A Process for Blaise Data Emulation of Complex Instruments

Todd Flannery, Ed Dolbow and Curtis Cooper, Westat

As instruments grow in complexity, difficulties arise in testing routes due to the sheer number of combinations of data items and the relationships among those items. This complexity makes it difficult for specification testing to cover every one of thousands of routes through an instrument. It limits the efficiency of prescribed scenario testing, in that only those planned scenarios can be developed and tested, resulting in potential gaps in coverage regarding some novel combinations of data items. Although automated regression testing can identify the effects of changes to an instrument from one test or round of tests to another, it will not account for persistently problematic routes or data combinations. Data emulation of many records can assist in producing a test base that is more comprehensive and thus more representative of possible combinations that can occur in a field study, however unlikely.

Large scale testing involving multiple instruments with many interdependent inputs, or longitudinal data collections with multiple rounds of administration add to the difficulty of ensuring full coverage of the instrument during testing. These testing efforts also require additional coordination among different participants within an organization. Each of these factors increases the difficulty of ensuring full coverage during testing.

This paper will present an approach implemented on a large national longitudinal study to test widely varied routes through a complex Blaise instrument. Westat applied two data generation tools and scripting to a large survey instrument containing 1700 items and more than 2500 unique variables (multiple response and other specify categories). Blaise software offers two utilities to emulate data in an instrument called BTEMULA.EXE and BLEMU.EXE The application of the emulator tools warrant the use of scripting to address integer variables, date comparisons, hard edits and so forth. The National Standards of Institute and Technology (NIST) in the US offers a software product called ACTS (Automated Combinatorial Testing for Software) to generate data based on combinatorial theory. These generated data were one of three sources of inputs to the emulator along with the variable values produced by the scripting process. Data from previous rounds of data collection and scenarios were the other two sources. The use of inputs, scripting and emulation produced large numbers of randomized data records and provided a method to enable more robust analyses of questionnaire items.

Theory

Generating large quantities of records with randomized routes through a Blaise instrument can identify errors that may go undetected via more traditional approaches to testing, such as specification testing, scenario testing, and regression testing. The capabilities of the Blaise Emulator Tool, as applied to the test data within an iterative process cover more routes and data combinations than these traditional approaches, ultimately result in increased integrity of instruments, and study data.

Methodology

Westat applied several methods to preparing the preloads for an instrument. One set of inputs (preloads) were based on responses to the survey in a prior year as well as responses to other instruments during the same administration. The latter category typically involved management variables intended to avoid repetition. For example, several instruments had contact sections at the end that one needed to be answered only once during the administration.

Virtually all of the preloads were derived variables based on a surprising number of questions asked in prior years. These variables were based on specifications approved by the client and extensive programming. The process was time consuming, and the results were not available until later in the development process for the next wave. Nevertheless, these data were a valuable source for the emulation testing and testing performed by the corporate test team.

The second source of data were the variables generated by combinatorial logic of the ACTS software NIST has performed extensive research on the use of combinatorial methods to uncover system failures. Many system failures are caused by a single factor or a combination of two factors. The interaction effects of three or more variables cause a smaller but significant number of failures. NIST research indicated that single and two-factor testing accounted for 67 to 93% of the failures but that four to six way testing accounted for virtually 100% of failures. Developing test data to meet requirements for interaction testing requires many observations. 10 input variables each with two values generates 1024 test cases if each test case covers a single three-way test. Figure 1 demonstrates how the NIST software uses trios of the data items (the columns) to cover several combination groupings (red, yellow, blue) with each three-way test[1]. Each row can represent a row of input prior to emulating record-level random data.
In this example, the ACTS software requires only 13 records to cover all of the three way interactions. Westat used ACTS software to generate four-way interactions because routes through the instrument required many conditions.

