific, well defined circumstances. In the ACS questionnaire for instance, some figures have to be subtracted from others to obtain their difference. The minus sign is printed in the questionaire, but some respondents will, erroneously, reproduce it in the answer box. Often these minus signs are, again erroneously, typed in. This will generate Blaise-error messages, which can easily be corrected automatically by testing whether deleting the minus sign will remove the error message.

Deterministic automatic corrections can be invoked in Blaise in the same way as the safety-index is computed. A dummy variable called 'auto' is specified, which has default value 'off'. The subject specialist can turn this variable 'on', and press <shift F3> to have automatic corrections.

Another deterministic correction was developed for those forms where part of the data is missing. In the ACS especially questions on production details are numerous. Some respondents only present figures on their production total, and skip the details. Verboon (1995) describes how automatic imputations perform, using cell means and last year's firm-specific data.

References

Boucher, L. (1991): *Micro-editing for the Annual Survey of Manufactures: What is the value added?* Proceedings of the U.S. Bureau of the Census 1991 Annual Research Conference. pp. 765-781

Fellegi, I.P., and D. Holt (1976): *A systematic approach to automatic edit and imputation*. **Journal of the American Statistical Association** 71, pp. 17-35

Granquist, L. (1994): *Macro-editing - A review of methods for rationalizing the editing of survey data*. In: United Nations Statistical Commission and Economic Commission for Europe: Statistical Data Editing, Vol. 1, Methods and Techniques (United Nations, Geneva)

Kovar, , J.G., and P. Whitridge (1990): Generalized edit and imputation system: overview and applications. **Revista Brasileira Estatística** 51, pp. 85-100

Hidiroglou, M.A., and J.-M. Berthelot (1986): *Statistical editing and imputation for periodic business surveys.* **Survey Methodology** 12, pp. 73-83

Informatica Comunidad de Madrid SA (1993): *Lince, Sistema de validación e imputation automatica de datos estadisticos*; manual de usario (ICM, Madrid)

Latouche, M., and J.-M. Berthelot (1992): *Use of a score function to prioritize and limit recontacts in business surveys*. **Journal of Official Statistics** 8, pp. 389-400

Lindell, K. (1994): *Evaluation of the editing process of the salary statistics for employees in country councils. paper presented at the UN congress on data editing in Cork, Ireland*. To appear in 'Statistical Data Editing, vol. 2' (UN Statistical Commission and Economic Commision for Europe, Geneva)

Van de Pol, F. (1994): *Selective editing in the Netherlands Annual Construction Survey. paper presented at the UN congress on data editing in Cork, Ireland.* To appear in 'Statistical Data Editing, vol. 2' (UN Statistical Commission and Economic Commision for Europe, Geneva)

Verboon (1995): *Editing subdivisions in the Annual Construction Survey*. Paper presented at the Bristol conference on Survey Measurement and Process Quality (Statistics Netherlands, Department of Statistical Methods)

Winkler, W., and L.R. Draper (1994): *Application of the SPEER edit system*. Research paper (US Bureau of the Census, Washington)

Developing a Multi-Mode Survey System

Roger Schou, National Agricultural Statistics Service, USA

1. Background

The National Agricultural Statistics Service (NASS) is an agency of the United States Department of Agriculture. NASS includes headquarters (HQ) in Washington, D.C., a research division in Fairfax, Virginia, and state statistical offices (SSO's) in 45 of the 50 states. HQ, with input from the SSO's, designs the national level surveys. The CASIC Group, in HQ, authors and maintains the data collection and data editing instruments for these national level surveys.

One of the major survey applications conducted is the Quarterly Agricultural Survey (QAS). This survey is conducted in March, June, September, and December, in all states except Alaska and Hawaii. Information collected on the QAS includes crop acreage and production, grain stocks stored on farms, and hogs. Related livestock surveys, conducted in January and July, collect data on cattle, sheep, and goats. The particular questions used to collect this information varies state to state, as different parts of the country raise different crops and livestock.

A sample is drawn for each state, and every sampled unit results in either a completed interview, a refusal, or an inaccessible. Since the sample is known before the survey begins, the Blaise dataset is initialized with the sample information for each contact.

There is approximately four weeks from the first day of data collection until the publication of the estimates. The time allowed for data collection and interactive editing (IE) is only the first 14 to 18 days of the one-month period. Due to the restricted time limitations, NASS uses the following modes of data collection:

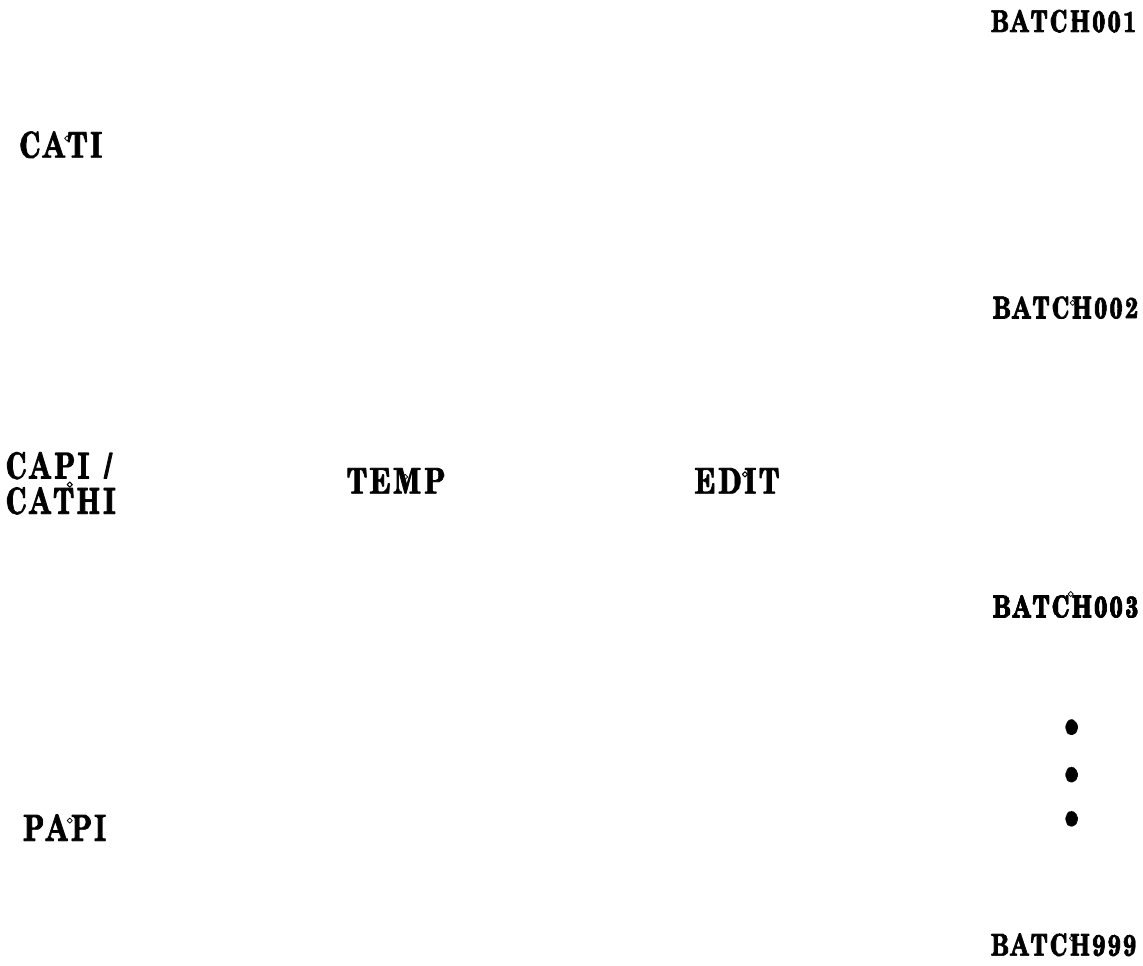
CATI The telephone enumerators (or interviewers) use the prepared CATI instrument and the call scheduler to make contacts from the office.

PAPI FIELD enumerators will use paper instruments for all contacts unless they are fortunate enough to be supplied with a laptop. Currently, only one of our SSO's field staff is equipped with laptops.

MAILing paper questionnaires is another mode of data collection used, if a sufficient response rate is achievable. Respondents may return 20-30 percent of the mailed questionnaires. Use of mail varies from state office to state office.

CAPI / CATHI The field enumerators, who have been supplied with a laptop, use a prepared CAPI instrument to conduct the personal interview. CATHI is an acronym for Computer Assisted Telephone Home Interviewing. In other words, the field enumerator conducts a telephone interview from their home.

Sample sizes may be rather large for the short window for data collection and editing. Accounting for such a large volume of questionnaires in a limited time, the survey system must be carefully designed to handle the flow of data through the cycle. This system must handle any of the modes just described.



The following diagram pictorially describes the flow of data through the data collection and interactive editing phases, with Manipula playing a major role. All of the areas, except the PAPI area, represent a Blaise dataset during the survey process.

2. The Instrument

Each survey has one instrument designed to do both the automated data collection and the IE. The coding of this data model can be challenging at times, because of the varied data collection modes. If a form enters the process through PAPI, the only fields input are ones with tags. In other words, all of the screening-type questions must be off the route for data collected on PAPI. Also, certain edit checks pertain only to data collection, while other checks are exclusive to IE.

3. Initialization

The entire sample is initialized into the "CATI" dataset at the beginning of the survey, before data collection begins. The batch files, that run the whole survey process, are written as generically as possible, so they are useable across many surveys. In order to accomplish this, the batch file passes the survey directory as a parameter, using DOS environment variables. The two main DOS environment variables are 1) DATANAME which is the name of the Blaise dataset, and 2) INSTNAME which is the name of the Blaise meta-file.

The initialization process requires four input files. The first file is the Target file, which is the names, addresses, phone numbers, and other identification information. The second is the Partner file, which is the same type of information as the Target file, but identifies the partners, if any, involved with the Target. The third file is the Comment file, which contains any pertinent comments related to the Target. The fourth input file is the Sample Master file, which contains the sample and historical information for every record that must be accounted for during the survey. With Manipula, these files are sorted, matched, and the data is output to the "CATI" dataset.

The United States spans seven time zones, and most of the country changes to daylight-saving time in the spring and summer. A Manipula setup uses information such as state, county, zip code, and time of year to determine the correct time zone for each form as well as the time zone for the SSO. This setup runs during the initialization process, but it may be run again to reset the time zones, if the survey period includes the day that daylight-saving time begins or ends.

Since NASS surveys are rarely 100% CATI, a process was developed to make the form unavailable to the call scheduler. The process is called CATI/Non-CATI transactions. The user prepares an ASCII transaction file consisting of the Blaise key and a constant "2" to withhold from CATI and "1" to reactivate CATI. This file becomes an input file to a Manipula setup which updates the TCatiMana block. If the RegsCalls array (registered calls) is not empty, the setup shifts the information in this array as if another call had been made. It then sets the five fields in RegsCalls[1] to reflect the "Non-CATI" call. The high-level block name is CatiMana.CatiCall.RegsCalls[1]. The following five fields are updated to withhold the form from CATI:

WhoMade := 'SUPERVISOR'

DayNumber := computed from the SYSDATE and CatiMana.CatiCall.FirstDay

DialTime := SYSTIME

NrOfDials := 1

DialResult := 8 (Other)

This same process can reactivate the form for CATI. The only difference is the constant input on the ASCII file, and the computation into DialResult is 3 (Busy).

4. Data Collection

The combination of data collection modes varies by survey and by SSO. The March 1995 QAS was the first NASS application to use Blaise III, but its only use was for data collection. The June 1995 QAS was the second NASS application and it used the whole survey process. It is best suited to illustrate the multiple modes of data collection. Indiana, Colorado, and Wyoming, designated as the three Blaise pilot states, conducted these surveys.

The June 1995 QAS in Indiana used all of the illustrated modes, thus utilizing all pieces of the survey process. As mentioned earlier, the entire sample was initialized into the "CATI" dataset. Indiana also mailed paper questionnaires to their entire sample. Prior to the start of the survey, they identified the forms that the field enumerators would contact using the laptops. The corresponding forms in the "CATI" dataset were withheld from the call scheduler using the CATI/Non-CATI transactions. A bug was detected in the call scheduler when more than ten enumerators were on the system, so Indiana disabled the call scheduler and tracked the appointments on Manipula-generated paper call sheets. This, however, did not affect the survey data flow, because the in-office CAPI stored the data in the "CATI" dataset. The other two Blaise pilot states continued to use the call scheduler, as their interviewer staff is smaller.

In the other two pilot states, the field enumerators conducted all their interviews on paper questionnaires, and in Indiana, some respondents returned paper forms. As the paper forms arrived in the office, they were checked-in using a procedure similar to the CATI/Non-CATI transactions. The check-in process minimizes the chances of contacting a respondent after receiving a PAPI form. The check-in transaction makes a form unavailable to the CATI call scheduler, and the instrument contained a CAPI only screen to alert the enumerator not to call this form, as it was already in the office.

5. Batch Edit

In order to minimize the interruptions to the CATI data collection, the system allows completed forms to periodically move from the "CATI" dataset to the "EDIT" dataset. This keeps all editing processes outside the domain of "CATI." Batch edit is the procedure designed to do this. It sounds like an easy task, but this is probably the most complicated part of the whole survey process. The general concept is that on the office Local Area Network (LAN), a given form may only exist in one dataset at a time. This eliminates the risks of duplicating the data, and losing control of the data.

The first step in the batch edit is to move (not copy) the forms completed with CATI from the "CATI" dataset to the temporary dataset ("TEMP"). The batch file prompts the user for a "CATI batch run number" which Manipula computes into a field when it writes the form to "TEMP." Discussion of the use of this "batch run number" appears later.

The second step checks for a dataset in the CAPI/CATHI directory. If a dataset exists, it moves the forms from the CAPI/CATHI dataset to "TEMP" and deletes the corresponding form from "CATI." The batch file prompts the user for a "CAPI batch run number" as with the CATI data. During this move, it also checks "EDIT" for any completed mail-returned forms. If it finds a match, it generates a report of CAPI/CATHI forms that overlaid PAPI forms. The paper form would still reside in the office, so if, during the edit, the user decided the data on the paper form was better than the CAPI/CATHI form, they could enter the data through IE.

The third step involves reading in the code data. The paper forms are keyed into batches in a software called Key Entry III. The resulting file contains a key and sets of item code/item data pairs. A series of Manipula setups is run to get this data into "TEMP." One Manipula setup writes the Blaise keys that exist in "CATI," "TEMP," and "EDIT" to an ASCII file of eligible keys. Another setup matches these keys with the keys in the code data file. Any overlays and non-matches in the code data file are reported, and set aside in another file for later processing. Yet another Manipula setup moves the Blaise forms that have a corresponding record in the code data file from their current dataset to "TEMP." The Code Data Converter, a Manipula setup written by Mark Pierzchala and myself, converts the code data directly into the Blaise dataset. The batch file again prompts the user for a "code data batch run number," which Manipula reads in, after the code data is read into "TEMP."

The fourth and final step of the batch edit moves all of the forms in "TEMP" to "EDIT." The process includes running an integral check on the forms in "TEMP" just before moving them to "EDIT." During an integral check, the instrument performs certain computations conditioned on data source. Also, there may be a difference in FORMSTATUS between a completed CATI interview and one that is reviewed in IE (or integral checked in CADI), since some of the edits in the data model are conditioned on CADI. As of the June 1995 QAS, the integral check was not available. Once the forms are in "EDIT," they are ready for IE.

