Developing a Blaise Instrument for the Spanish Bladder Cancer Study

Richard Frey, Westat, U.S.

I. Introduction

The Spanish Bladder Cancer Study (SBC) is an occupational health epidemiologic study that has involved development and programming of an innovative Blaise computer assisted personal interviewing (CAPI) system for administering, in Spanish, seven core health study sections and sixty-three occupational module questionnaires. There are other study components that involve hospital case ascertainment and control selection in multiple hospital locations throughout Spain, along with a self-administered dietary history questionnaire and blood, urine and toenails specimen collection and shipment. The study is, however, mainly characterized by a large Blaise CAPI system with technically complex programming requirements, an extremely tight design and development schedule, and international study implementation challenges.

It was the development and implementation of the programming requirements that presented the Blaise development staff with the technical challenges this paper addresses. Challenges include instrument size and complexity, moving from a DOS environment to Windows, the design of a many-to-many data relationship, some bugs, instrument change restrictions, and data export.

Finally, this paper will address some of the major keys to the success of the Spanish Bladder Cancer Study CAPI system and how the staff organization, the development process, and the working relationship with Statistics Netherlands contributed to its successful implementation.

1. Study Background

Westat is assisting the Occupational Epidemiology Branch of the Epidemiology and Biostatistics Program, Division of Cancer Epidemiology and Genetics of the National Cancer Institute (NCI) in conducting this interdisciplinary case-control study of bladder cancer in Spain. Westat has established a subcontract with investigators from the Institut Municipal d'Investigacio Medica (IMIM) in Barcelona, Spain, who are conducting the study to evaluate the etiology of bladder cancer, particularly in relation to external risk factors such as occupational and environmental exposures.

The case respondents are patients who have been diagnosed with bladder cancer in Spain, and control respondents are those who have been diagnosed with other specified diseases and conditions The personal interview typically takes place in a hospital room in several collaborating hospital study centers in Spain.

2. Topic of this Paper

This paper is limited to the technical challenges and the solutions of implementing this survey instrument in Blaise. It does not address the research, methodology, or other aspects of the project.

II. Challenges in Developing the SBC Instrument

There were several technical challenges that had to be met to put the Spanish Bladder Cancer (SBC) survey instrument into Blaise. These challenges included the size of the instrument, many-to-many data mappings, enforced backward movement to jump from a work history table to a selected occupational module, English speaking programmers developing for Spanish-speaking interviewers, efficient implementation of so-called time spent blocks, and programming a custom occupational coding procedure.

In addition, three other factors added complexity to the design of the instrument. First, the initial development and statistical production use of the instrument were to be in Blaise III, the last DOS version of Blaise. While this version has very large capacity, we did not feel we could not execute some potential solutions that might work much better in a purely Windows environment. Secondly, while we knew the basic structure of the instrument, it was designed while many of the sections and most of the occupational modules were still being specified. Thus at the time of instrument design we did not have a good idea of eventual size. Third, we were aware that the interviewers would not be using the most powerful laptops available. An alternative architecture, for example, could be to tie together a main instrument with 63 separate instruments, one for each occupational module. But then we would have had the overhead of 64 different instruments in computer memory and in a management system. If we were to start from scratch today, given the wealth of project experience and the much more powerful Windows version of Blaise, and the advanced state of computer laptops today, we may well do some things described here differently

1. Scheme and Size of Instrument

The SBC instrument has two main parts. The first part is the main part of the instrument with a succession of sections for demographics, tobacco use and coffee/other beverage consumption, occupational history, residential history and environmental background, medical history, family history and quality of life/personal information. This main part of the instrument, by itself, would qualify as a medium- to large-sized questionnaire of some complexity.

The second part of the instrument, 63 modules that were designed to assess occupational exposures for selected job titles, gives the overall instrument a totally different quality. Each of the occupational modules can be considered to be a questionnaire in itself. On paper, the modules range in size from 2 or 3 pages (for a street vendor) to over 70 pages (for a welder or farm worker), with about 8 to 10 questions per paper page. The overall number of pages for a paper questionnaire, of only 1 copy of each module, is 1,675. However, there can be up to 10 instances of each module, so the approximate number of pages for the full paper questionnaire is about 16,075. This would be about 1 meter tall (using a very conservative estimate of pages per centimeter). If you take into account that the survey was done in two languages, then the overall paper questionnaire is 2 meters tall.