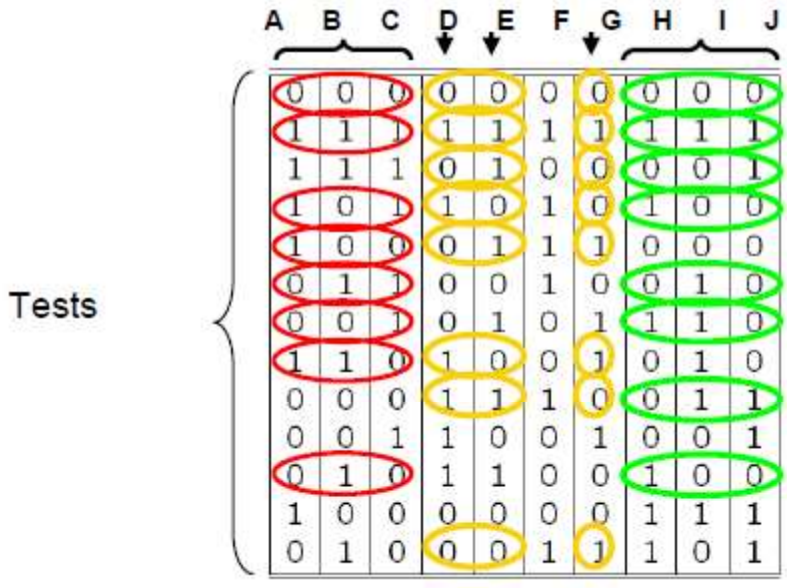

Figure 1 – Multiple tests on one row [1]

There are several advantages of using ACTS data rather than scenarios or longitudinal data. It ensures wide coverage of all possibilities some of which may not have occurred in the first half of the prior way. The original ACTS code took several days to produce but it was modified easily using either the ACTS

interface or edits of the XML file produced by the software. The data can be produced on a timely schedule in less than two days so the method is kind to staffing resources and scheduling during periods of intensive development.

The Blaise Emulator

Statistics Netherlands provides two tools to emulate data, BTEMULA.EXE and BLEMU.EXE. The BTEMULA program is single-threaded, meaning it handles a single record at a time, and the BLEMU program is multi-threaded. The programs can be used to stress-test CATI instruments, but we were more interested in the capability to generate randomized values for all FIELDs on the route of an instrument. The tools execute via a command-line with an option file or a user interface with settings (Figure 2).