7. Interactive Editing

Interactive editing (IE) may occur directly in the "EDIT" dataset. It may also occur in a subset dataset of "EDIT." A process that uses three-digit batch run numbers entered during the batch edit, allows the editor to edit forms all collected in the same mode. During this subset-creating process, the batch file prompts the user for a batch run number. The batch file creates a directory named BATCH###, where ### is the batch run number. Then it moves the forms in "EDIT," that have this number stored in the form, to the "BATCH" area, and executes the DEP in edit mode. If this directory already exists, the batch file does not create the directory, but immediately executes the DEP in edit mode. When the user exits the DEP, the batch file prompts them to see if they want to move the forms back to "EDIT" or leave it there for future editing.

All forms reside in "EDIT," after completing data collection and IE. From here, the data is ready for conversion to the appropriate format for summarization. NASS uses SAS to summarize some surveys on the LAN, and the mainframe for others including the QAS. The QAS data, converted from Blaise to code data using the Code Data Converter, runs through a mainframe batch edit system (SPS Edit) prior to a mainframe SAS summary. All SSO's have access to an IBM Mainframe located in Orlando, Florida, on which NASS contracts for processing time. This mainframe batch edit system is what the Blaise edits mimic, so the SPS Edit detects only a minimal number of critical errors. As the Blaise QAS develops, the goal is to catch all of these errors in Blaise, and eventually eliminate the SPS Edit.

8. Conclusion

The key to making this whole system work smoothly was Manipula. The fact that Manipula can create and update a Blaise dataset, helped streamline the survey data flow as summarized in this paper. Although handling and tracking the multiple modes of data collection was a challenge, this system proved to withstand the strains of a production survey.

From Products to Systems: Addressing the Needs of CAI Surveys at Westat

James E. Smith, Westat Inc., USA

1. Introduction

Computer-assisted interviewing (CAI) and computer-assisted survey information collection (CASIC) have become important buzzwords in survey research. They represent a wide range of methods for survey data collection using microcomputer and telecommunications technologies that replace traditional paper-and-pencil techniques.

Today, the predominant CAI methods are computer-assisted telephone interviewing (CATI) and computer-assisted personal interviewing (CAPI). Each of these methods relies on interviewers (telephone or in-person) who ask questions given by a computerized questionnaire. By entering responses directly into the computer, interviewers eliminate separate data entry operations at home office. In addition, as responses are entered they are checked for errors and inconsistencies, thus avoiding later re-contacting of the respondents. These advantages combined with its ability to simplify the administration of complex questionnaires, have made CAI the method of choice for many kinds of surveys.

Although interviewing with a computerized questionnaire is at the heart of CAPI and CATI surveys, it is not the whole survey process. Every survey practitioner knows of many other essential activities that make up a whole survey project, such as designing and drawing the sample, determining survey content and questionnaire wording, planning and executing data collection logistics, and various post-collection data processing tasks like sample weighting, sampling variance estimation, creating analytical variables and files, and documenting and disseminating the data. Balanced attention is needed to all of these processes for a survey to be successful in its ultimate goal of collecting and delivering useful information to its intended audience.

After giving some background on CAI systems at Westat, I will comment on our CAI systems integration perspective applied to planning, acquiring, and developing CAI capabilities. Two examples from ongoing surveys will illustrate the importance of systems that support non-interviewing work in CAI surveys, showing the need for better product inter-operability in CAI systems.

2. CAI Systems at Westat

Westat Inc. is an employee-owned research corporation conducting statistical research and related services to agencies of the U.S. Government and other clients. Over the past decades, the course we have taken as survey practitioners using computer-assisted survey methods is similar to other survey organizations. Beginning in the late 1970's, as minicomputers became

more available and powerful, we programmed a CATI system on a multi-user VAX system using readily available screen handling and file handling capabilities of that system..

During the early 1980's this first-generation CATI system was replaced by a new generation of CATI software called Cheshire. This software, which includes several components, was designed and developed in-house to meet our growing needs to perform larger and more sophisticated CATI surveys. Cheshire treats a CATI questionnaire in three integrated parts: flow and computation expressed in a PL/1-like questionnaire authoring language, interviewing screens held in a screen library and called up as needed by the questionnaire language, and a data dictionary defining a hierarchical ("rostering") database used by the questionnaire language.

A Cheshire questionnaire is executed under the control of a run-time program that handles common interviewer navigational and operational tasks, such as backing up, breaking off, restarting, producing audit trails, and interfacing with the operating system and hardware for screen handling and disk access. In a CATI survey, a separate call scheduling program delivers cases to interviewers and performs call prioritizing, work allocation among interviewers, call status tracking, and call processing and scheduling functions. The Cheshire system with the call scheduler has been used in a wide range of CATI surveys from small to large, and from simple to very complex in their questionnaire and operational requirements. The Cheshire system has also proven useful for a number of non-CATI applications such as on-line receipt control and computer-assisted data entry (CADE).

During the 1980's, in addition to rapid developments in PC's themselves, other related developments were important for CAI surveys, particularly the evolution of PC networks and the emergence of portable computers. PC networks opened the door for PC's to replace minicomputers as host systems for CATI surveys, and portable computers permitted computer-assisted interviewing to be carried into the field as CAPI. During this time the run-time component of our Cheshire system was converted to work on DOS-based PC's in anticipation of cost-effective laptop computers and CAPI. Today we conduct CATI and CAPI surveys with the Cheshire system on VAX and PC computers, respectively. Our current systems efforts are aimed at the Microsoft Windows environment with network, database server, and telecommunications technologies for CAPI and PC-CATI..

In general, developments of our CAI systems have been driven by requirements for system capabilities that were otherwise unavailable at the time. For example, several key developments in the first Cheshire system were precipitated by survey requirements that were a step ahead of ours or any other CATI system's reach. The practice of targeting our CAI systems development efforts at particular survey needs as they arise has helped to ensure that the right capabilities are developed at the right time, while also giving into the development process strong technical focus and a clear understanding of who the intended users are.

While we find it essential to continue with this approach to CAI systems development, we also recognize that this approach does not completely address all desirable CAI systems changes and developments for the long term. Therefore, we find it important to plan for CAI systems changes and developments in response to specific survey requirements, while at the same time planning for farther reaching changes, including new technologies and new approaches in CAI.

3. Our Systems Interactive Approach

A key perspective in our CAI system planning, acquisition, and development is to conceive of CAI systems as integrated systems comprised of multiple products that work together. In adopting this perspective we have been encouraged by the fact that today's computer software products often adhere to defacto standards, such as the Xbase database format, that allow some level of compatibility between products. At a more sophisticated level, PC operating systems are now supporting and encouraging inter-operability between products through standards such as Object Linking and Embedding (OLE) in Microsoft Windows. Emerging "plug and play" hardware/software standards also reinforce the trend toward product inter-operability. While we are far from seeing universal compatibility and inter-operability across software products, inter-operability is the goal toward which modern software products and operating systems are headed.

Thus, from our survey practitioner's perspective, it is increasingly important for the CAI software we use to perform well at certain CAI tasks, *and* to perform these tasks with a high degree of compatibility and inter-operability with other potential CAI software. There is an instructive analogy in survey statistical data processing. Here the database handling capabilities of database packages and the analytical capabilities of statistical packages are both needed, and different products excel in each domain. While the data processing for some surveys, particularly relatively simple or smaller surveys, might be handled effectively only within a statistical package despite its limited database handling capabilities, or only within a database package despite its limited statistical analysis capabilities, most major survey applications cannot be served in this way. Thus, what has emerged are statistical packages that can process data residing in database systems using, for example, SQL queries executed from the statistical package. Similarly, a single CAI system that tries to be provide a full range of capabilities needed in a major statistical survey, including complex instrument design, instrument execution in a production environment, and post-collection data processing and delivery may end up trying to do too much, and therefore doing too little for many surveys.

4. Example: Specifying a CAPI Instrument

A computer-assisted questionnaire needs to be specified before it can be implemented. Only in the limiting case of a single researcher who sets up his or her own questionnaire directly in a CAI system can a survey avoid a specification process that involves dialogue, communication, and iteration over questionnaire specifications. A large CAI instrument, for example the family medical survey CAPI questionnaire in the National Medical Care Expenditure Survey, may involve over thousand questionnaire items, many of which are complex rosters, grids, or compound items with overlay screens.

As challenging as implementing this questionnaire in a CAI system is, perhaps the greatest challenge is getting the instrument specified to the point that it can be implemented. This process begins with content determination and general design, including chunking the content into manageable sections based on numbers of items, major skip patterns, cognitive

considerations, etc. Specification then proceeds to the item level using standard item types -- yes/no, pick-all, roster, grid, etc. -- wherever possible. Wording is supplied for questions and responses, and response categories are determined. Skips and other flows, like loops, are specified, as are range and consistency checks to be performed during interviewing.

While it may be possible to incorporate all of these types of specifications directly into the authoring language of a CAI system, we are not aware of a CAI system that fully supports the full questionnaire specification process that involves multiple questionnaire authors, numerous reviewers, the circulation and assimilation of comments from multiple sources, and the incremental building of all the necessary specification elements during this iterative process.

Our approach has been less than ideal, but workable. It is to provide questionnaire authoring system separate from the CAI production system but able to integrate with it. By entering specifications into the authoring system, current specifications are maintained in one place (but with network multi-user access). In addition, during the specification entry process, numerous problems, such as incomplete or confusing revisions, overlapping response categories, and impossible skip instructions are detected, the latter automatically by the authoring system. Various hardcopy or electronic document outputs are obtained from the authoring system to facilitate further iterations in the specification process. Among these outputs is a fully formatted "hardcopy questionnaire" that incorporates most of the specifications into a format familiar from hardcopy surveys. As a final specification, the authoring system generates screen definitions, database information, and other information in files that are directly loadable into the CAI system, or are usable in hardcopy form to implement the questionnaire in the CAI system.

This experience suggests to us that the process of specifying a large and complex CAI instrument, benefits from an authoring system that specializes in managing the specification elements and facilitates iteration, communication, and revisions in this process. At this time, it does not seem practical that the same CAI systems which provides run-time support for questionnaire execution in the field could, or would want to, carry all the baggage of front-end questionnaire specification support. Moreover, many nuances of implementing an instrument in a CAI system are extraneous during the specification stage, and should not be forced on the specification team. Therefore, the development of specialized front-end authoring systems, or at least tools, that are inter-operable with CAI systems is desirable.

5. Example: CAPI Data Transmission

The data in a CAPI survey does little good if it doesn't arrive from the field in a timely way. While mailing floppy disks may serve as an emergency backup procedure, CAPI surveys rely heavily on electronic data communications to move data. In many CAPI surveys, such as the multi-round National Medical Care Beneficiary Survey (MCBS) which maintains more than 200 interviewers in the field almost continuously, data transmission involves more than getting completed CAPI interviews to home office. Field status information, case reassignments, questionnaire updates, software updates, and various housekeeping chores are also supported by the data transmission system.

Currently, MCBS and other Westat CAPI surveys use the telephone system as a "wide area network" connection between CAPI field interviewers and home office. (Other value-added networks have some potential here, as does the Internet eventually, but these are not yet cost-effective or sufficiently widely available CAPI surveys.) Some key requirements of our surveys for a system to support data communications through the phone system are ease of use by interviewers, error detection and correction with understandable warning and error messages, complete transaction logging and communications audit trails, programmable rules to accommodate various project needs and changing needs within a project, and sufficiently fast and error-free data transmission.

An earlier CAPI system which we used involved a popular modem communications package with script capabilities. Although operationally acceptable for some time, this system fell short of providing several of the key requirements to the level of a more sophisticated commercial product which is now in use. This product resides on a communications server at home office and operates under the control of project-specific scripts which specify general and interviewer-specific file transfer rules. The client portion of the system resides on interviewer laptop computers in the field and performs automatic dialing, error correction, and retry attempts to reach the home office server. When connected, the client operates under control of the home office server.

In practice, implementing this seemingly straightforward communications process has raised numerous problems concerning the interaction between communications software and modems, idiosyncrasies and surprising incompatibilities between field and home office modems, and wide geographic variations in telephone line quality. While none of these problems have proven insuperable, and none have caused data corruption or loss, they have demonstrated to us that specialized data communications systems which can anticipate and handle these issues are essential. This seems the only way to provide CAPI interviewers with a turn-key communications system that isolates interviewers as much as possible from the details of the communications process.

A systems integration issue that was faced early in the implementation of CAPI communications was the consistency between the look-and-feel and operation of the CAI system and the communications software. The added training and support costs for teaching interviewers about a different user interface in the communication system, along error handling, would have been high, and would have distracted interviewers from their principal task of learning the CAPI questionnaire. Hence, a high level of integration of the communications component with the CAI system was desirable. This has been accomplished in part by invoking the communications software in the standard CAI system menu, and then operating the communications system with pre-stored commands. However, messages and reports from the communication system are seen by the interviewer and must be handled through the communications software interface. Here is a case where some product integration has been forced, but hopefully will become more graceful under newer software and operating system standards for inter-operability.

6. Conclusion

Methods for computer-assisted interviewing (CAI) and systems that support CAI are becoming increasingly important in survey research. While discussions about these methods and systems often focus on interviewing, a CAI survey includes other important processes that must be successfully carried out to design and complete the survey. For major surveys, I suggest that systems which integrate multiple products into an overall CAI system will be preferred.

Use of Blaise and Manipula in the Annual Survey of Employment and Wages

Leo van Toor, Statistics Netherlands

1. Introduction

This paper describes the use of Blaise and Manipula in the Annual Survey of Employment and Wages of Statistics Netherlands. The survey covers approximately 75,000 establishments and 1,000,000 employees. There are two questionnaires. For each establishment there is one Employment questionnaire. This questionnaire consist of 10 questions (e.g. number of male and female employees, and for each municipality of business the number of employees). The second questionnaire is the Wage questionnaire. This questionnaire has 28 questions at the employee level (e.g. year of

birth, sex, and wage). Forms must be filled in for a sample of employees.

One of the objectives of the survey is to determine accurate statistics on the development of the number of employees (per municipality), and the development of the average wages. Therefore, it is important to carry out checks comparing figures from the current and the past year.

Since the two types of questionnaires have different return dates, two separate survey processing systems were built: an Employment and a Wage system. The global structure of both systems is approximately the same. Therefore, they are described in this paper as if they are one system.

To reduce maintenance problems as much as possible, it was decided to build the systems using standard software components, Blaise 2.5 being one of them. The developers of the system were confronted with two major problems:

1) Due the use of many external files, and extensive computations in the check section, Blaise ran into memory problems,

2) Many establishments preferred sending statistical information on tape or diskette over having to fill in paper forms.

At the same time, the Department changed its ideas about data editing. Instead of spending many resources on cleaning up each form, its was felt that editing should concentrate on only correcting errors having an impact on the quality of the published statistics. Furthermore, wherever possible, automatic editing should be implemented. In this way, human involvement in data editing activities could be reduced without it having a negative effect on data quality.

In short, the main objective for redesigning the system was faster survey processing with less people, thus improving timeliness and reducing cost.

The consequence of this approach was less focus on interactive, form oriented procedures, and more emphasis on batch-wise, file oriented procedures. Manipula was selected as the standard tool for these batch procedures. In this way, Manipula became the central part of the new survey processing system.

For batch systems, it is important to have progress and status reports. So-called 'system report files' must be produced containing information about things like execution time of modules, and number of processed records. It is important for the 'system manager' to analyse these reports before taking any further action.

The ideas about automatic editing emerged after the survey processing system was almost ready. Fortunately, the modular and flexible architecture of the system made is possible to include automatic editing modules, without getting problems with the other, already present modules.

The survey processing system had to produce a large number of tables. The size of these tables, the type calculations required, and the nature and amount of subtotals, made it impossible to use Abacus. For example, Abacus can calculate totals and averages, but not the ratio of two totals, like the average wage per hour (sum of wages divided by sum of hours). Also the table-size was beyond the limitations of Abacus. Manipula turned out to be the way out. For tables, Manipula is not as user-friendly and fast as Abacus. However the powerful Manipula language can be used to define standard setups that can be applied in many situations.

