Testing a Production Blaise Computer-Assisted Telephone (CATI) Instrument

Amanda F. Sardenberg, U.S. Census Bureau
John W. Gloster, U.S. Census Bureau

Abstract

Good testing practices are just as important to a Blaise computer-assisted telephone interview (CATI) system as they are to any other production software system. A rigorous approach to testing can put the system in production sooner and avoid production setbacks in the introduction of enhancements. We will focus on the Blaise CATI system, which supports the Telephone Followup (TFU) operation of the American Community Survey (ACS). The ACS generates over 10,000 problem cases a month for TFU. It is imperative that this system function efficiently and as close to error-free as possible. Careful testing is a key ingredient to making this possible.

We will address the following aspects of our testing approach:

$ Maintained detailed specification: The many specialized aspects of Blaise software can require the same level of complex, lengthy specifications and production testing as a much larger system. Instrument optimization testing would ideally account for every possible path or universe, every possible error condition and flag, and every possible coded outcome prior to each new production release. This level of testing requires detailed specifications and rigorous specifications maintenance.

$ Separate testing environment: Staging grounds, that is, a test environment, must be established in order to exactly mimic the production environment; additionally, if the test environment must be accessible by multiple users other than the administrator, a pre-test environment would also need to be established.

$ Test plan: The different testing paradigms and designs, balanced usage and cyclic versus continous implementation of the respective testing procedures will be explored in more detail using actual experiences garnered while conducting ongoing testing for the American Community Surveys Telephone Follow-Up (TFU) Blaise CATI instrument.

$ Test log: Handling priority items of this nature is most efficiently done in an inclusive, communal manner involving administrative, analytic, and developer staff. For this purpose, a highly detailed tracking log is maintained by the administrators while incorporating analytic tester results and input from developers.

$ Version control: Procedures involving date stamps and structured network storage of code delivery, archived or frozen source code, current data and metadata, etc., also need to be established and followed rigorously for ease of maintenance, tracing, and reinstatement of code, if ever needed.

$ Production simulation: Testing must replicate the production environment in order to accurately assess the viability current instrument fixes or upgrades. Ongoing modifications to a production system also mandate other, more practical considerations that must be balanced with any thorough, systemic specifications and modifications testing.

$ Deployment of new releases of the Blaise instrument: All optimization testing needs to simultaneously be balanced with other efficiency constraints, such as staffing, time to next upgrade of software or instrument, and issue priority. Additionally, new releases of Blaise software require their own testing to assess their fitness in the production environment

$ Diagnosis of and response to problems in the production system: Optimization testing during

development and modification of a Blaise instrument will often reveal pressing issues that must be resolved to maintain minimum required functionality. We have implemented an issues and problems tracking log for speedy resolution.

The ACS will expand in the next few years by a factor of 3.5 over current workload. Testing of the Blaise CATI TFU system will take on an even greater importance with this demanding workload.

Introduction

The American Community Survey (ACS) was begun for the purpose of collecting current demographic information on housing and population data on a continuous basis. The ACS consists of a mailed questionnaire with an automated clerical edit for questionnaires. The initial by-hand determination of what was sent to TFU was automated via a set of algorithms that determine whether a survey passes clerical edit (CEDIT) or fails and is routed to TFU. The automated telephone follow-up (TFU) operation that utilizes Blaise software, and a non-response CATI and computer-assisted personal interview (CAPI) component, round out the survey operations. Within the next few years, the U.S. Census Bureau's American Community Survey (ACS) is slated to become the largest permanent and ongoing survey in the country, sampling approximately 3 million households annually.

Thus, the process of automating TFU became critical as the geographic scope of the survey expanded from a small pool of test sites to using the ACS methodology currently in 1,239 counties across the country, and the time from conception to initial production operation was extremely brief. As a result, the TFU instrument was first operational in an essentially beta form and has required extensive and continual testing to maintain its production functionality and data-handling integrity. Ongoing testing has remained a critical aspect of the maintenance and periodic upgrade of our production instrument.