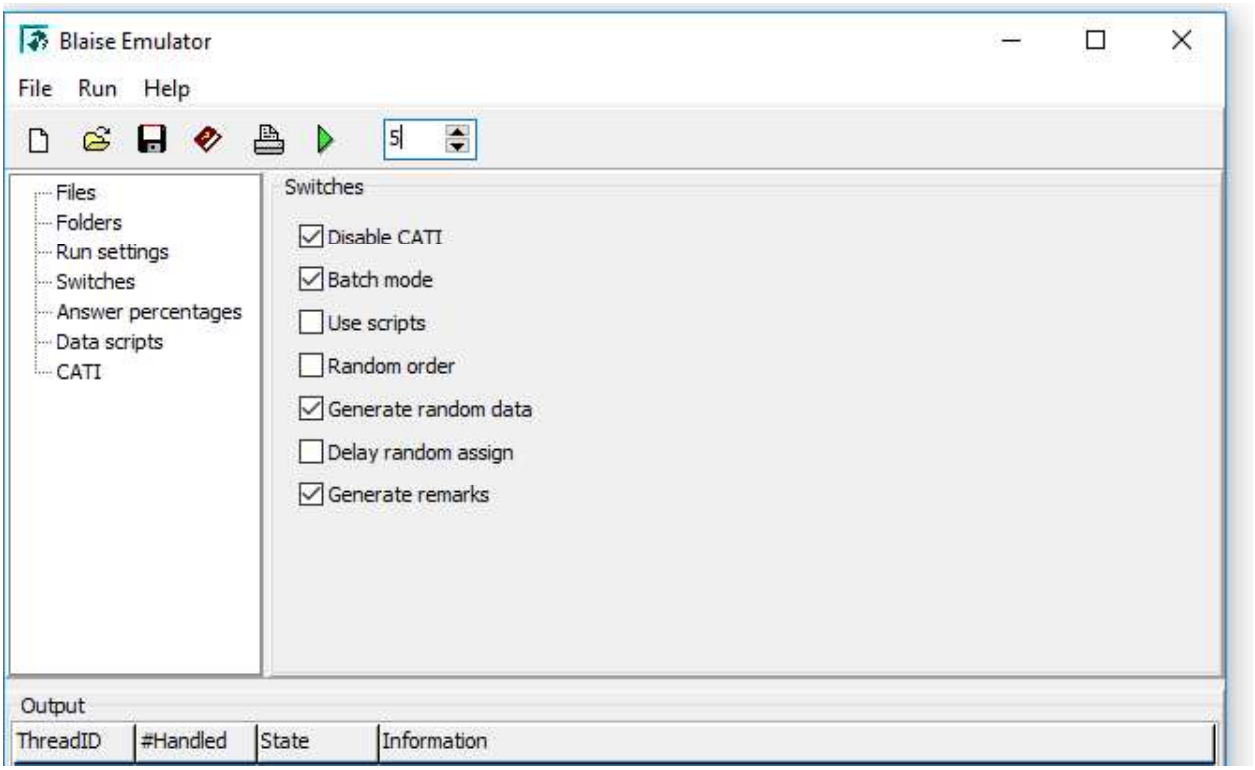

Figure 2 – Blaise Emulator Tool Interface

As we begin to emulate data for a complex datamodel, some problems arise due to the nature of data and relationships among the data items:

- Inputs are often related to one another, and randomization of values for these inputs does not consider constraints that may exist in the imported data items.
- Randomizing values for certain items, like dates, times, or statuses that are stored as strings will likely create data that is not usable or causes problems with routing in the instrument.
- Eligibility determined by screening questions or sampling algorithms may result in few or no eligible respondents, and this will limit the effectiveness of randomization on most routes.
- Values that are created in randomization are based on the *definition of the FIELD*. The emulator does not use hard or soft edits (CHECKs and SIGNALs) in determining the random values. This can result in setting many outrageous values for FIELDs defined with wide ranges of integers.

We have tools that mitigate each of these issues and produce large-scale data that can provide valuable information regarding the integrity of our code or specifications. To begin, we isolate any inputs to the randomization process from the emulator, since the emulator tool does not overwrite existing data with emulated values. This will control and enforce data integrity for those inputs. We later discuss some methods for creating values in the input data. Additionally, we can avoid some of these errors in emulated values by scripting particular combinations of fields and prescribed values. We apply these tools as steps in the iterative emulation process.