This paper describes the most important modules of the survey processing system. Not all details are mentioned, since that would ask too much knowledge of the Dutch situation. The most interesting aspects of the system are:

- Only standard-components were used: Manipula and Blaise,
- All components were tied together in MS-DOS batch files,
- The system is controlled through NOVELL-menus,
- Every module in the batch system produces a system report file,
- Manipula is used for checking the data,
- Tables are produced with Manipula using standard setup,
- An automatic editing module is implemented.

The survey processing system is described in more detail in the subsequent sections. Section 2 gives an overview of the system, from data collection to tabulation. Section 3 handles the system report files. Section 4 gives more details about the implementation of the auto-edit module. Section 5 describes the use of Manipula for tabulation. Section 6 gives some concluding remarks. More information about the details of the survey processing systems can be obtained by contacting the author.

2. Overview of the system

This section gives an overview of the various modules in the system. Where possible, the description of the modules is given in the order in which they are activated in the system.

2.1. Data collection

There are two sources of data: paper forms on the one hand, and tapes/diskettes on the other. The data on the paper forms are entered with a Blaise CADI program. The work is done by data typists with little knowledge of the subject-matter of the survey. Only simple checks are carried out by the CADI program. There are range checks, and checks that consult external files to determine the existence the establishment identification number, and the municipality code. The reasons to limit the editing in this stage of the process, are

- The data-entry process is faster;
- Focus is on errors having an impact on data quality, and checks for such errors have to be carried at a later stage,
- The ambition to use as much automatic editing as possible,
- The difference in knowledge and experience between data typists and subject-matter analysts.

The tapes or diskettes containing data are translated into ASCII subfiles with Manipula, after which these ASCII-subfiles are converted and added to the Blaise data file with the Blaise conversion tool Convert.

After all paper forms, tapes, and diskettes have been processed, the result is a raw data file. The next step is data editing.

2.2. Data checking, phase 1

The raw Blaise data file is checked by the same Blaise CADI program as mentioned in section 2.1. The data structure is the same as that of the data entry program. The CADI program is run in batch mode (with the command line parameter /C), so an integral check is performed on the data file. As a result the status parameter of each form is set either to *CLEAN* (no error), *SUSPECT* (only soft errors), or *DIRTY* (hard errors).

The data model underlying the Blaise survey specification, is an hierarchical model with three levels:

- The first, and highest level contains information at the level of the establishment,
- The second level contains information at the level of the establishment in a specific municipality,

- The third, and lowest, level is the employee level.

To be able to handle the information at the various levels, the Blaise data file has a subfile-structure corresponding to these levels.

The Blaise conversion tool Convert is used to export the clean records to ASCII files, and to delete this records from the Blaise data file. The dirty records remain in the Blaise data file. Convert is more suitable for this job than Manipula, because Convert it able to keep the subfile structure. Furthermore, it is much easier to use Convert to delete the clean records from the Blaise data file.

The disk space allocated to the deleted records is not released by Convert. As a consequence, the file size is not reduced by this operation. The Blaise tool Formman is used to clean up the Blaise data file.

The Blaise data file with the remaining dirty records is processed by the data-typists using the Blaise CADI program that was used to enter the data. Of course, no data is entered at this stage, but available data is corrected.

The clean records go to the next step in the survey process, and that is the computation of aggregated records.

The aggregation step consists of using the employee level data to compute figures at the level of the establishment in the municipality. Examples of aggregated figures are

- Number of male and female employees, and the total number of employees per municipality,
- Number of employees per age group,
- Average wage per hour, and total of wages per year.

Furthermore, these figures are linked to the corresponding figures from the previous year.

Aggregating and linking is carried out in a single Manipula run. Also, a number of checks are carried out in this run. The checks can be divided in two groups. The first group of checks deals with the data from the previous year. Examples of these so-called level 1 checks are:

- The sum of the number of male and female employees must be equal to the total number of employees,
- The sums of the numbers of employees per municipality must be equal to the total number of employees.

The second group of checks compares data from the two years. Examples are:

- The change in the number of employees must be within certain bounds,
- The ratio of the current average wage and the previous average wage must be within certain bounds.

The Manipula setup takes advantage of the possibility to create more than one output file. One output file contains the linked figures of the current and the previous year, and the other output file lists the error codes per establishment.

At a later date, also the auto-edit module was included in this setup. The auto-edit module exist of separate Manipula-setups. More details can be found in section 4.

Other Manipula setups were used to link the data records in the different ASCII sub-files with the corresponding error codes. The linked records (i.e. records with one or more error codes) are written to the suspect-data file, the other (not linked) records were written and added to the clean data file, which is a file of the format INDEX. The suspect data records are converted from ASCII to Blaise with Convert, and must be handled by a Blaise data editing program. This program differs from the Blaise program mentioned in section 2.1.

2.3. Editing current year's data.

Current year's data is edited when the comparison with previous year show too large changes. Analysts use another (the second) Blaise CADI program for these editing activities. The previous year's data is made available through external files. The analysts work through the file with forms that have been marked as suspect by Manipula. The problem description produced by Manipula, is also available in the Blaise program. The analysts have three possibilities to solve the problems:

1) Edit the record, i.e. change a value in a current year's field,

2) Enter an 'OK code', indicating that nothing is wrong. In this case, they also have to enter an explanation,

3) Enter a code indicating the previous year must be corrected.

2.4. Data checking, phase 2

The people handling this phase of the system, are almost the same as those mentioned in section 2.2. (data checking, phase 1). The differences between these two systems are:

1) Now it must be taken into account that the data from the previous year can be changed (section 2.5),

2) It may now happen that the an establishment is suspect, but the analyst has given it an 'OK code' and 'error explanation'. This concerns changes in figures between years, and not consistency checks within a form,

3) The analyst may now want to edit records from the previous year.

All these aspects are taken care of in phase 2 of data checking.

In integral check is carried out on all records (with the command line parameter /C). The same Blaise program is used as in section 2.1. The reason to do this integral check is to be sure there no accidental errors are introduced during the data editing process (e.g. value range errors). To make this integral check possible, extra variables (OK_CODE, EXPLAN) are defined in the Blaise program. These variables have the attribute HIDDEN. In this way it is possible to do an integral check on other Blaise files with the same program.

The 'clean' records are converted to ASCII and deleted from the Blaise file. The 'dirty' records remain in the Blaise file. This is all done with Convert. Normally there are no records left in the Blaise-file after this phase.

Manipula is used to select the records with the code 'CORRECTION PREVIOUS YEAR' from the data file of the previous year. Then Convert is used to convert these records to a Blaise file. The treatment of these records is described in sections 2.5 and 2.6.

The Manipula setup of section 2.2 is used to carry out calculations, and to link the figures. The only differences are that previous year's records may have been edited (see section 2.6), and that the remaining suspected data must be handled again with the data editing program (for the current year). Switches in the Manipula setups take care of the different use of these setups.

2.5. Editing the data of the previous year

Comparing figures of the current with those of the previous year is only useful if it can be assumed that the figures from the previous year are correct. So, where possible, checks are carried out on previous year's data, and where errors are detected, they are corrected. The selected and converted data from the previous year are edited by analysts with a Blaise-program (the third). Note that the previous year's data is not confronted with the data of yet an earlier year.

2.6. Data checking, phase 3

After the data from the previous year has been edited, an integral check is carried out on the forms to make sure the status of each form is updated. The program Convert is used to convert 'clean' records into ASCII, and to delete these records from the Blaise file. The 'dirty' records remain in the Blaise file. This remaining Blaise-file is cleaned up with Formman, and then handled again with the data editing program (of section 2.5). The 'clean' ASCII records are translated into the a Manipula INDEX file. Then the data checking phase 2 module is used for calculating previous year's figures.

The phase 2 data checking module uses the corrected previous year records instead of the original records. The advantage of this method is that by making a copy record and editing

this copy record, it is afterwards possible to see which records, and how many records from the previous year were corrected. It is also possible to undo corrections, if necessary.

The process mentioned in the sections 2.1 to 2.6 is repeated as many times as necessary to obtain a sufficient amount of response to tabulate.

The activities in the sections 2.2, 2.4 and 2.6 are managed by the 'SYSTEM MANAGER'. He activates systems, and judges their progress. He has a sufficient amount of subject-matter knowledge to be able to make informed decisions.

2.7. Weighting

After the clean data file is ready, weights have to be computed in order to make the sample representative for the population. A file with population frequencies of a number of auxiliary variables is available for this purpose. These calculations were made with Manipula.

Also some data manipulations are carried out during this phase. For example, ISIC codes are attached to establishments, and birth dates are transformed into ages.

A new data file is created, the so-called the standard file. This file forms the basis for computing population estimates.

2.8. Macro checking

The estimates based on the standard file are still not ready for publication. First, preliminary estimates are computed. These estimates are used to carry out some checks at the macro level.

Manipula is used to compute estimates of population statistics for the current and the previous year. Examples of these statistics are the number of employees and average wage per hour in each 'cell', where a cell is a combination sex, age and economic activity.

The computed statistics of the current and previous year are brought together in one file, after which Manipula computes changes. Extreme changes are marked as being suspect. In this case, all relevant data about the cell are written to a print file.

Every extreme change has his own 'suspect code'. The cell data and the suspect codes are written to a second output file. These suspect codes (on the cell level) corresponds with one or more error codes, which are assigned on the level of the establishment. The link between these errors and the suspect codes is recorded in an 'error relation file'.

The 'error relation file' is used to select all records with an error code corresponding to a suspected cell in the 'clean' data file. With Manipula, the selected records are deleted from the 'clean' data file.

The data editing and data checking processes that follows, has already been described in sections 2.2 to 2.6.

3. System report files

All modules in the batch systems generate system report files. These files contain important information about the progress of the process. The information should help in solving problems that may occur. Each system report file contains information about the time the module was started, the execution time, and the number of processed records.

In the following example, the system report file has the standard name DAY.PRN, and it is controlled by an MS-DOS batch-file. The general structure is displayed in figure 3.1.

Figure 3.1. General structure of the system report file

```
ECHO START SYSTEM: xx  > DAY.PRN
SYSTIME               >> DAY.PRN
MAP n:=.....          >> DAY.PRN
MAP INS s1:=.....     >> DAY.PRN

     ECHO information >> DAY.PRN

     MANIPULA ..... D=DAY.PRN

     CONVERT  PF=n:x.cvt
     COPY DAY.PRN + n:x.DAY
     DEL n:x.DAY

     FORMMAN  PF=n:y.frm
     COPY DAY.PRN + y.DAY
     DEL y.DAY

     NPRINT   ..... >> DAY.PRN

     COPY x + y z   >> DAY.PRN

MAP DEL n:          >> DAY.PRN
MAP DEL s1:         >> DAY.PRN
SYSTIME             >> DAY.PRN
ECHO END SYSTEM: xx >> DAY.PRN
NPRINT DAY.PRN
```

The user has no complete control over the names of the files produced by Convert and Formman. Therefore, MS-DOS batch commands are required to add information generated by these tools to the system report file. Unfortunately, the integral check with CADI does not produce any process information at all. If necessary, such information can be generated with Manipula.

4. An example of auto-editing.

Analysts correct many small errors without contacting the establishments concerned. If errors can be corrected in such a relative simple way, it is worth while considering doing the corrections automatically. Moreover, many of these corrections do not seem to have a substantial impact on the accuracy of the produced statistics.

To be certain that automatic editing does not make things worse, the results have to checked. The general rule is: after auto-editing, there must be less errors than before. If this is not the case, the auto-editing must be cancelled. Such a rule can be tested with existing modules in the survey processing system. The module(s) determining the error codes is carried out twice, one before, and once after auto-editing. By comparing the results a decision can be taken whether to accept or reject the auto-edit.

Here is an example of an auto-edit. If the difference between the total number of employees and the sum of male and female employees is less than 10, an auto-edit is carried out. The Manipula specification of this edit is shown in figure 4.1.

Figure 4.1. Example of an auto-edit in Manipula

```
IF (errorcode = '01') AND ABS(Male + Female - Employees) < 10 THEN
   Male:= ROUND(Employees * (Male / (Male + Female)));
   Female:= Employees - Male;
ENDIF
```

5. A standard method for tabulation with Manipula.

The survey results consist of a large amount of tables. These tables are made by Manipula. To avoid having to develop and maintain a large number of Manipula setups, the decision was taken to standardise the setups as much as possible. For each type of table, a standard setup was developed and stored in a separate file. This approach to tabulation has a number of advantages:

- Less development efforts,
- Less maintenance efforts,
- Easier administration of the tables,
- Easier to work with by the people of the Department.

The following elements of the table specifications are stored in a standard way:

- Record descriptions,
- Calculations,
- Coding files,
- Subtotal files,
- Text files.

One of the most interesting elements is the way to make complex sub-totals. Suppose, the input file contains the variable age (with a range from 16 to 64 year, in single year age classes), and the output must contain sub-totals for the following new age classes: 16-64, 16-18, 19-20, 16-20, 21-22, 23-24, 21-24, 25-29, 30-60, 61-62, 63-64, 60-64, 23-64, 21-64. Then, two actions must be undertaken:

1) Create a file with all the codes for the classes in the input file (16 to 64). Let each code be followed by a code representing the (aggregated) class in the output file. Here is an example of a small part of such a file:

```
16 010300
17 010300
18 010300
 .
 .
60 0811121300
61 0911121300
62 0911121300
63 1011121300
64 1011121300
```

2) Make a Manipula setup with two input files, the first input file is the data file, and the second input file concerns the input and output class codes. The manipulate section of the

Manipulate setup uses a duplicate instruction. After duplication, the records must be sorted and summed in a sort section. Here is an example of such a Manipula setup:

```
INPUTFILE "DATA"
           Age               1  STRING[2]
           No_of_Employees   3  INTEGER[7]

        INPUTFILE "DUPL" LIST
           Age               1  STRING[2]
           Age_sub           4  ARRAY[1..5] OF STRING[2]

        OUTPUTFILE
           Age               1  STRING[2]
           No_of_Employees   3  INTEGER[7]
        VAR
           loop : INTEGER;
        MANIPULATE
           FOR loop:=1 TO 5 DO
              IF Age_sub[loop]<>'  '
                 THEN Age:=Age_sub[loop]
                      DUPLICATE;
              ENDIF;
           ENDDOE;
        SORT
           Age;
           No_of_Employees, SUM;
```

6. Conclusion

Five years ago, the Annual Survey of Employment and Wages covered approximately 60,000 establishments with 350,000 employees. It took 27 statistical employees to process the data. After the redesign of the survey in 1994, the number of establishments increased to 75,000 with 1,000,000 employees. Due to the more efficient survey process, the substantially larger amount of data is now taken care of by only 22 people.

The redesign required a different way of working of the people involved. That took some time of getting used to. However, the change turned out to be a success. More data is processed with less people in less time, without reducing the quality of the produced statistics.

CAPI and the Collection of Living Conditions Data; A User's View

Ari Tyrkkö and Hilkka Vihavainen, Statistics Finland

1. Introduction

The Living Conditions Survey 1994 was the first comprehensive survey, where the CAPI-method was widely applied in Finland. The sample and data collection of the survey were combined with the Income Distribution Survey.

The decision to start with the new method and to combine two different types of surveys was taken one year before the beginning of data collection. The computerisation of the field interviewers was considered such a big investment that it needed the backing of a real data collection project, not merely a pilot. At that time, the Income Distribution Survey was the only annual survey with personal interviewing. The project was ambitious, because the CAPI-organisation had to be created from the very start (see [1]). Furthermore, the period for building up a totally new data collection system was very short.

The field work was preceded by a pilot survey with a sample of 1 000 individuals. Half of the persons were interviewed by the CAPI-method and half with the conventional PAPI-method. The aim of this work was to collect comparable data with both methods for methodological research in order to study the effect of the interviewing method on living conditions data. The results of this study will be reported in the near future.