It should be mentioned that the bulk of the paper questionnaire involves measuring the time spent on a particular activity or time spent in exposure to a substance. Every time it is necessary to measure 'time spent,' there are several questions that have to be asked to make sure that an accurate measure is achieved. The questionnaire's design resulted in a reasonable average length of administration, approximately 90 minutes. The following technical descriptions illustrate the size of this Blaise instrument.

1. Overall counts	Value
Number of uniquely defined fields*[1]	22,314
Number of elementary fields*[2]	18,941
Number of defined data fields*[3]	119,896
Number of defined block fields*[4]	3,373
Number of defined blocks	224
Number of embedded blocks	108
Number of block instances	20,877
Number of key fields	4
Number of defined answer categories	2,476
Total length of string fields	1,677,270
Total length of open fields	0
Total length of field texts	1,128,212
Total length of value texts	52,292
Number of stored signals and checks	161,511
Total number of signals and checks	161,511

*1) All the fields defined in the FIELDS section
*2) All the fields defined in the FIELDS section which are not of type BLOCK
*3) Number of fields in the data files (an array counts for more than one)
*4) Number of fields of type block

2. Data fields, number fields in data file	Number	Length
Integer	30,729	102,482
Real	14,156	85,099
Enumerated	45,931	46,788
Set	443	611
Classification	0	0
Datetype	1,397	11,176
Timetype	1,408	112,264
String	25,832	1,677,270
Open	0	0
Total in data model	119,896	1,934,690

In addition, there are thousands of pages in the Blaise instrument, it is in two languages, and there are over 100 ASCIIRELATIONAL output files.

How the Interview Works

For each respondent, the interviewer completes the first two sections on demographics and tobacco use, coffee and other beverage consumption. The third section consists of the occupational history and module questions. This information is collected in a table of about 40 columns by 20 rows. Each row collects descriptive information about each job the respondent held in his life for a period of 6 months or more. For each job the instrument collects start date, end date, name of employer, job title, job description, names of chemicals and tools worked with, and other information. From several of these descriptive fields, the interviewer codes an occupation with the aid of a classification procedure programmed in Blaise. If the occupation matches any of the 63 modules, the module is to be collected immediately. In other words, it is necessary to jump from the table in the occupational history section to another part of the instrument and then back to the next row in the table once the module is completed. After the occupational section (and all appropriate modules) is completed, the interviewer proceeds to the next sections (residential history/environmental background, medical history, family history, and quality of life/personal information) and finally exits the instrument.

2. Many-to-Many Data Relationships

Each row in the occupational history table can map to any of the 63 modules in the instrument and this mapping can occur more than once. For example, the first job may be *welder*, the second *fork lift operator*, and the third *welder* again. This model is more of a relational data structure than a hierarchical data structure (which Blaise explicitly supports). Blaise does not naturally support a relational data structure, so a programming strategy was devised to deal with this problem.

Programming Pointers

The essence of handling a relational data structure is in defining pointers from one data block to another and keeping track of the pointers. In every row of the occupational table, there are fields for module acronym and module instance number. These two fields together point to a specific instance of the appropriate module block where the data for that module are collected. In the module itself are fields that keep track of the instance number of the block and the number of the row of the table it is connected to.

A summary of these pointers is kept at a high level in a block called *BookKeep* with the fields *Counter*, *Module*, and *Number*. The *BookKeep* block is an array that tracks the number of instances that a module is used in the instrument. The rules of the data model both within the occupational module and a higher level keep straight which instances of which modules connect with which line in the occupational table. While the block *BookKeep* is simply defined, the rules to make the module and instance number assignments are complex and took a great deal of testing.

3. Jumping to an Occupational Module

As we began constructing the instrument, it became apparent that there would be two limitations that we had to overcome. The first is a hard limitation of the number of Blaise pages (or FormPanes). The maximum page limitation of Blaise is 16,000 pages. We found this limit the hard way by trying to prepare the instrument with all instances of all modules appearing in a Blaise page. In the process of troubleshooting the page limitation problem, we discovered a second problem with instrument performance. We found that when you try to move across thousands of Blaise pages to arrive at the correct instance of the appropriate module, processing time might be slow, taking many seconds to get to the appropriate module.

Ask Module and Copy To Holding Modules

The solution to the these problems was to put only one instance of each module in a Blaise page. This process became known as the *Ask* version of the module. When it became necessary to jump from the occupational module, the interviewer traveled to the *Ask* instance of the module. For each *Ask* instance of a module there are 10 holding instances. When the interviewer is finished collecting data for a module, the data from the *Ask* module are copied via a Blaise block copy to the proper instance of a holding block for that module. This solved the two problems mentioned above. First, we could prepare the instrument with all 63 modules, and second, we increased the speed of the jump from the occupational table to the appropriate asking block.