Process

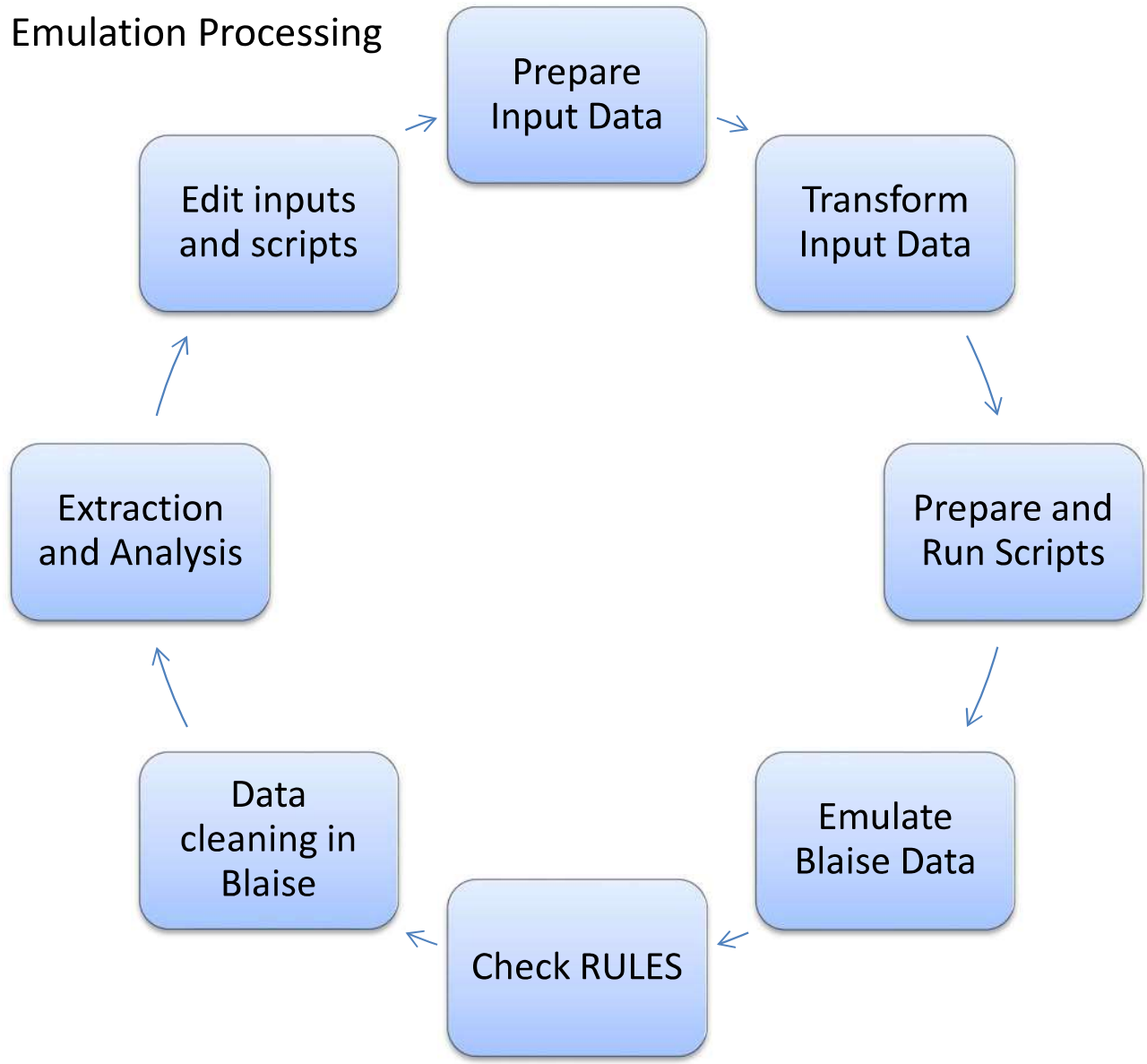

Figure 3 – Processing steps for emulation of data

Preparing the input data

Inputs can be comprised of any data from external sources, such as management data (statuses or demographic characteristics of a respondent), sampling rates or flags, or data collected and passed in from other Blaise instruments. Sources of the input data can be:

- Derived via a combinatorial approach. The NIST combinatorial approach allows us to apply constraints to the input data in order to maintain relational integrity among the input items.
- Imported using Blaise data via prescribed scenarios – a planned set of testing inputs are preloaded into the database via Manipula.
- Imported using existing Blaise data from a prior data collection – prior wave data are preloaded into the database via Manipula.

The data preparation involves creating, in Blaise, any field values external to the instrument that could affect the routes. Next, we copied these records over several times to associate multiple routes through the instrument with each set of inputs. For each instrument, we decided to create about 15 thousand records to pass through the emulation process.

Transforming the input data

The data transformation involves a simple Manipula import to bring our input into the existing datamodels that we are testing via emulation. At this point, we do not need to check the rules because data integrity is assumed intact when the input values are imported.

Preparing and running scripts

Scripting populates the instrument with additional values to provide constraints and limit outlandish values. To minimize the impact of datamodel changes while an instrument is being used to collect data, we intentionally allow broad ranges for continuous values like age. This ensures that we can maintain compatibility with existing data in the case that we decide to update our ranges. Scripting limits these broad ranges to a discrete number of meaningful values. We often use hard or soft edits to handle errors in data collection. In emulating data via BTEmula or BLEmu, these edits are ignored, and we can sometimes end up with some values that would not be allowed via the DEP so we apply scripts, by section. Applying scripts by instrument section allows us to assign discrete values that avoid unusual data that can arise due to broad definitions. This enforces data integrity that the emulator tool cannot achieve without using the scripts. Designing scripts can be an intensive process, based on the complexity of edit checks, so we use experts who know the instruments to determine the discrete values that would affect routing for critical and/or integer fields.