In general there were no technical problems with the use of the CAPI-method (with BLAISE) in the individual-based Living Conditions Survey. The difficulties, which came up with the new method, were mainly related to the organisation of the survey and the documentation of data collection instruments. The Living Conditions Survey is conducted in Finland every seventh or eighth year. The preceding one was from the year 1986. Less than half of the questions were maintained in their old form.

The household-based Income Distribution Survey had a more complicated questionnaire structure containing questions addressed both to the whole household and its individual members. The problems which arose in this connection were different in nature. The Income Distribution Survey is conducted in Finland on an annual basis. More than 90 percent of the income items are gathered from different registers. The main purpose of the interview is to collect information on household composition, on the economic activity of the household members and on some rare income items, which are not registered. The Income Distribution Survey is based on a two year panel.

2. Programming of a BLAISE-questionnaire

The programming of an individual-based questionnaire was quite straight forward. The BLAISE-method did not set any limitations to the structuring of the questions, which were presented earlier in the previous PAPI-interviews. Only a part of the more complicated question groups had to be remodified.

The programming of household-based questions in the Income Distribution Survey was a much more complicated task. The old PAPI-based formulation of some questions was impossible to translate into BLAISE-form. This concerned especially the rosters of household members and their economic activity.

Though the BLAISE is developed partly as a tool for a researcher with no ADP-background, the programming of a complicated or comprehensive BLAISE-questionnaire could be carried out only by an ADP-expert. An ADP-expert was used in the programming of both parts of the survey (Living Conditions and Income Distribution). The reasons for this decision were mainly practical caused by the short time period for the planning of the survey. Anyway, working with the new tool seemed much like the learning-by-doing method - as usual.

3. Data processing

In the earlier PAPI-based production systems interviewers sent all the questionnaires to the central unit, where the data was coded manually and checked centrally with the computer. The clean data was combined with data from registers and other data sources. The production took place in an IBM-mainframe system.

Part of the earlier centralised data checking procedures were transferred to BLAISE-questionnaire to be conducted during the interview. As a consequence the data was obtained 'cleaner' than before, when it arrived from interviewers.

In earlier PAPI-interviews the interviewers made their remarks on the margins of questionnaires. These remarks were checked afterwards separately household by household during the data processing. In CAPI-interviews the respective remarks were gathered under certain code numbers. The checking of these remarks was more effective, because it could be conducted question by question. The remarks also gave easy-to-access information on the quality of questions.

The coding of some classifications was transferred to the CAPI-questionnaire. This concerned the computer-assisted coding of municipalities and occupations. Especially the coding of occupations was supposed to give better results during the interview than afterwards centrally. The system was built up on the dictionary coding-program of BLAISE 2.4. However, it proved out to be too mechanical and involve still a lot of development: The Scandinavian letters were missing and the right items of occupation were difficult to find. This time all the classification numbers had to be checked afterwards manually.

The data transmission (from interviewers to the central unit) system turned out to be problematic for the data processing (see [1]). Whenever interviewers sent new data to the central unit, their whole data file was transmitted including all the earlier concluded interviews. This system enabled the interviewer to make corrections to the old interviews almost freely. It also meant that the data was not ready for processing until the last interviewer contact was made. All in all, this system had the result that most of the advantages of the more effective data collection method for time tables were lost. Earlier when the data was collected by PAPI-questionnaires, it was processed immediately after the central unit had received the questionnaires. Now they had to wait for the last questionnaire. As a result the entire production timetable was delayed by two months from the planned schedule. However, we were able to publish some results of the Living Conditions Survey about half a year earlier than in the previous survey.

One of the subjects, which still needs a lot of development work in the production of the Income Distribution Survey, is how to transfer a large amount of data from BLAISE into the Datacom database, which is in the IBM-mainframe.

The sequential files that were produced with BLAISE-program were rigid to use for further analysis. This concerned especially the Income Distribution Survey. The sequential file that was produced from the questionnaire was 8 000 bytes long, most of which was empty space. This was caused by the household - person roster of the questionnaire. For every potential member a field had to be reserved, if the question was an individual-based one. It meant a space for 16 potential persons per question.

Panel surveys contain the possibility of using the information, which has been collected earlier, in interview situations. This characteristic would have been one of the advantages of the CAPI-method. However, this time it was not utilised.

4. Survey organisation

The introduction of new methods into on-going production systems within a short period of time sets high requirements for the co-operation of people. One of the consequences of the change from PAPI to CAPI/BLAISE was a new division of labour between ADP- and statistical experts. In our case the programming of the questionnaire was left mainly to people specialised in ADP. Sometimes the process appeared to be like a 'black box' to the statistical experts. The functioning of the questionnaire in different situations was difficult to test. If the programming of the questionnaire would have been transferred solely to the statistical experts, it would have reduced the share of ADP-experts' work in the production system and lead to another kind of division of labour.

One of the problems with the survey organisation was the documentation of the questionnaires. A paper copy of a more complicated CAPI-questionnaire with modifications according to different household structures and answers to certain questions is impossible (in practice) to produce. If a survey is conducted regularly but with longer intervals (e.g. every fifth year) the documentation has to be careful. The interviewers as well as some research oriented data users need also some kind of flow chart of the contents of the questionnaire or a list of questions in order to get a total view of the questionnaire. Due to the comparative

survey of interview methods a PAPI-questionnaire of the Living Conditions Survey was produced (but only for this once). The questions from the Income Distribution Survey were also listed on paper.

5. Conclusions

There were three main goals in the implementation of the CAPI-method in the Living Conditions Survey: to collect the data in conjunction with the Income Distribution Survey; to meet the data quality requirements; and to speed up the data collection process.

The first goal was achieved although the design of the survey was very complicated and gave rise to quite a lot of discussion and work. The quality profile of the data has proved to be fairly good. The method did not have a considerable impact on non-response and on major distributions, and the quality of the data has presumably improved. The third goal was met only partly. Although we expected to get the data ready earlier than we actually did, we can feel satisfied with the speed the results became available.

Still, two important lessons remain to be learned. The integration of two large scale surveys was a very demanding task - maybe a bit too demanding. The linking of two different types of questionnaires to one interview session led to a comprehensive and complicated instrument, which was difficult to control. Another main difficulty was the documentation of the questionnaires. Since the research oriented users of the Living Conditions Survey data are interested to know the design of the questions and the structure of the questionnaire, a paper version is necessary. A paper copy from a CAPI-questionnaire is, if not impossible to produce, at least impossible to understand for a non-expert.

Much more emphasis should have been put on the planning of the project. For the sake of the short planning period, the development of new methods was concentrated on the data collection phase of the survey. Yet the full benefits of the new methods could only have been reached, if the whole production system had been trimmed. This concerns especially the speeding up of timetables. The data, which the interviewers send to the central unit, is immediately ready for further analysis and for the publication of preliminary results, at least in principle. The 'on-line' system is possible only when the production system has been prepared for this purpose beforehand.

Reference

[1] Kuusela Vesa (1995): *Interviewer Interface of the CAPI system of Statistics Finland*. In this book.

Update from "Downunder"; History, plans and functions we've built for CAI and Blaise in Australia

Fred Wensing and the CAI team at ABS

1. History and plans for CAI and Blaise III

1993

In late 1993, the business case was accepted for introduction of computer assisted interviewing (CAI) in the ABS household surveys program subject to a comprehensive test being conducted. Previous considerations of CAI had concluded that systems were not advanced enough and cost savings had also been made through the introduction of optical mark reading (OMR) for household survey forms in the late 1980's.

1994

A review of the available software and an evaluation of the potential of Blaise III led to a decision to adopt Blaise as the data collection tool for CAI. Blaise was seen as suitable for ABS household surveys due to its language characteristics (simple syntax and block structure), its metadata-centered approach (shared by all the tools) and the availability of data and metadata conversions to interface with the ABS environment. The main limitation of Blaise III was (and still is) the absence of coding facilities.

A trial of CAI using a Beta-test version of Blaise III was carried out in July 1994 using the Household Expenditure Survey (HES). The HES was deliberately chosen as it was a large survey questionnaire involving more than 800 questions which needed around 12,000 lines of Blaise. The trial was conducted on a sample of 450 households using 10 interviewers who were each equipped with a 486 notebook computer (monochrome screen). The test also included the transmission of data via E-mail. The trial was successful and supported the claims that CAI would improve data quality and timeliness and lead to some saving in costs. The trial also showed that it was relatively easy to introduce CAI to an interviewer workforce who have had very little computing experience. Respondents were not at all concerned by the use of computers in the field.

1995 and the future for CAI

The ABS has now embarked on a full systems redevelopment project which will see CAI (using Blaise III) used in the Monthly Population Survey program (sample size of 30,000 households, involving 560 interviewers) as from October 1997 to coincide with a major post-census revision of the sample. CAI will also be used in our program of special supplementary surveys commencing with the longitudinal Survey of Employment and Unemployment Patterns (sample size of 10,000 persons) starting

with the second wave in September 1996. Developments also include the introduction of telephone interviewing for the Monthly Population Survey in mid 1996. Between the start of 1995 and full implementation in late 1997, the ABS will conduct more than 10 tests of CAI including one major test of 6 months duration.

2. Household Surveys Facilities being developed

The introduction of CAI provides the opportunity to redevelop all the facilities for household surveys in the ABS. A combined team of programmers and survey developers has been established to carry out the developments over the next three years.

The main features of the facilities to be developed include:

- Blaise (version III) will be used as the main data capture tool
- Survey data is expected to be transferred to the client-server (Oracle-Unix) environment so that it can be integrated with other data stores and systems, particularly the ABS data warehouse and generalised survey processing interface
- Links will also be built to enable Blaise metadata to be transferred to and from the data warehouse
- Interviewer workload management facilities are being developed in Turbo Pascal and are likely to be redeveloped using Maniplus
- Office management facilities are likely to be built using Oracle
- Office based coding facilities will make use of Windows-based computer assisted coding facilities
- Office based processing and estimation will make use of SAS
- Tabulation and output processes will make use of other corporate tools

Some specific functions we have developed for CAI are outlined in the following sections.

3. Keyboard remapping using C

The Blaise data entry program allows the remapping of some of the keys on the keyboard to perform common functions required during a data entry session. Unfortunately it does not allow remapping of some of the most commonly used keys such as PgUp, PgDn and Alt-X.

Some sub-notebook computers due to the small size of their keyboards use the Fn key in combination with other keys to produce these key-presses. This often requires computer users to use two hands to activate a function.

To ease the job of our interviewers (who often perform interviews standing at the door) we have produced a C program to remap these keys to function keys, which can be pressed using one hand

The C program, once run, remains in memory and intercepts user key presses. If the key pressed is one to be remapped, the program converts the keystrokes and outputs the converted key-presses. In this way the program can be used to remap any key on the keyboard to any other.

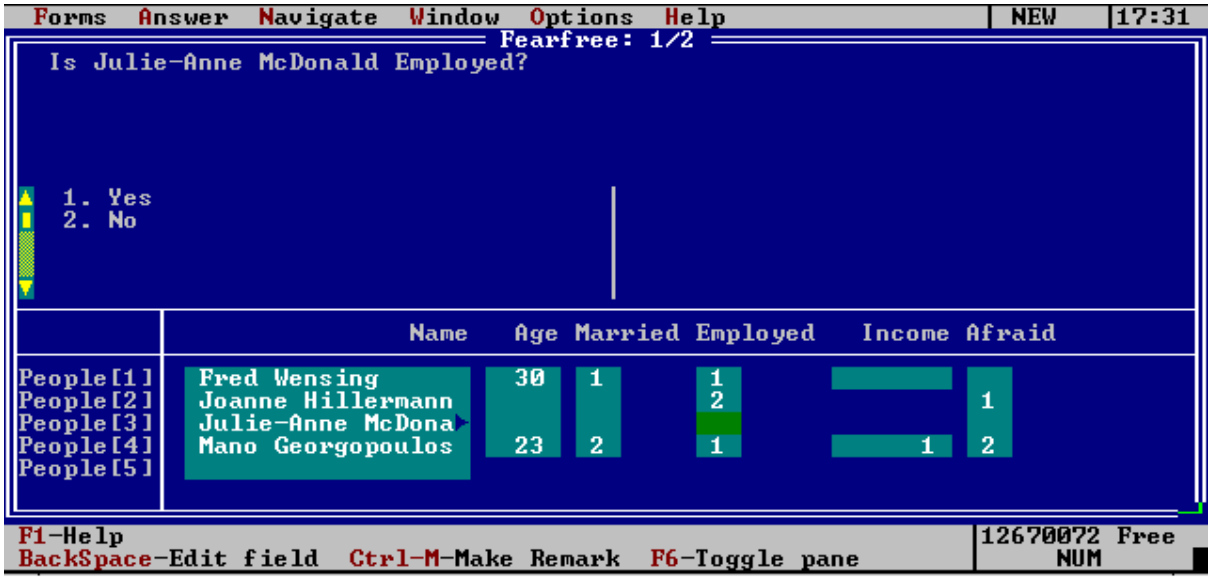
Listing of KEY.C

```
#pragma -mc                          // force compact memory model

#include <dos.h>

void    interrupt int_9();
void    interrupt (*old_int9)(void);
void    main(void);
void    tsr(unsigned size);
void    find_dos_active_flag(void);

// Run only when program is first run
void main(void)
{       disable();                   // Disable interrupts
        old_int9 = getvect(0x09);    // Save old keyboard processing routine
        setvect(0x09,int_9);  // Set new keyboard processing routine
        enable();                    // Enable interrupts
        tsr(1000);                   // Allocate some memory?
}

// Allocate some memory
void tsr(unsigned size)
{       union REGS r;
        r.h.ah = 49;
        r.h.al = 0;
        r.x.dx = size;
        int86(0x21,&r, &r);
}

// Our keyboard routine
void interrupt int_9(void)
{       char far *keybuf = (char far *) 1050;  // head pointer of keyboard
        (*old_int9)();               // use old keyboard routines first
        if(*keybuf != *(keybuf+2))           // if buffer not empty
        {       keybuf += *keybuf-30+5;      // go to character position
                // Key = F7
                if (*keybuf == 65)
                {       // Change key to PGDN
                        *keybuf = 81;
                }
                // Key = F8
                else if(*keybuf == 66)
                {       // Change key to PGUP
                        *keybuf = 73;
                }
                // Key = F3
                else if(*keybuf == 61)
                {       // Change key to ALT-X
                        *keybuf = 45;
                }
                // Key = Ctrl-F3
                else if(*keybuf == 96)
                {       // Change key to F3
                        *keybuf = 61;
                }
        }
}
```

4. Freedom of entry in Blaise tables

In our paper-based household surveys we use a separate household form to collect basic demographic information about all members of a household before collecting information from the individuals. The household form is in a table format, which allows the interviewer to collect information in any order that suits the interview situation. For example, all information about one individual may be collected first, or the same data item may be collected for all members of the household.

We have developed a method for implementing this freedom of data collection in a Blaise table. This is achieved by allowing all fields in the table to be "Empty" and so allowing the interviewers to move around the form as they like. In order to ensure no fields are accidentally left empty we have implemented an edit mechanism to check that all required data has been entered.



Once the final question on the household form has been answered, various edits are triggered to ensure that all fields on the household form that should have been answered have had values entered.