When programming these complicated movements, you need to manually adapt that which Blaise normally does for you. For example, if you want to re-enter a specific instance of an occupational module, you have to cause the Data Entry Program to copy the appropriate block instance from the holding module to its *Ask* instance.

We also had to cause the cursor to jump back automatically over several hundreds of thousands of pages. This task is done by bringing an *Ask* module onto the route and with an empty field that has the NOEMPTY attribute. While this backward jumping technique is introduced in the Blaise Developer's Guide, it is not something you implement casually. This technique itself was well tested.

A bug was found in the block copy from the *Ask* block to the holding blocks. In Blaise III and in the first Windows versions of Blaise, remarks would not be copied with a block copy from one instance of a block to another. This problem was identified during the survey, and the solution to this bug is discussed later in the paper.

4. ¿Habla Español?

Programming the survey instrument in two languages presented its own set of challenges. Since the survey was to be conducted in Spain for a Spanish-speaking population, the instrument had to be translated into Spanish, a foreign language for most staff on the project at Westat. In order to meet the language requirements of the survey, specifications and paper versions of the survey instrument were first produced in English, and programming of these sections similarly followed first in English. In this way, the overall survey development proceeded first in English. As development proceeded, concurrently English specifications were sent to translators in Spain where each section was translated into Spanish. When the translation for a section was completed, it was returned to developers for implementation in the Blaise program.

Parts of the instrument that had to be translated into Spanish included question text, response categories, hundreds of thousands of text fills, interviewer instructions and edit messages. While the implementation of the Spanish version had to wait for the translations (which were completed in a timely way), the programming strategy allowed for the copying in of the Spanish text by leaving hooks throughout the instrument source code, making great use of the Blaise ACTIVELANGUAGE feature in the rules section. When the translations arrived, it was another task to copy the appropriate texts from a word processor to the application source code. There were other challenges as well. For example, we had to make sure that the proper 'code page' was in place on the developers' computers so that the diacritical Spanish characters would be recognized in the U.S. (This was a DOS problem.)

5. Time Spent Modules

Much of the instrument asks about the amount of time spent doing an activity or time spent in exposure to a substance. There was a great deal of sameness and similarity in how all 'time spent' constructs appeared. Several time-spent blocks were defined and re-used repeatedly. The most common time spent block, *TimeSpent1*, has the following elements:

```
P1   {Entry cell, gives chance to change mind.}
P1A  {Number of times per day, week, month, or year}
P1b  {Time unit (day, week, month, or year)}
P2a  {Number of hours, or days, or weeks, or months}
P2b  {Second time unit (day, week, month, or year)}
P2c  {Time unit}
P3   {Percent of time}
P4a  {Fraction of time, numerator}
P4b  {Fraction of time, denominator}
P5a  {Number of minutes or hours}
P5b  {Minutes or hours}
P6   {Yes no question}
P7   {How many times}
```

Routing through this construct depends on how the respondent wants to answer a question about time spent in an activity. For example, the respondent was allowed to answer "I did this activity 10 times per day, 5 days per week, for about 20 minutes each time." Or she could say, "I spent half of my time doing that." The interviewers were able to easily administer these questions, with proper routing through the several fields depending at any time on the pattern of responses.

There are several hundred lines of code for the *TimeSpent1* module including complex routing, text manipulations, and several edits that checked reasonableness of answers. This particular module, considering all instancing, is used several thousand times. Overall there are 6 time spent modules, 3 number of times modules, and an assortment of other similar measurement modules that were programmed one time and used all over the place in a very efficient use of Blaise blocks and procedures.