In a sense, the scripting process may belie our stated goal of producing great variation in the data, and some prudence should be exercised in the development of scripts. We use scripts to limit particular data items to exclusive values or combinations of values, excluding these values from being randomly selected by the emulator. Although the selection of any particular script among many is randomized, the values within the script will be assigned to the data. Given this, we recommend keeping the number of FIELDs referenced in a single script limited to a few. Note that we create some scripts to increase *access* to routes that we know could be disproportionately skipped if randomization alone were used. Additionally, scripting can prevent range errors that may skew the data analysis from emulated output.

Once the scripts have been created, we can use the emulator tool to apply them to the data, as is described in the Blaise help for emulation. In our case, we determined that scripts were required for each section of

the instrument and, since we already had some of the existing data in place, we did not want scripting to overwrite those values. So we handled scripting via a Manipula batch process, and this allowed us to have greater control over how the scripts were processed into data values. Our process randomly selects a single script per section and only writes the scripted data where no data currently exists in the instrument for those values. Note that individual scripts can contain logically inconsistent data, as the data will be cleaned in a later stage of processing.

Emulating everything else

After the scripts have been applied to the now mostly-empty data set, we can run the emulation step via BTEmula or BLEmu. Emulating all of the remaining data in the datamodel can produce some inconsistencies and/or errors that may not occur in the real world. In emulation, all possible values are equally likely, and this can produce some unexpected results. Here are some considerations:

- Continuous problem – as described above, defining broad ranges produce many outrageous results (e.g., the 500-year-old man or someone who smokes 900 cigarettes each day.)
- Strings can take on any values depending on their length, and results may not be useful.
- Randomized dates can lead to date combinations that make little sense or generate errors.
- Sampling algorithms or screener questions can produce few or no results (or respondents) due to randomizing field values that tend to almost always have a positive result in the real world (e.g., enumeration for "Does the respondent consent?" having values "1-yes, 2-not now, 3-soft refusal, 4-hard refusal") will result in a lot of missing data.

Through successive iterations of the process, we can address issues that arise via creating scripts that will address these data issues individually.

Checking the RULES

We ensure clean routes by running a Manipula script with these settings:

```
CHECKRULES = YES
DYNAMICROUTING = YES
```

At this point, we can separate records where hard or soft errors are found. Trapping these errors informs future iterations of scripting.

"Cleaning" the data

Prior to extracting the data from Blaise, we need to clean some values created as a result of randomization in the emulator. This process requires another Manipula program that uses recursion to parse both the datamodel and the data values. This customization is largely dependent on project needs. Some tasks:

- Cleaning up strings to use a single value, like "ZZZ", since these are not often useful for analysis and don't affect skips in the instrument.
- Setting limits on SET-type responses to accept only 1 or 2 selections, if preferred.

- Iterating through pairs (or trios, etc.) of values that are producing hard or soft edits.
- Removing values where the enumerated response text was intentionally blanked-out.

We can use results from the cleaning process to inform scripting for a subsequent processing cycle.

Extracting and analyzing the data

For our project, Westat converted the data to SAS for analysis. In the SAS analysis, our project and clients define populations and routes that have known probabilities. The emulated values are compared to the expected values to determine if routes are being missed, administered improperly or expected ranges in the data are exceeded. In this way, we can determine if an error exists, and then once determined, we can investigate whether these errors are related to specifications or coding.

Editing inputs and scripts

Instrument experts, Blaise systems staff, and data experts use the results of this analysis to:

- Identify problems in the routing or specifications.
- Determine how to construct or edit inputs and scripts that will allow for the most value in terms of coverage of routes in the data.

These edited inputs and scripts are then applied to our edited datamodel as we begin the emulation process with a new set of test records.