Listing of FEARFREE.BLA

```
DATAMODEL Fear "The Fear Survey"
    BLOCK BPerson "One person"
         FIELDS
              Name "What is your name?"             : STRING[20], EMPTY
              Age "What is the age of ^Name?"       : 0..120    , EMPTY
              Married "Is ^Name married?"           : (Yes, No) , EMPTY
              Employed "Is ^Name Employed?"         : (Yes, No) , EMPTY
              Income "What is the income of ^Name?" : 0..100000 , EMPTY
              Afraid "Is ^Name ever afraid?"        : (Yes, No) , EMPTY
          RULES
               Name
               IF Name = RESPONSE THEN
                     Age
                     IF CheckTable = RESPONSE THEN
                         CHECK
                         Age = Response "Missing value for age"
                     ENDIF
                     Married
                     IF CheckTable = RESPONSE THEN
                         CHECK
                         Married = Response "Missing value for married"
                     ENDIF
                     IF (Age = RESPONSE) AND (Married = RESPONSE) THEN
                         IF Age < 15 THEN
                              CHECK
                              Married = No "Cannot be married under 15"
                         ENDIF
                     ENDIF
                     Employed
                     IF CheckTable = RESPONSE THEN
                          CHECK
                          Employed = Response "Missing value for employed"
                     ENDIF
                     IF Employed = Yes THEN
                          Income
                          IF CheckTable = RESPONSE THEN
                               CHECK
                               Income = Response "Missing value for income"
                          ENDIF
                     ENDIF
                     Afraid
                     IF CheckTable = RESPONSE THEN
                         CHECK
                         Afraid = Response "Missing value for afraid"
                     ENDIF
                ENDIF
    ENDBLOCK
    TABLE BPeople
         LOCALS
              Count : INTEGER
         FIELDS
              People : ARRAY [1..5] OF BPerson
         RULES
              FOR Count := 1 TO 5 DO
                   People[Count]
              ENDDO
    ENDTABLE
    FIELDS
         People : BPeople
         CheckTable "Press '1' to check table" : (Press1 "Press '1'")
    RULES
         CheckTable.KEEP
         People
         CheckTable
ENDMODEL
```

5. Producing a list of edits from a Blaise program

It is often useful to produce a list of edits implemented in a Blaise program. This report can be used by programmers and subject areas to verify that all specified edits have been implemented.

To do this we have written a C program which parses the source code of a Blaise program and outputs a list of the edits found. The output from this program can be loaded into a spreadsheet and used to produce tabular output showing the edits in the Blaise source.

This program relies on the Blaise coding conventions being used in the ABS to produce the list. These programming conventions require that the "CHECK" or "SIGNAL" keywords must be immediately preceding an edit specification in the program. The program searches though the source of a Blaise questionnaire, for the "CHECK" and "SIGNAL" keywords, and interprets the following statements as the edit condition and message specifications.

This program currently only outputs the condition in the edit. A version of this program is currently being developed which also outputs the scope of the edit from the surrounding IF statements. This new version will be written in Manipula rather than C to integrate better with the Blaise suite of software.

Listing of FEAR.BLA

```
DATAMODEL Fear "The Fear Survey"
    FIELDS
         Name "What is your name?"
              : STRING[20]
         Age "What is your age?"
              : 0..120
         Married "Are you married?"
              : (Yes "Yes",
                 No "No")
         Income "What is your monthly income?"
              : 0..100000
         Afraid "Are ^Name ever afraid?"
              : (Yes "Yes",
                 No "No")
         WhatAfraid "WHAT IS ^Name AFRAID OF?"
              : SET OF (Monsters "Monsters",
                        Dark     "The Dark")

    RULES
         Name
         Age
         Income
         SIGNAL
              Income <= 5000
              "Please confirm that income is more than 5000"
         Married
         IF Married = Yes THEN
              CHECK
                   Age >= 15
                   "Must be over 15 years of age to be married"
```

```
        ENDIF
        Afraid
        IF Afraid = Yes THEN WhatAfraid ENDIF
ENDMODEL
```

Output from the Check program:

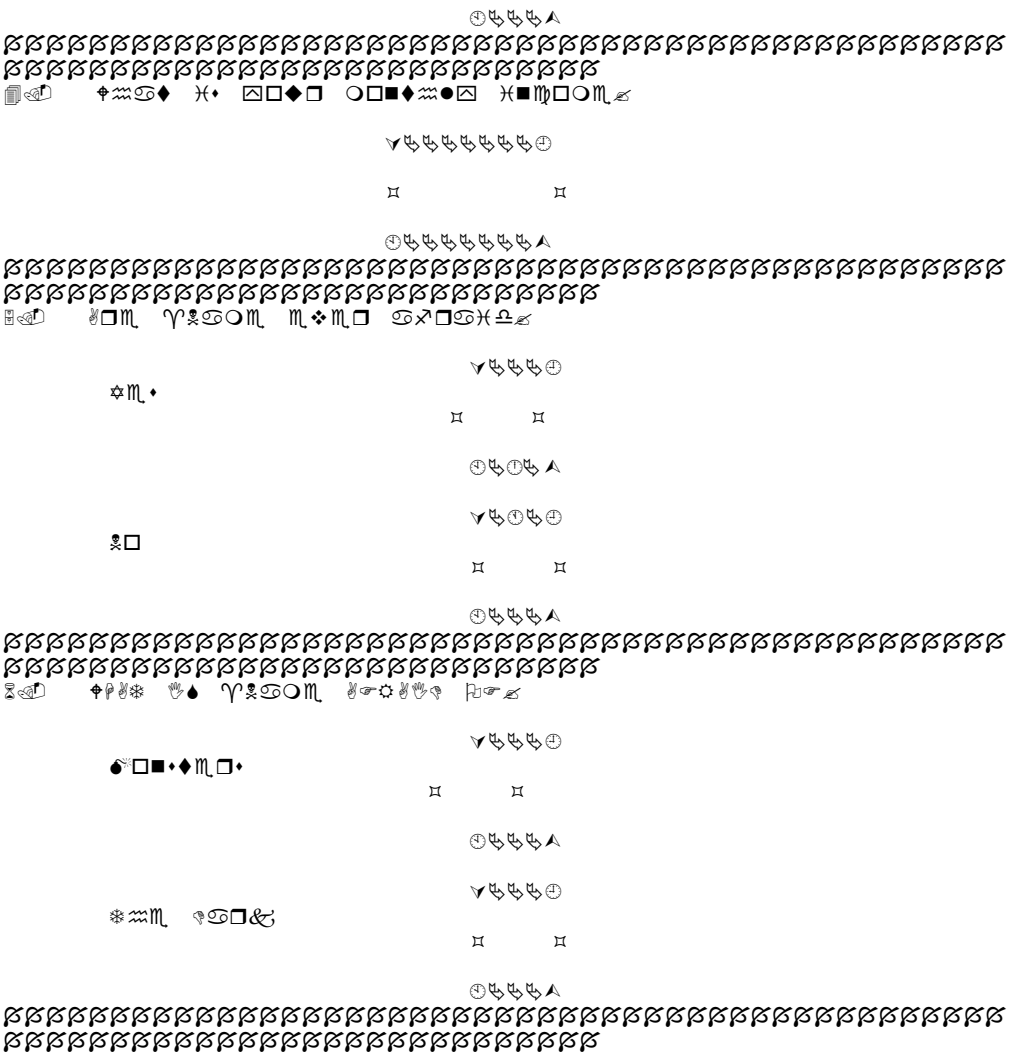
File	Type	Condition	Message
FEAR.BLA	SIGNAL	Income <= 5000	Please confirm that income is more than 5000
FEAR.BLA	CHECK	Age >= 15	Must be over 15 years of age to be married

6. Generating a paper image of a Blaise questionnaire

We consider that a paper version of a questionnaire is a useful component of the CAI environment and have developed a program that generates such a document.

The paper questionnaire generator that we have developed uses Cameleon to access the meta-data of a Blaise questionnaire. Cameleon outputs this meta-data to an intermediate file in a format similar to our current paper questionnaire standards. A Manipula program then processes this intermediate file to make the questionnaire more visually appealing.

Sample output from paper generator using the Fear model from section 5.



This paper questionnaire generator makes no attempt to interpret replacement words in the question text and contains none of the sequencing instructions. Given that future versions of Cameleon may be able to access the "rules" in a Blaise program, these enhancements may be possible later.

7. Interviewer Workload Management System in Turbo Pascal

We have written an Interviewer Workload Management System (IWMS) in Turbo Pascal to enable interviewers to manage their interviews and interact with Blaise.

It allows interviewers to:

- select a survey to enumerate (allows for multiple surveys)
- list the households at which they need to interview
- select which household to view based on completion status;
- sort the list of households by various criteria;
- make interview appointments for households;
- call up the DEP program to perform an interview for a household;

- register calls made to households;
- list summary information about households;
- monitor the progress of their workload;
- call up transmission software to transfer data to the office; and
- access other functions such as training and keyboard familiarisation.

The IWMS was developed in Turbo Pascal so that the look and feel is similar to the Blaise suite. It replaces previous facilities developed using Boreas and may be rewritten in Maniplus when available.

How it works

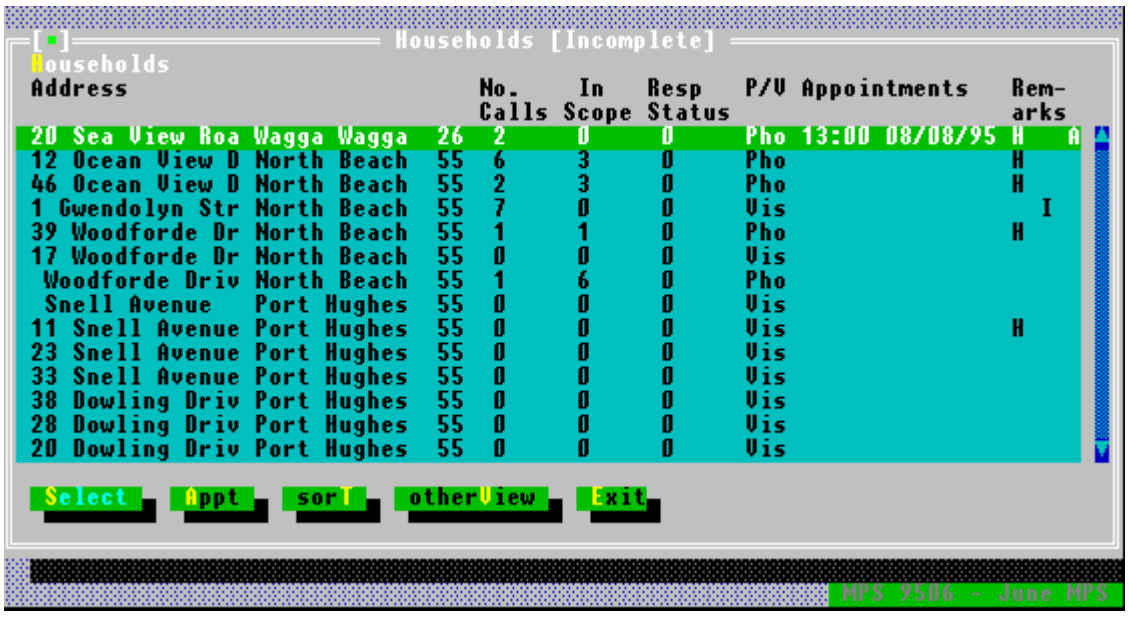
The Blaise files and manipula programs are stored in a separate directory on the notebook for each survey being enumerated. Information used within the IWMS which is not part of the interview process, such as calls and appointments, is stored in text files in the same directory.

Initial information for each interviewer's workload is loaded into a Blaise data file in the office using Manipula. This information includes indicative, address and any data to be included from previous interviews.

In the field, the IWMS extracts this information using Manipula to create the pick list of households (see Screen A).

When a household is selected a more detailed screen of information appears (see Screen B). From this screen the interviewer can call up the Blaise data entry program (DEP) to perform the interview.

On completion of the interview, the IWMS runs Manipula again to extract the household list so that any updated information is reflected within the IWMS.



8. Program to assist with the transfer of Blaise data toRelational databases

We have developed a Cameleon program that converts questionnaire data from a Blaise data structure to a relational database structure.

The structure of the relational database table generated is as follows:

Blaise record key	Blaise field name	Value of field

The generalised table structure has been devised as being suitable for storing of data from any Blaise collected data, without regard to the underlying structure of the Blaise questionnaire. From this basic structure the data can be readily transformed into other table structures using SQL statements prior to further processing.

The system uses Cameleon to extract the meta-data from a Blaise questionnaire, and to produce a Manipula program. The Manipula program when executed extracts data collected using the Blaise questionnaire and writes an SQL program which inserts this data into the relational database table.

Warning: With large Blaise datamodels, it is very easy to produce Manipula programs that are too big to execute.

Listing of BLSE2RDB.CIF

```
[OutFile:= '.\Extract.Man']
USES
    DATAMODEL BSql
```

```
         FIELDS
              OutRec
                   : STRING[[255]]
    ENDMODEL

INPUTFILE IFile : BFile ("input", BLAISE3)

OUTPUTFILE OFile : BSql ("Extract.SQL", ASCII)
SETTINGS
        TRAILINGSPACES = No

MANIPULATE
    DISPLAY (KEY(IFile))
[BlockProc]
    [QuestionLoop]
         [ArrayLoop]
              [SetLoop]
                   [If QuestionType = Block Then]
                        [BlockCall]
                   [Else]
    IF [FullQuestionName] <> EMPTY THEN
         OutRec := 'INSERT INTO [QuestionnaireName] VALUES ('''
                   + KEY(IFile)
                   + ''', ''[FullQuestionName]'', '''
                        [If QuestionType = String Then]
                   + [FullQuestionName][&]
                        [ElseIf QuestionType = Date Then]
                   + DATETOSTR([FullQuestionName])[&]
                        [ElseIf QuestionType = Time Then]
                   + TIMETOSTR([FullQuestionName])[&]
                        [Else]
                   + STR([FullQuestionName])[&]
                        [EndIf]
 + ''');'
         WRITE( OFile )
    ENDIF
                   [EndIf]
              [EndSetLoop]
         [EndArrayLoop]
    [EndQuestionLoop]
[EndBlockProc]
ENDWHILE
```

Note. Copies of the programs and other sample code can be made available on disk or by E-mail (Contact: cai.team@abs.telememo.au or fax +61-6-2525281)

Advances in Automated and Computer Assisted Coding Software at Statistics Canada

M. J. Wenzowski, Statistics Canada

1. Introduction

Statistics Canada has successfully employed generalized automated coding software in numerous applications since the original release of the ACTR[i] (Automated Coding by Text Recognition) system in 1986. As successful as this software has been, additional requirements have continued to arrive, such that it has become necessary for a replacement to be developed. This paper describes the successor to the current production system.

Designed to operate in all of the major computing environments at Statistics Canada, including hand-held, desktop, server, and mainframe environments, this new release represents a complete re-engineering of the software and its underlying algorithms and components. It contains no proprietary (third party) software, and can be delivered "shrink wrapped" for use at any site, with no additional software purchases or setup required. Coding functions can be embedded within any other application, making the presence of the software completely transparent to the user(s) of the hosting applications. This makes the software particularly well suited for use in data capture operations, including CATI and CAPI. The system also includes an easy to use Graphical User Interface (GUI) which minimizes the burden of maintaining coding dictionaries and coding strategies, and provides a comprehensive automated and computer assisted coding environment for stand-alone coding applications. The kernel of the system is a highly optimized coding engine, designed to concurrently search multiple coding databases, while still providing users with the ability to exercise complete control over the system's text manipulation and recognition processes.

2. System Overview

The next release of ACTR, known as "ACTR version 3" (ACTR v3) is based largely upon the existing production version of ACTR, but contains significant changes in capability. As with its predecessor, ACTR depends upon the pre-definition (entirely under user control) of a database of text and code correlations in order to perform both automated and computer assisted coding. ACTR provides a comprehensive toolset to allow for the creation and maintenance of this *coding database* both from high-volume batch-oriented and GUI-based methods. Once built, the coding database is the object of various searching methods used to attempt to map text to a known code in a computer assisted or automated coding application. In effect, this is the "Text Recognition" referred to in ACTR's name. As described below, the text searching process is quite sophisticated, and is capable of locating not only matching text, but "close matches" as well. As a summary of the overall process, consider the diagram which appears in figure a: the actr process model.