The extreme re-use of these time and number measurement modules meant that the question text and fill text had to be customized for each instance. The following code snippets show how this works. First we imported text before the block call using parameters.

```
PARAMETERS
  IMPORT
    Phrase1, Phrase2, Topic : STRING
```

The field text was defined as follows:

```
P1a "^Phrase1 ^Phrase1a
    @/NUMBER OF TIMES PER DAY OR WEEK OR MONTH OR YEAR.
    @/@/@Y[INTERVIEWER] PRESS <ENTER> TO ANSWER IN TERMS
    OF TIME UNITS, PERCENT OF TIME, OR FRACTION OF TIME."
    "^Phrase1 ^Phrase1a
    @/NUMERO DE VECES POR DIAS O SEMANAS O MESES O AÑOS.
    @/@/@Y[ENCUESTADOR] PRESIONE <ENTER> PARA RESPONDER EN
    TERMINOS DE UNIDADES DE TIEMPO, PORCENTAJE DE TIEMPO, O
    FRACCION DE TIEMPO." : 1..9990, EMPTY
```

where *Phrase1* was imported into the block and *Phrase1a* was computed in the block if necessary. A few within-block computations of text strings are shown next.

```
IF ACTIVELANGUAGE = ENG THEN
  Fraction := 'FRACTION OF TIME'
ELSEIF ACTIVELANGUAGE = ESP THEN
  Fraction := 'FRACCION DE TIEMPO'
ENDIF
```

6. Occupational Module Selection

The occupational table collects descriptive information about the kind of job that was actually being performed. This method of collection also identified key aspects of the job that could be used for selecting the most appropriate occupational module programmed into the instrument. The descriptive information consisted of job title, main activities and industry description, as well as other items such as 'chemicals exposed to' and 'tools used on the job.' For each job, the instrument was required to first collect and then inspect these descriptions, and suggest to the interviewer possible occupational modules that might be mapped to the job.

The first attempt to do this process was with the Blaise trigram coding scheme. This approach did not work mainly because the concatenation of all the descriptions could be hundreds of characters long. This length of text string feeding into a trigram search does not work well (the trigram was not meant for this purpose). The solution was to create a procedure that could take the concatenated descriptions, and inspect them for exact matches to a list of hundreds of keywords. A score is kept internally to the procedure and it then displays the modules that possibly match. The interviewer, in consultation with the respondent, can accept any of the suggestions or override them.

This was first programmed and debugged with English. Then the keywords were translated into Spanish and that version of the procedure was used in the instrument. We found out, among other things, that the effectiveness of the procedure is somewhat dependent on the language. For example, precision in spelling is key to the process. Spelling a Spanish word without the appropriate diacritical mark causes a problem. Overall, however, this process works well.

7. Meeting the Technical Challenges

There were several aspects to meeting the technical challenges of developing such a demanding instrument. First, the clients (both in the U.S. and in Spain) and the project personnel at Westat did an excellent job of communicating the needs of the survey and in specifying it. Additionally, they were able to identify challenging aspects of the instrument in advance. Secondly, there was very strong prototyping of solutions for all of the above mentioned challenges and more. These were done with so-called mini data models. These mini data models allowed the programmers and the project people to focus on particular issues and specific parts of the instrument, and facilitated iterative development to a solution. Thirdly, the instrument as a whole was developed in terms of mini data models, with well over 80 of them that are used. Given the size of the instrument, it was inconceivable that it could have been developed any other way. Using mini data models allowed programmers to work on different sections simultaneously, and allowed their sections to be compiled more quickly. The use of explicit parameters to connect completed blocks into an integrated instrument made it very easy to link these modules into a coherent whole. If a repair has to be made to a module it is done with the mini data model, and then re-integrated into the main instrument.

Simultaneous Top-Down and Bottom-Up Development

At the same time the prototyping and the module development were undertaken, a parallel development with the overall instrument structure took place. There were several issues in tying things together that had to be worked out. Most integration issues were resolved when there were only a few of the questionnaire sections and modules finished (indeed, even before most modules and sections were specified). This meant there was effective top-down and simultaneous bottom-up development of the instrument.

Standards and Use of Expertise

Other aspects that were critical for the success of the instrument development were the early setting of programming standards and ways of working. There were code reviews early in the process to make sure that methods developed in the prototyping were applied correctly to production code. In addition, the project drew on the experience of Blaise experts at Westat.

III. In the Beginning There Was Blaise III for DOS

The development of the Spanish Bladder Cancer instrument was undertaken first in the final DOS version of Blaise, that is, Blaise III. This was because the Windows version was not yet even in beta mode. In fact, the beginning of the survey was conducted in Blaise III. Blaise III has some limitations that affected the survey, and those limitations and their solutions are discussed here.

1. Memory Limitations

The instrument was first prepared and fielded with 48 of the 63 modules in place, but in both English and Spanish. As the instrument was tied together in full for the first time, we found a memory limitation that prohibited it from running in a DOS window under Windows (any version). As discussed above, the instrument had already been as efficiently designed as possible, so there wasn't very much inefficient code to optimize, and we wouldn't have had the time anyway. Since the survey is operational only in Spain, and in that sense English is not necessary, it was decided to comment out all the English language text to see if that would allow the instrument to run in a DOS window. However, as described above, the language statements and manipulations appear all over the source code. It would have been impossible to comment out all the English text and computations by hand.