Results

Our first pass through the emulation process utilized an approach where an expert tester entered scenarios, and the remaining data was emulated. This produced about 15,000 test records, each containing 4000 columns of Blaise data. The iteration took about 3 working days, with processing steps for building and running scripts and the single pass through emulator program requiring the most. As a result of the SAS analysis, we were able to identify over one hundred areas of interest for further investigation within the instrument. Since our focus is in improving performance of the process through successive iterations, it is likely that this first pass will help most in determining where emulation scripting and cleaning can be improved.

Our second pass through the process utilized the longitudinal approach, using collected data from an earlier wave of data collection to populate the inputs and perhaps produce results likely to be encountered in an upcoming field collection. We generated 12,500 records, and the analysis again provided many results for further investigation.

We also were able to run the NIST ACTS tool to generate two and four-way combination records. The records were imported as inputs to our data, and we generated x records for analysis. For the 2 factor test, ACTS generated 85 records covering tens of thousands of combinations. For the 4 factor test, ACTS generated 4717 records covering tens of millions of combinations. The software is very efficient and produced even the 4 way combination in minutes.

Considerations

We needed to make adjustments regarding the scripting and editing some emulated values. Scripting for data where values already exist will overwrite the existing value with the scripted value, and we preferred the pre-existing values to take precedence over the scripts. In addition, the scripts needed to be in the same folder as the emulated data, and our number of scripts per section was getting burdensome enough for us to use a batch process with a Manipula program to be able to have more control over the scripting process. Our customized process randomly selects a script from each scripting folder to write its contents to the active record. This allows us to script more data in a single run through all of the emulated records.

Emulation via the multi-threaded BLEMU.exe or the single-threaded BLEMULA.exe allows values to be emulated where response text has been suppressed. In coding, programmers sometimes pass blank strings as response text in order to conditionally hide or disallow an inconsistent response. The emulation tool ignores these restrictions and may use the value in the emulated data, so we reset these values to non-response when we come across them, as part of our data cleaning.

Recommendations

The ability to assign weights to scripts or emulated values could potentially provide better coverage of targeted combinations or groupings. As it stands, each script or value is treated equally, and this does allow for variety covering a full range of responses. It is possible to mimic the behavior of weighting scripts by creating many scripts with preferred data. However, it would be better to assign weights to particular scripts to allow a higher prevalence of that combination. The weighted scripts could be used to produce data more in line with known prevalent trends.

Using experts with different backgrounds improves how the scripts or input data can be constructed between cycles of emulation. We need to restrict the inputs or scripts to provide meaningful results, but we also seek to produce a multitude of varied data in order to maximize coverage of the routes though the large-scale method of data emulation. Instrument experts have intimate knowledge of the desired combinations that provide insight regarding prevalent routes or key fields that determine crucial skips in an instrument, whereas data experts use the analysis of the emulated data to develop frequencies that detect route errors.

We can further adapt the process of emulating random data for special situations by using Manipula features to handle some of the exceptions we noted. Being able to access error information will allow us to re-randomize when a hard or soft edit has been raised, and we can determine if responses are excluded conditionally to reduce some of the emulation-specific errors that we discover in the analysis stage. Additionally, developing Manipula processing could allow us to adapt the process for Blaise 5 instruments without having to convert between Blaise versions.

Conclusions

Applying a process for emulation of data allows greater coverage of combinations than standard scenario testing or data entry testing. Although the preparation of scripts, processing and cleaning of individually emulated test records can be initially intensive and time-consuming, the iterative nature of the process and subsequent refinement of inputs for cycles of processing can provide high-quality test data for longitudinal or cross-sectional instruments. The more burdensome tasks associated with data entry testing can be alleviated via the emulation process, allowing more time for testing staff to focus on targeted scenarios. Additionally, since emulation can begin as soon as a datamodel is prepared, the data can be

analyzed while testing is taking place, and this reduced "lag time" between data entry and analysis eliminates duplication of error reporting that occurs due to a lengthy test cycle followed by an analysis cycle.

References

D. Richard Kuhn, Raghu N. Kacker, Yu Lei. "Practical Combinatorial Testing" NIST Special Publications 800-142, October, 2010