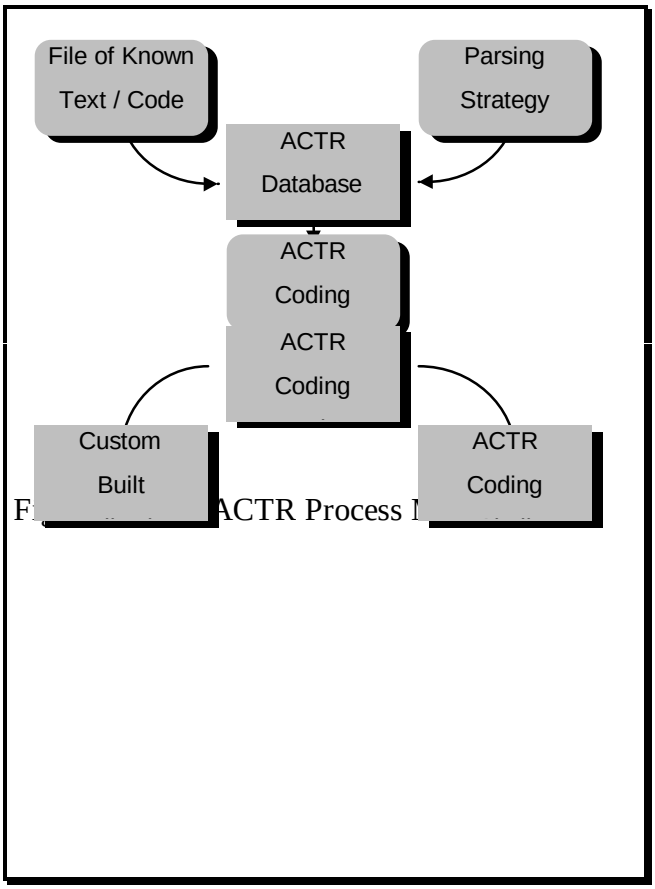


Figure a: ACTR Process Model

The development of an ACTR-based coding application begins with the definition of a set of codes, closely associated with text. These text and code correlations are loaded into an ACTR coding database, and are subsequently used as the subject of a text search during a coding task. When text provided as input to the search is matched with text in the coding database, the previously associated code is returned. Though rather simplistically stated, this is essentially how ACTR functions. Of course, there are numerous complicating factors in searching for matching text - not the least of which is simply defining what we mean by "matching text."

Text recognition requires that sufficiently similar word formats be recognized and consistently mapped to a single word. Examples of variances in word format which a coding application can expect to encounter are: the presence of trivial words; inconsistent use of double letters; and multiple word groupings which may be hyphenated, joined, or entered as separate, multiple words. In ACTR, the function responsible for the standardization of words is known as *parsing*. The flexibility and ease with which users of the software can control the parsing process has played a key role in the widespread success of ACTR within diverse applications, using numerous coding strategies. ACTR has been successfully employed to code a wide variety of variables, including occupations, geographic codes, commodities, marine vessel names and various social and cultural variables. Probably the most successful application to date, in terms of both cost reduction and improved data quality is the use of ACTR in the 1991 Canadian Census of Population[ii]. In recognition that ACTR's user-accessible, highly flexible parsing mechanism has played a key role in its success across diverse applications, ACTR v3 provides extends this mechanism and provides significantly enhanced parsing

capabilities. The parsing process is discussed in greater detail below, in the section entitled: "Text Parsing."

The recognition process continues with the consideration of groups of words, or *phrases*. Examples of problems encountered in this process include missing or extraneous words, and words occurring in an inconsistent order. In the case of a well-tuned, highly optimized coding application, the coding database contains most of the cases expected to be encountered on input. Accordingly, a mechanism must exist for locating these "complete" matches as quickly as possible. This capability is critical in an automated coding function in both high-volume batch processing applications and in high-speed interactive applications, such as CATI and CAPI. The recognition process must also have the ability to locate "partial" matches, or phrases which contain most of the significant words contained in the input text. As expected, performance is critical in batch operations, but since the partial match technique is particularly useful in computer assisted coding functions, performance is again an issue, since interactive response must be as rapid as possible . The text searching process is discussed in greater detail below, in the section entitled: " Search Methods."

3. Text Parsing

As sophisticated as ACTR's searching and matching methods might be, the quality of a text match is almost entirely dependent upon the outcome of the parsing process. Accordingly, the ACTR parser is entirely user-controlled and offers full control of both the parsing data used to effect the parse, and the order in which the parsing steps are applied. A graphic summary of the parsing process appears in figure b: the actr parsing mechanism. It may be useful to reference this diagram while reading the following description of the parsing process.

Ideally, the outcome from parsing is such that any two descriptions with the same words will be identical in their ACTR representation regardless of their syntactical and grammatical differences. For example, the two descriptions: "`computer programmer`," and "`programming of computers`" could, when parsed, result in "`comput program`". How the input descriptions are parsed is controlled by the parsing strategy defined by the user of the software. The strategy may involve the reduction of plural forms, elimination of trivial words, removal of suffixes or any number of other operations.

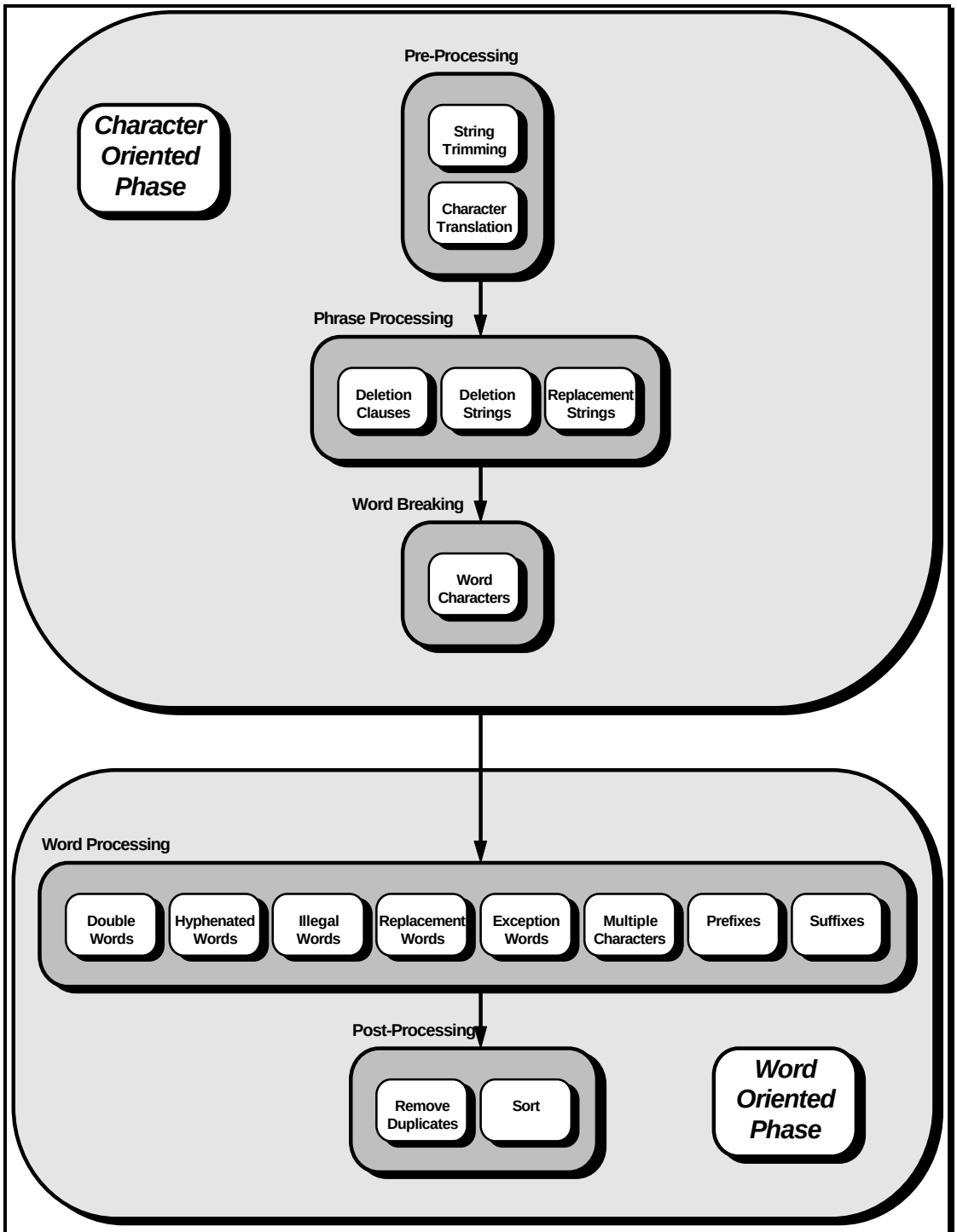


Figure B: The ACTR Parsing Mechanism

The parsing strategy can be easily tailored to suit a particular coding project's requirements. There are three principal mechanisms used to change the parsing strategy:

1. by changing the types of parsing processes that are used,
2. by changing the order in which parsing processes are used, and
3. by changing the parsing data used for parsing processes.

Some research and experimentation is always necessary, in order to determine the best parsing strategy for a particular application. The process of defining a parsing strategy can be complex, especially since changes made within one step or process may affect subsequent steps and processes.

The parsing strategy is comprised of two main *phases*, each with multiple *parsing steps*, acting on their associated *data*. The difference between the phases is the manner in which text is processed. In the "Character Oriented Phase," all text manipulations are performed on a character by character basis, regardless of context or proximity to other characters. Words which may appear in the text are not recognized as such until the second phase of processing, the "Word Oriented Phase."

Within the two parsing phases are five parsing steps. Within each of these steps, users have control over which processes are to be performed, and how this processing is to be performed. This is accomplished through the use of parsing data - changing the parsing data alters the associated process's performance. In addition, within each of the "Phrase Processing" and "Word Processing" steps users may exercise further control by deciding the order in which the processes within these steps will execute.

As with any complex procedure, the parsing process is best understood by individually considering its components. The following sections provide a more in-depth presentation of each of ACTR's parsing components in approximately the same order in which they appear in the diagram.

The "Character Oriented Phase" is composed of the three parsing steps: "Pre-Processing", "Phrase Processing" and "Word Breaking." In these steps, the input text is processed as a continuous stream of characters, and as such, no consideration is given to grammar, punctuation, spelling, word order, or any other contextual information. The central purpose of this phase is to allow for the recognition of particular character sequences, exactly as they appear *in situ*. This can prevent information losses that might occur when the text is subsequently grouped into its constituent words.

The "Pre-Processing" step is composed of two processes: "String Trimming" and "Character Translation," which are always executed in the same order. String trimming is a process which standardizes the input text by eliminating extraneous characters. This may include multiple blanks, tab characters, newline characters, etc. Character translation uses a list of valid word characters and their associated translations to transform the entire input description. This allows control over case sensitivity, and also allows for control of the treatment of accented characters. For example, in converting lower to upper case some applications may wish to transform the lower case letter "é" to an upper case equivalent of "É" (Canadian French), while others may wish to convert it to "E" (International French). Still others may convert both "é" and "É" to "E" and effectively ignore all accents.

The Phrase Processing step is composed of three processes: "Deletion Clauses," "Deletion Strings" and "Replacement Strings." These processes are all optional and if specified, are performed in any order the user chooses. Deletion clauses are a method of delimiting a phrase which should be removed from the input text. By defining a beginning and ending string, any characters enclosed between the strings are removed from the input text. Deletion strings allow for character sequences found at any position in a description to be removed. Replacement Strings are most useful for standardizing abbreviations. This is useful since abbreviations commonly include characters which would normally be processed as word separators.

The Word Breaking step defines a word as any contiguous sequence of characters in an input string which are all members of the set of characters contained in the user controlled word character list. Characters not included in this list are used as word delimiters.

Within the Word Oriented Phase processing is performed on a word by word basis. This phase is composed of two steps: "Word Processing" and "Post-Processing." The order of these steps cannot be changed, but the order of the processes within the steps and the data used by each of the processes can be modified.

The Word Processing step is the first point at which the parser treats the input text as a collection of words. It is composed of eight processes, which can be executed in any order the user chooses. Together, these steps allow for:

- consistent treatment of multiple word occurrences
- consistent treatment of hyphenated words
- removal of words known to be devoid of information content
- replacement of words to ensure consistency (also used to ignore trivial words)
- recognition of words by examining the word root
- reduction of multiple character sequences to a single character instance
- removal of prefixes
- removal of suffixes

The Post Processing step completes the task of parsing. This step is composed of two processes, but their sequence of execution does not affect the form of the parsed output. This step provides for control over multiple word occurrences as well as the final order of the words. For some applications, multiple words are significant while in others they obscure the sense of the text. Similarly, word order may or may not be significant, depending on the application.

The preceding discussion of the ACTR parsing mechanism is merely an outline of its functionality. For complete details the interested reader is directed to the complete set of ACTR documentation[iii].

4. Search Methods

ACTR employs two distinctly different methods for locating matching text. The first of these locates only complete matches, using a technique known as the *Complete Phrase Key* (CPK), while the second, known as the *Indirect Match*, retrieves partial matches, in which both the input text and the database text have at least one word in common.

Ideally, in order to reduce uncertainty and increase coding quality, applications which use ACTR should strive for achieving a state in which their application's coding database contains as many instances of expected text as possible. This being the case, the most efficient manner in which text can be retrieved is to search the database for the input text, precisely as it appears after having been parsed. This requirement is addressed by the CPK. ACTR forms the CPK by accepting the word set returned by the parser, and using a data compression technique, generates a binary string which can be used to provide keyed access into the coding database. To prepare the coding database for CPK searches, all text in the database has a CPK associated with it at load time. In fact, the CPK is used at load time to locate duplicate text in the database, and so control redundancy.

Data compression within the CPK is effected through the location of commonly occurring two and three character groupings, known respectively as digrams and trigrams. These character sequences are replaced by a single byte (eight bits), used by ACTR to represent a

particular digram or trigram instance. Within ACTR, text characters are represented by a single eight bit byte. This representation offers 2^8, or 256 different bit string combinations. However, a typical ACTR application employs a character set which uses only 26 upper case alphabetic characters, 10 numeric characters and the hyphen character. ACTR in turn reserves two additional bytes to represent string termination and word separation. Combined, the total number of used characters is only 39. This means that in the typical case there are 217 bit string combinations which are not used. ACTR associates these unused bit string representations with previously defined, commonly occurring, digrams and trigrams. The result is that the length of the CPK generated by ACTR is typically significantly less than 50% of the original text's length.

The availability of the CPK ensures that ACTR can locate matching text within the coding database in a very efficient manner. It also provides ACTR with the means by which data access can be significantly accelerated, to improve system performance and response times. At the same time, use of the CPK ensures that no information loss occurs as a result of this access optimization.

Typically, Indirect Matching is employed only for those cases in which ACTR is unable to locate a database match using the CPK technique. Basically stated, the technique employed is to find all text with words in common and, through the application of a scoring technique, determine the "closeness" of the matches found. The method is based on the availability of previously calculated weights associated with every word known to the database.