Automated Parsing and Commenting Out of English-Language Code

Since the instrument authors programmed according to well-defined standards, it was possible to comment out the English language text and code by automated means. In a few days time, a senior Blaise engineer programmed a Manipula setup that was able to parse all the source code files and comment out English text and manipulations. The virtual perfect implementation of the source code according to standards enabled the Manipula program to search for a do-able set of patterns and keywords and comment out the English code wherever it found it. For example, it could search for the phrase IF ACTIVELANGUAGE = ENG and figure out where to put the beginning comment brace and where to put the ending comment brace. In actual fact, this parsing and automated commenting out of the English proceeded with few problems and the first operational version of the instrument was fielded on schedule.

While it was possible to field the 48-module instrument in Blaise III, it would not have been possible for all 63 modules. The timely arrival of the Windows version of Blaise enabled the instrument to handle all 63 modules and work again in both languages.

2. Other Limitations

Some limitations in the DOS version of Blaise affected the project, including overall block size and the size of the rules section. We found, for example, that the absolute limit on size of a rules section, including programmers' comments, is 64Kb. And we found that some blocks were too big to parse. The solution to these problems was to break big blocks into smaller blocks and to program some rules sections as efficiently as possible. Some of these limitations extend into the Windows version of Blaise.

Another limitation is with the size of a TYPE section that can be compiled with other instrument code. There are almost 2,500 type definitions. When the type definitions were incorporated into the Blaise source code with an INCLUDE statement, the instrument would not prepare. The solution to this problem is to pre-prepare the type library. This simple step not only allows the preparation of the instrument, but also speeds up preparation considerably.

IV. Progressing into the Windows Version of Blaise

The eventual arrival of the successive Windows versions of Blaise provided a more robust platform for this project. The Windows version of Blaise provides much better memory management and 32-bit processing, both of which are indispensable to the project. With these capabilities, Blaise can now handle all 63 modules and both languages.

Windows 95 has its own large memory requirements and so each machine was upgraded to 48 MB of RAM. Though Blaise in Windows has a graphical user interface, the system operates in much the same way as it does in DOS and thus the transition from the DOS version to the Windows version was relatively smooth for the interviewers.

Even before the first Windows version was released, the developers began using the Windows developer's environment to produce the DOS instruments. They found, as have others, that the Windows Control Centre is much more powerful than its DOS counterpart and could prepare the modules faster. When it was time to produce Windows instruments, the project verified that there is true upward compatibility between the two versions.

Bugs

As one of the world's first surveys to be fielded in Blaise for Windows, the SBC project found several bugs in the software. One of these is worth mentioning here. When copying a block of data from an *Ask* block to a holding block, the copy did not include the copying of remarks. Westat reported this problem to Statistics Netherlands and they corrected this bug. During the interim, a manual work-around procedure was implemented in Spain where the interviewers recorded their remarks in any occupational module separately. This quick response of Statistics Netherlands fixing important bugs was very important for this groundbreaking effort.

New Features in Blaise

The Windows version of Blaise also came with new features that were found to be very useful. First was the audit trail that is not available for the DOS version. This provided a means of data backup for lost remarks (see above) and for debugging some problems. Another feature that is the new ability to tie an executable program to a function key. This was used to enable an interviewer to invoke another data entry program and instrument when the respondent was getting tired or starting to refuse the questionnaire. This is called the *Critical Items* questionnaire. The idea is to get key data from these respondents if possible. Since this situation could arise at any point in the interview, the ability to tie this to a function key was very important. This could have been executed through a parallel block as well, but then the data model definition would have changed and the project wanted to avoid that.

V. Operational Considerations

There are some operational issues with Blaise that were important for the SBC to anticipate and manage correctly. These include performance of the instrument on the supplied hardware, versioning of the data models, use of Maniplus for laptop survey management, SAS limitations, and supporting operations in Spain.