The method used to calculate the weight for a given word is shown in **figure c: word weight calculation** Within that figure, *wordcodes* is the total number of unique codes associated with text in which this particular word occurs, and *totalcodes* is the total number of unique codes defined within the coding database. No pretense is made regarding the validity of the calculation with reference to information theory. Our experience to date indicates that word weights have their greatest usefulness in trimming the search process. Word weights also participate in the text score calculation, used to compute an index of the "closeness" of matches found.

$$WordWeight = 1 - \frac{\log(wordcodes)}{\log(totalcodes)}$$

Figure C: Word Weight Calculation

The technique used to compute a score is shown in **figure d: text score calculation** In this calculation:

- *count(COMMONwords)* is a count of the words in common, between the input text and the database text currently being evaluated
- *count(DBwords)* is a count of the number of words in the database text being compared
- *weight(COMMONwords)* is the word weight of each of the words in common between the input and database text
- *weight(INPUTwords)* is the word weight of each of the known words in the input text

As with word weights, no pretense as to the theoretical correctness of the scoring method is made. It is used as a general heuristic for ranking incomplete matches, and is also used to trim the indirect matching search process. Note that in an effort to make the values more intuitively useful to users, word weights are always in the range of 0..1, and text scores are always in the range of 0..10.

$$a = \frac{2 \times (count(COMMONwords))}{count(DBwords) + count(INPUTwords)}$$

$$b = \frac{\sum weight(COMMONwords)}{\sum weight(INPUTwords)}$$

$$TextScore = 10 \times \left(\frac{a + 2b}{3} \right)$$

Figure D: Text Score Calculation

The search for an incomplete match begins with a database search for all of the words in the input text. For each word found, its associated word weight is retrieved. In descending order by weight, each word is used to find all text in which it occurs. When text is retrieved, a score for the match is computed and compared against threshold values provided by the user, to determine whether or not an acceptable match has been found. In the process, the scoring technique is also used to predict the most optimistic score possible for each word based search iteration. The optimistic score value is compared against the same threshold values provided by the user, to determine whether or not the current iteration of the search should proceed, and in this way serves to reduce search times significantly.

6. Context Switching

ACTR has the ability to assign codes based on multiple database searches. This provides coding applications with a means by which multiple fields may be coded with a single pass. This feature is equally as useful to high volume batch processing applications - in which data file manipulation must be minimized - as it is to interactive processing applications (particularly CATI / CAPI) in which the immediate availability of the respondent requires a single, logical pass through the interview script.

In ACTR terminology, each database opened for matching is referred to as a *context*. An open context associates the database to be used for matching, its parsing strategy and other related matching information. At run time, the application simply passes to ACTR the identity of the context to be used, and ACTR automatically switches its processing, to be associated with the desired context.

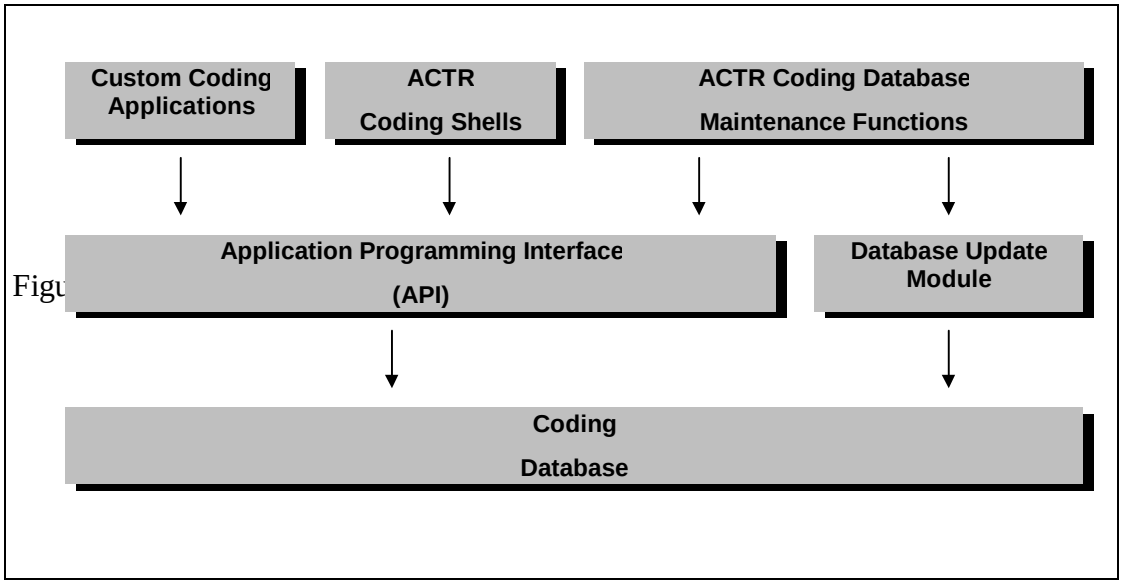
When embedded within a custom application this feature also provides the means by which multi-field dependency coding can be performed. An example of this requirement is the coding of industry and occupation, particularly from data sources submitted by the general population. Coding these variables from the information contained within a single field typically results in low success rates and low quality coding[ii]. By providing the hosting application with multi-field coding, through the use of ACTR contexts, the application developers are provided with the ability to more closely model the manual coding process currently in use.

7. Operating Modes

ACTR is designed to provide coding services in all of the operating modes typically encountered in a large scale statistical agency. The system includes batch processing facilities to provide automated coding services for high volume, unattended coding. These facilities are typically employed through the use of some form of scripting language and are controlled through parameter and control file settings. ACTR also provides a GUI-based interactive facility through which both automated and computer assisted coding can be performed.

Both batch and GUI-based components are referred to a "shells," since they are services which provide the user with a high level interface to lower level coding functions. Users may also call these lower level functions directly, through a custom-developed application. ACTR makes this possible through the provision of an *Application Programming Interface* (API) to the matching functions. The relationships among these components is shown in figure e: actr component model. Note that the Database Maintenance Functions also use the coding API for all database search functions. By ensuring that the API can support all of ACTR's high level processes, custom applications which also use the API can be assured of full functionality.

Coding database maintenance and parsing strategy manipulation functions are provided in both batch and interactive modes, but not through a publicly accessible API. This is an



Figu

intentional design feature. Coding databases tend to be very stable throughout the lifetime of the coding application, with updates typically performed very infrequently and under highly controlled conditions. In coding applications, high-performance coding functions are essential, and in order to achieve this performance, trade-offs are required in the domain of database update mechanisms. Limiting the update mechanism removes the requirement to provide the overhead functions (such as record locking, logging, before and after imaging, etc.) typically associated with multi-user update capability.

8. Summary

The latest release of generalized computer assisted and automated coding software at Statistics Canada has been described in this paper. As this release is still in beta test mode, no performance statistics are yet available. However, a number of significant applications have

already committed to using this new release of ACTR in their production cycle within the next year.

While it is hoped that this paper has provided the reader with a thorough overview of the software, the description is by no means complete. Those who wish to pursue their investigation of this software further are directed to the ACTR documentation set[iii].

References

MANIPLUS, A powerful environment for managing Blaise III applications

Hans Wings and Lon Hofman, Statistics Netherlands

1. Introduction

With the introduction of survey processing software like the Blaise system more and more statistical offices, researchers and commercial marketing agencies are changing from PAPI to CADI, CAPI or CATI. After several years of experience the benefits of computer assisted interviewing are clear: better quality of data, lower costs, and more efficiency. The first generation of statistical software at Statistics Netherlands (Blaise CADI, CAPI and CATI programs, Abacus, Bascula, etc) was developed as a tool to assist in a specific stage of the survey process. With the introduction of Blaise III a next step to integrated survey processing has been made. All tools are integrated into one environment, they speak the same (meta) language and the interfaces are standardised.

Recently, a new member has been added to the Blaise III family of tools: ManiPlus. ManiPlus is a powerful tool to build applications involving Blaise. It gives survey system designers the possibility to combine tools like the Data Entry Program, the Data Viewer, Manipula, and a DOS Shell into a user-friendly package controlled by user-defined menu systems and dialog boxes.

In this paper we will present a brief overview of the components available in ManiPlus. Because the development of case management systems for CAPI is very much in the public eye, we will illustrate the features of ManiPlus using LIPS [1] (Laptop Information-system for Personal Surveys) of Statistics Netherlands as an example.

2. Case management for CAPI

A few years ago, the first steps in designing laptop case management systems were taken. Some organisations (like Statistics Netherlands with LIPS) have designed their own laptop case management systems with their own objectives or emphases. One thing these systems have in common is that they are all shells around the CAPI questionnaire programs. This is because the CAPI software is usually not capable of dealing with all the functional requirements of survey management. Although these sytems do their jobs well, they have a number of drawbacks. The survey management systems are developed with systems like database systems (Clipper, FoxPro, etc),

programming languages (Pascal, C, etc) or just DOS batch files, which differ a lot from the CAPI development systems. This leads, among other things, to different user-interfaces or data representations within one survey management system instead of one face for the whole system and one data format within the whole sytem.

Other organisations have adopted existing systems. Usually these adopted systems don't satisfy the requirements of that specific organisation completely. Adjusting to these requirements is often difficult or impossible.

A disadvantage of the LIPS system at Statistics Netherlands is the dependency on the current version of Blaise. With the introduction of Blaise III the system must be rewritten or adjusted to the new version. It is obvious that a tool, like ManiPlus, which will solve the problems mentioned and avoid the disadvantages of the currently developed survey systems would be a great help.

Although every laptop case management system will have it's own requirements, many of them have a lot of functions in common. According to Nichols and Kindel [2] laptop case management systems have to be able to perform the next core functions:

1. Accept and store interviewing assignments.
2. Display a list of these assignments to the interviewer.
3. Select a case to interview.
4. Store the interviewer data and retrieve it when necessary for partial interviews.
5. Record the status or disposition of each case, such as complete, partial, appointment, or untried.
6. Initiate telecommunications or prepare diskettes for mailing.
7. Perform various system functions, such as logging in, setting and using passwords, and maintenance of core files.
8. Provide unidirectional or bi-directional e-mail communication between interviewers and supervisors.

Of course, every organisation wants to add it's own functionality to this list. So the tool needed must be able to implement the above-mentioned core functions, but it should also provide enough flexibility to add almost any required functionality.

3. Introducing ManiPlus

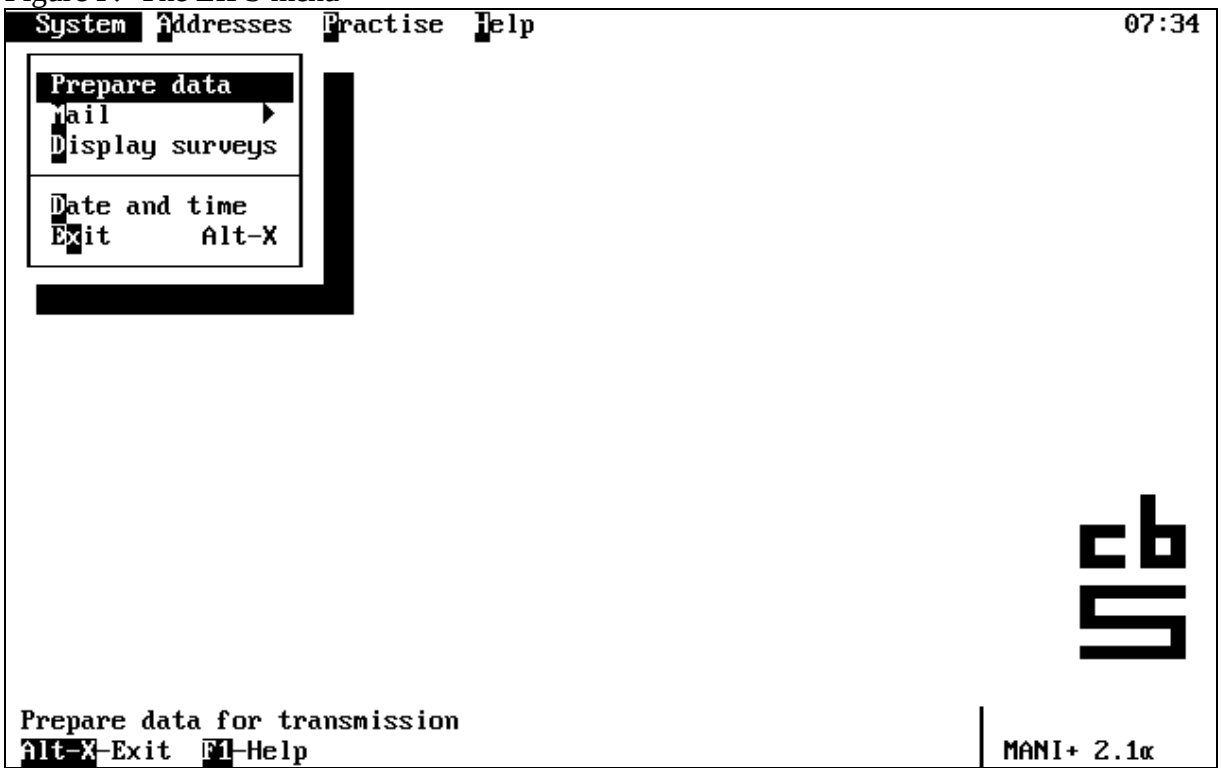
A new tool, called ManiPlus, has been added to the family of Blaise III tools. This tool is well suited for development of (survey management) systems involving Blaise. As the name suggests the basis of this tool is Manipula, so all the functionality which is already present in Manipula is also available in ManiPlus. Further tools such as the Data Entry Program, the Data Viewer, Manipula itself, a DOS Shell and a fully equipped help facility are integrated in ManiPlus. All components of ManiPlus can be controlled by user-defined menu systems and dialog boxes. Finally, file handling is not implicit like in Manipula but fully controlled by the user.

In the next sections we want to give you a feeling of what is possible with ManiPlus. This description is by no means exhaustive. We will only take a short look at some components. In each section we will show how a part of the LIPS system could be built using ManiPlus. On passing we will refer to the core functions indicated by Nichols and Kindel in order to indicate that all these functions can be implemented using ManiPlus.

3.1. Menus

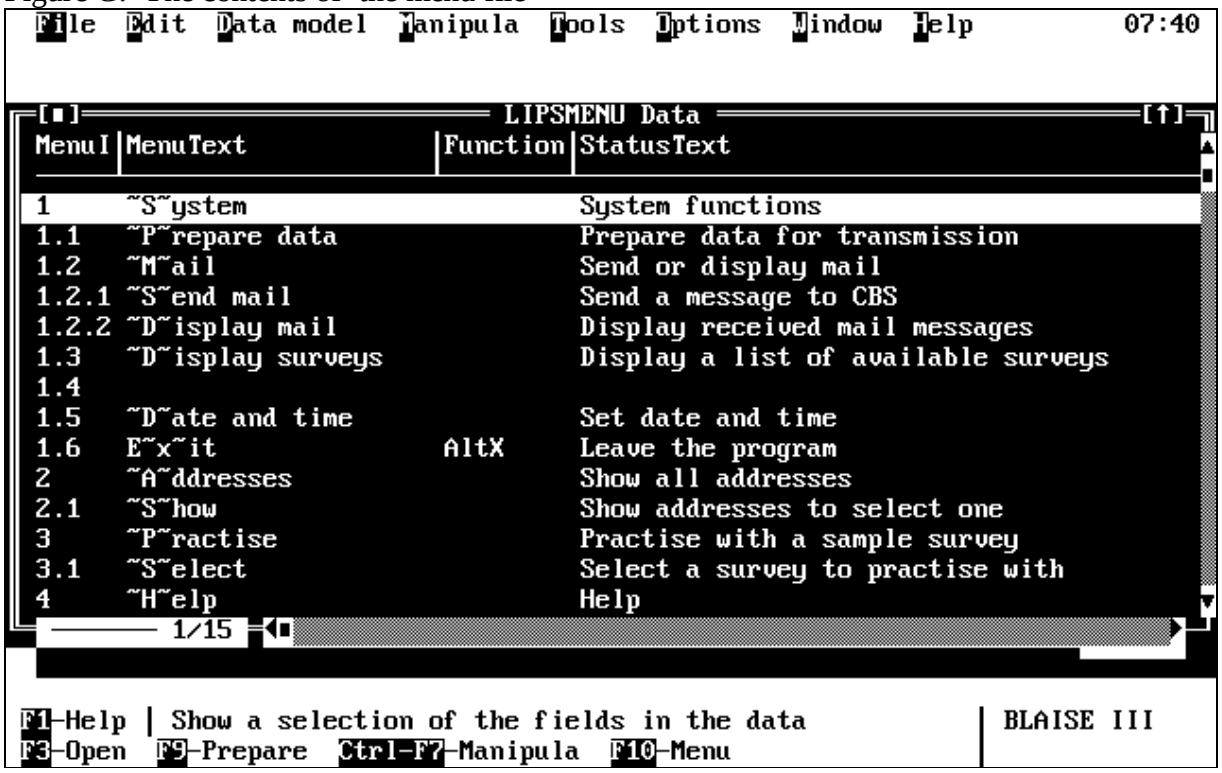
Operators must be able to work with laptop interviewing system and therefore they need a menu, which guides them through the system. In the LIPS system the menu could look as follows:

Figure F. The LIPS menu



In ManiPlus the menu items are defined as records in a file of a specific datamodel. The definition of that data model has to satisfy a number of conditions, which we will not explain in detail here. The following figure shows the Data Viewer of the Blaise control center with the contents of the data file which defines the menu of the LIPS system. Each line in the Data Viewer corresponds to one menu item entry in figure 1.

Figure G. The contents of the menu file



The ManiPlus setup which runs the LIPS system looks as follows:

```
InitProc
REPEAT
  Res:= LipsMenu.Menu('~Alt-X~-Exit ~F1~-Help')
  IF (FunctionKey <> altx) AND (Res = 0) THEN
    CASE ModuleName of
      'showaddresses'   : ShowAddresses
      'makemessage'     : MakeMessage
      'preparedata'     : PrepareData
      'surveys'         : ShowSurveys
      'datetime'        : SetDateAndTime
      'selectsurvey'    : SelectSurveyToPractise
    ENDCASE
  ENDIF
UNTIL FunctionKey= AltX
```

The instruction *LipsMenu.MENU* activates the menu in the program. After selecting a menu item the data that corresponds with the menu entry is available in the program. When the menu item *Display surveys* in figure 1 has been selected the sixth record (see figure 2) of the file *LipsMenu* becomes the current record. This record contains a field *ModuleName* to identify the procedure which is associated with the menu entry. In this case this field has the value 'surveys'. The CASE statement invokes the procedure which is associated to the selected menu item.

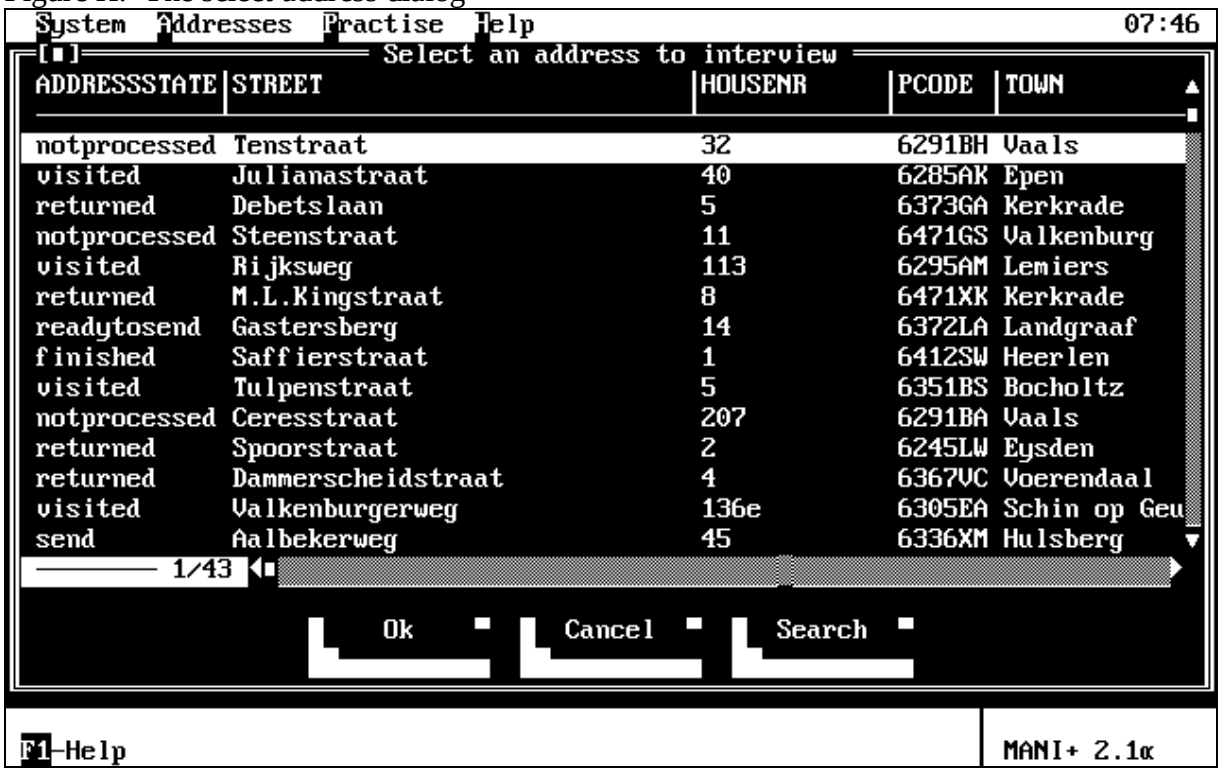
By using a REPEAT loop as the core of the ManiPlus setup, the operation of the setup is completely menu driven. A procedure (also a new feature of Manipula 2.1), which will execute the statements associated with the menu item, is assigned to every menu item.

The procedure *InitProc* could check in a specified directory for new addresses that have to be visited and store these addresses in the appropiate files of the system (the first core function of a laptop interviewing system).

3.2. Dialogs

To implement the second and third core functions of a laptop managament system you need a dialog which helps the interviewer to select a case to interview. In LIPS a dialog with a list of available addresses is shown to the interviewer. This list contains only the necessary information of the addresses for the interviewer.

Figure H. The select address dialog



The definition of this dialog in the ManiPlus setup looks as follows:

```
DIALOG SelectAddressDialog "Select an address to interview"
  LOOKUP AddrTempo
    POSITION = (1,1)
    SIZE = (77,16)
    FIELDS
      AddressState
      Street
      HouseNr
      PCode
     Town
  BUTTON ButtonResult
    CAPTION = ' ~O~k '
    VALUE = ok
  BUTTON ButtonResult
    CAPTION = ' ~C~ancel '
    VALUE = cancel
  BUTTON
    CAPTION = ' ~S~earch '
    ONPRESS = SEARCH (AddrTempo)
```

The dialog *SelectAddressDialog* is composed of different dialog elements, in this case three buttons and a lookup. The lookup is assigned to the file *AddrTempo* and only five of the fields (the information necessary for the interviewer) of each record are displayed in the lookup. When the third button is pressed the search dialog of the DataViewer is activated. With this dialog you can change the order in which the records must be displayed according to the keys which are defined in the datamodel of the file concerned. When the first button is pressed the dialog is closed and the currently selected form becomes available for further processing.

Each dialog can be composed from different dialog elements. In total, ManiPlus provides you four different dialog elements:

1. A text at a certain position,
2. A button which genererates a command when pressed,
3. A control which allows the user to enter data into a field, and
4. A lookup which displays a Data Viewer.

These four dialog elements give you the flexibility to fully design your own dialogs.

3.3. Data Entry

If an interviewer has selected an address to interview, you need to start a data entry program (DEP). The DEP of Blaise III has been integrated into ManiPlus, so it is not necessary to start it outside the management program in a DOS shell. There are two ways to start the DEP in ManiPlus:

1. If you know the data model and the name of the survey beforehand (that is to say while designing the ManiPlus setup) the DEP can be started as a method of a file with *F.EDIT(S)*, where F is a file identifier of a Blaise III update file. With the text *S* you can pass the same command-line options like for the stand-alone DEP, except for the meta-file name and the option /F (because they are already defined in the UPDATEFILE section of *F*). Two extra options are useful:

/X	exit the DEP after treatment of one form
/G	start the DEP in the get form mode

The form that has been manipulated in the DEP will be available if the DEP is exited. The DEP can be executed on any Blaise III update file defined in the ManiPlus setup.

2. However, in LIPS, which supports multiple survey interviewing, the data models and the data files are unknown at design time. What survey will be used depends on the case an interviewer selects to interview, so the data model and data file to be used are determined at run time. In this case you start the DEP with *EDIT(S)*, where *S* must contain the name and location of the meta file and optionally the name and location of the data file.

The part in the ManiPlus setup which takes care of selecting an address and starting the DEP for this address looks as follows:

```
CreateAddrTempo(ActiveSurvey)
SelectAddressDialog
IF ButtonResult=ok THEN
  DetermineActiveSurvey
  EditResult:= EDIT(DataDirectory+ActiveSurvey+
                ' /F'+DataDirectory+ActiveSurvey+
                ' /K'+AddrTempo.Spil_ID+
                ' /X')
  UpdateAccountOfAddress
ENDIF
```

The procedure *CreateAddrTempo* creates a temporary file (a new file type in Manipula 2.1) to store all available addresses in memory. With the already discussed dialog *SelectAddressDialog* the interviewer selects a case to interview whereupon the DEP is started for the selected address only. If the interview is (partially) completed the DEP takes care of storing the data of the interview. The procedure *UpdateAccountOfAddress* takes care of accounting for the interview. This is comparable with the fifth core function of a laptop interviewing system.

3.4. Executing Manipula setups

A lot of survey systems use DOS batch files to execute successive Manipula setups. The execution of a Manipula setup (that is to say executing ManiDoit with the corresponding prepared MSF file) is also integrated in ManiPlus. To implement, for example, the preparation of data which has to be sent to the central office (the sixth core function of a laptop survey system) you need a Manipula setup which extracts the data from the Blaise data file. Because LIPS supports multiple survey interviewing, the data file from which the data has to be extracted is unknown beforehand. A possible solution to this problem is to provide the system with a Manipula setup for each survey, which will take care of extracting the completed interviews from the corresponding data file. In LIPS the interviewer invokes this setup by selecting the submenu item *Prepare* in the menu *System*. This menu item executes the procedure *PrepareData* in the setup:

```
PROCEDURE PrepareData
  SelectSurveyDialog
  CASE ButtonResult OF
    ok:
      IF CONFIRM('All completed interviews of the survey '+IntSurv.SurveyName+
                    ' will be prepared for export and deleted in the data file. Are you sure ?')
THEN
        Res:= CALL(SetupDirectory+IntSurvSurvey > NUL')
      ENDIF
    allsurveys:
      IF CONFIRM('The completed interviews of all surveys will be prepared for export '+
                    'and deleted in the data files. Are you sure ?') THEN
        IntSurv.Reset
        FOR I:= 1 TO  IntSurv.RECORDCOUNT DO
          IntSurv.READNEXT
          Res:= CALL(SetupDirectory+IntSurv.BlaiseName+' > NUL')
        ENDDO
      ENDIF
  ENDCASE
ENDPROCEDURE
```

With the CONFIRM function a message box with a YES- and NO-button comes up. The result of the CONFIRM function is *true* if the YES-button is pressed. If the user confirms the selected menu item the Manipula setup is executed with the instruction CALL. The parameter of the instruction contains the name of the setup.

3.5. Running DOS commands

To show how a system function (the seventh function of the eight core functions) may be implemented in ManiPlus we use the date and time setting function in LIPS as an example. On the laptop the interviewer can change the date and time of the system. Entering the new date and time is done by a dialog:

Figure I. The set date and time dialog

In the ManiPlus setup this dialog is defined with:

```
DIALOG DateAndTimeDialog "Set date and time"
  SIZE = (31,12)
  TEXT ('Enter the new date and time',2,2)
  CONTROL Date
    LABEL = ('Date',2,4)
    POSITION = (10,4)
    STATUS = 'Please enter the date'
  CONTROL Time
    LABEL = ('Time',2,6)
    POSITION = (10,6)
    STATUS = 'Please enter the time'
  BUTTON ButtonResult
    CAPTION = '   ~O~k   '
    POSITION = (2,9)
    VALUE = ok
  BUTTON ButtonResult
    CAPTION = ' ~C~ancel '
    POSITION = (16,9)
    VALUE = cancel
```

In this example two controls are used to enter data. The controls provide input lines to enter the new values. An auxiliary field is assigned to each control to recieve the values of the input lines. The text dialog element is used to display an additional text in the dialog. The *DateAndTimeDialog* is invoked by the following procedure just by mentioning the name of the dialog:

```
PROCEDURE SetDateAndTime
  DateAndTimeDialog
  IF ButtonResult=ok THEN
    RunResult:= RUN('DATE '+DATETOSTR(Date)+' > NUL')
    RunResult:= RUN('TIME '+TIMETOSTR(Time)+' > NUL')
  ENDIF
ENDPROCEDURE
```

To execute the system commands the RUN function is used. With the RUN function you get full access to DOS. This function releases as much memory as possible before executing the command. The result is zero if the command was executed succesfully, otherwise it is non-zero. The parameter of the RUN function is the DOS-text of the command to be executed.

When the *OK-button* in the dialog *SetDateAndTimeDialog* has been pressed, the value of the auxiliary field *ButtonResult* is *ok* and the two DOS commands are executed ″in the background″.

The last of the eight core functions of a laptop case management system can also be implemented using the RUN function. Sending an e-mail to the central office, for instance, is easily implemented with a RUN function that invokes a program which provides the appropriate functionality.

3.6. The Help facility

Nowadays every software system provides a help facility to guide the user through the menu and the interface of the system. Providing help information for an interactive system like a laptop survey management system is, of course, very important. With ManiPlus you get a hypertext help facility which links help texts to menu items or dialog elements. It is also possible to invoke a help text within the MANIPULATE section. The help text is stored in a help file.

4. Final remarks

With the introduction of ManiPlus the next step to integrated survey processing has been made. ManiPlus provides all the necessary components to develop survey management systems. Obviously, ManiPlus can be a great help in developing survey management systems. It is now up to you to explore the possibilities of this new tool. ManiPlus has a lot more to offer than we have dealt with in this paper. Not only survey management systems can be built with ManiPlus. The multi-survey shells at NASS [3] (National Agricultural Statistics Service), for instance, are a good example of a system which can be rebuilt using ManiPlus.

While rebuilding the LIPS system with ManiPlus we noted a tremendous decrease in development time. What took one computer programmer almost half a year was

rebuilt with ManiPlus in several weeks. Mind you! You still need some programming experience to build survey systems with ManiPlus.

References

[1] L.P.M.B. Hofman, W.J. Keller*: Management of Computer Assisted Interviews in the Netherlands*. In: **Journal of Official Statistics**, Vol. 9 No.4, Statistics Sweden, 1993, pp. 765-782.

[2] W.L. Nicholls II, K.K Kindel: *Case managment and communications for Computer Assisted Personal Interviewing*. In: **Journal of Official Statistics**, Vol.9 No.3, Statistics Sweden, 1993, pp. 623-639.

[3] R. Schou, M. Pierzchala: *Standard multi-survey shells in NASS*. in: **Essays on Blaise 1993**, First International Blaise Users Conference (1993), pp. 133-142.

---

[i] WENZOWSKI, M.: *ACTR: A Generalized Automated Coding System*. **Survey Methodology**, 14,2, 299-307.

[ii] TOURIGNY, J. and MOLONEY J.: *The 1991 Canadian Census of Population Experience with Automated Coding.* Statistical Data Editing, Volume 2. United Nations Statistical Commission, Economic Commission for Europe, Conference of European Statisticians. United Nations, New York.

[iii] WENZOWSKI, M. et al. (1995). *ACTR v3 User Guide*. Internal Document, Research and General Systems, System Development Division, Statistics Canada, Ottawa, Canada.

[iii] **Author's Address:**
Research and General Systems
System Development Division
R. H. Coats Building, Section 13-A
Ottawa, ON K1A 0T6 CANADA

Tel.: 613.951.1347
Fax.: 613.951.0607
email: wenzows@statcan.ca