1. Instrument Performance

The laptop in use in Spain has 48 MB of RAM and 133 Mhz processors. These were not top-end machines even when they were purchased and are considered slow by today's standards, but the huge data model runs fast and well. The key to this performance was the advanced appreciation of the challenge, expert knowledge of the Blaise selective checking mechanism, and programming standards that met this challenge. The key to improving performance in a large data model is to reduce the amount of parameter administration between blocks, and to reduce the number of blocks that are checked at any one time. This is an advanced topic not suitable for this paper, but if the selective checking mechanism is taken into account, you can implement large and complex data models.

2. Data model Data Definition Changes

It is a fact of life in the Blaise world that even relatively minor changes in an instrument can result in a change in data file definition. When such a change happens, it is necessary to translate the database definition from the old to the new manifestation. This process can be automated on a laptop and in the home office, but there is still a versioning issue. That is, with several data files in different locations, how can you be sure which database is in which version? This challenge was met with strict versioning procedures and planning for version changes. Changes to the data model were allowed more frequently if the data file definition did not change. If a data file change was necessary, the change had to be strictly executed and scheduled.

3. Survey Management on the Laptop in Maniplus

A Maniplus survey management system was used on the laptop to control access to the cases, data transmission, and other aspects of operations for the interviewer. The use of Maniplus to execute the instrument and the survey management is crucial to this survey given the combination of low-end hardware configuration and the size and complexity of the instrument. The advantages of Maniplus for this are several fold: the Data Entry Program and the Manipula are part of the Maniplus executable file, and when the instrument or a Manipula function is executed, it is not necessary to invoke another executable. Maniplus also has full knowledge of Blaise metadata and can write to and read from a Blaise database. It also loads the instrument ahead of time and this allows the interviewer to start the interview quickly if necessary.

Another decision was to put each data record in its own zip file. This decision was made because of the tremendous size of each record. We did not want to have to try to rebuild a data set with Hospital in case of data corruption, though this latter possibility has not been a problem.

4. SAS Output and a (Former) SAS Limitation

Due to the size of the data model, data are exported from the instrument with ASCIIRELATIONAL output. Over 100 output tables are produced. A former SAS limitation, that of 8 maximum characters for SAS VAR names, had to be overcome because some of the Blaise field names were over 8 characters. Another challenge was to get SAS data descriptions for the ASCIIRELATIONAL output. The Blaise system does not come with a totally automated way of getting the data description for each output table. With over 100 tables to take into account, even a one-time effort to produce descriptions by hand would have been very tedious. When you consider data model changes, the potential to have to redo these by hand becomes impossible. The solution here was to develop programs in Cameleon, Manipula, and DOS BAT files that totally automate this process.

5. Supporting Operations in Spain

There were several challenges in Westat's support of operational data collection in Spain. These problems included language differences, time zone differences, and long distance support. When an operational problem occurs, getting an accurate description of the problem is the first step at diagnosing and resolving the problem. Reporting the problem clearly and succinctly in two different languages does not always happen immediately, and sometimes additional time is required in iterations just to understand the problem. Also, the issue of a six-hour difference in time zones becomes another factor to weigh in responding to problems in the field. Even with these challenges, the project has proactively addressed problems with the cooperation and good will of everyone involved.

VI. Keys to Success

The success of the Spanish Bladder Cancer Study CAPI implementation can be summed up into these three key areas: the organization of staff; the development process, and the working relationship with Statistics Netherlands.

The organization of staff was simple in design but powerful in its operation. Heading the development side was the senior integrator. Below the integrator were five programmers each assigned to the section and occupational module programming. In a staff position was the independent test team, which also filled the role of specification writer. A senior Blaise technical expert gave strategic advice on design and instrument development methodology. The key, however, was the senior integrator. While not a Blaise expert, it was the integrator who understood the importance of following established standards and processes.

The key in the development process was that all standards and processes were defined up front. Based on requirements and the design it was decided that each section and each module would be developed and tested as its own data model. Later, each section and module was incorporated into the instrument 'mainline.' For each occupational module, 90% of the questions asked involved measuring how long it took to do something or how long an exposure there was. Type blocks and procedures were created and eventually used many thousands of times in the instrument.

The independent development of the sections and occupational modules gave way to the independent testing of each section and module. This testing process enabled the testers to focus on the results of each section and module test without concerning themselves with influences from other components of the system.

A final key was our working relationship with Statistics Netherlands. Whenever there was a problem, such as remarks not being copied from one block to another, or the Don’t Know/Refused symbols not being displayed, we were given a prompt response and in many times a version of Blaise with a fix to the problem. This relationship allowed us to continue developing the instrument without having to worry about 'a work around.' In addition, it built credibility, not only with those doing the development, but also with our client.