Specification Maintenance and Testing

The many specialized aspects of Blaise software can require the same level of complex specifications and production testing as much larger systems, regardless of sample size and scope. Ideally, instrument optimization testing will account for every possible coded outcome, path or universe, error condition, or flag prior to each new production release or software upgrade. This level of testing requires detailed specifications that are rigorously maintained and are then used to then develop testing procedures and to baseline the instrument as modifications are introduced. Once the instrument specifications have reached a certain level of development and have been reliably translated into stable instrument functionality, these essential specification verification routines would ideally become automated.

Maintenance of Separate Environments

A production instrument in a remote location mandates that the tester be able to replicate that environment exactly. This is necessary to accurately assess the viability of current fixes or upgrades. Ideally this environment is local and is physically separate from the actual production environment. Staging grounds, that is, a separate test environment, must be established in order to exactly mimic the production environment while testing new code or new functionality. Additionally, if the test environment must by accessed by multiple users, additional pre-test environment(s) would also need to be established. This allows for testing the basic integrity of new code deliveries or performing other administrative and environmental management tasks (or other more selective or disruptive instrument testing) without disturbing routine or ongoing testing. One example of an additional environment might be one maintained strictly for training. Another might be one maintained for certain disruptive tasks, such as testing that involves the Blaise scheduler and the appointment block, or for extensive recompiling of code.

Test Plan

Experience has shown testing to fall broadly into two major categories: core testing and “ad hoc” testing of specific issues or problems, typically arising from either routine testing or routine interviewer usage of the instrument. Core instrument specifications must be identified for core testing.

Core testing is both periodic and ongoing, as needed. It is typically done prior to deployment of new TFU instrument releases and also prior to any new upgrades or conversions to new Blaise software releases. It may also frequently be performed in the interim of either of the two situations above, based on implementation of new spec changes or new code deliveries, or based on problems that may have been detected. Examples of core testing for our TFU instrument include all types of essential specifications testing listed below:

1. Verification of data integrity and case disposition concerning data read into and out of DEP, and as it appears in the datafile, call history, audit trails, etc., per read in, read out, clerical edit and other specifications
2. Verification of pathing and universe constraints against the datamodel specification
3. Verification of correct outcomes, agendums per cumulative outcome, numberroute, and dial counter specifications
4. Verification of essential header text, question text, and both response categories and response text against questionnaire specifications
5. Verification of screen layout of infopanes, formpanes per question per the front/back specifications

“Ad hoc” item testing is ongoing and is based on any outstanding issue or problem that has been reported. A dated and detailed testing log is maintained to describe and track these problems as they arise and will also be described in more detail below. The implementation of both core and ad hoc testing procedures will be discussed below.

Implementation and Usage of Blaise Instrument Testing Plan

To maximize office resources and develop an efficient system for approving changes to Blaise code, various testing plans and procedures are employed for the Blaise TFU instrument. The testing plans are geared to cover a variety of instrument changes, including but not limited to: structure changes, data input routines, questionnaire content, and outcome code assignments.

During the course of ACS survey management, changes to the daily operation may require instrument specifications to be revised and updated. To update the Blaise TFU instrument, requests and revised specifications are submitted to in-house and/or contract developers for recode. Once the new code is received, it is incorporated in a recompiled Blaise TFU Data Entry Program for testing. Unfortunately, with multiple change requests for a particular block of code, any lapse in strict version control may result in reintroduction of outdated code into the instrument. While performing core testing or cursory checks, aberrations or unexpected outcomes are invariably identified. These aberrations, such as improper skip patterns, outcome codes, or incorrect subject or verb fill, are typically related to the most recent changes but also may have existed for an indeterminate amount of time before being identified. Consistent usage of regular testing approaches is the best way to identify and resolve aberrations.

When a new version or build of Blaise software is released, our organization decides whether or not to upgrade our current Blaise survey instruments based on evaluation of expected benefits to survey operations, and/or on recommendations from Westat (the North American Blaise technical representative) or our systems technology staff. At other times, significant changes to the TFU instrument structure are made specifically to elicit desired instrument behavior or resolve particular issues. These two scenarios, in addition to the circumstances described above, require varying degrees of testing. Detailed below are aspects of our testing plan that are employed, dependent upon the circumstance and level of instrument change.

Cyclical and Continuous Testing

Whenever an instrument specification is changed and Blaise code is subsequently modified to accommodate the new spec, a thorough testing of the instrument portion must be performed. Once the new code is recompiled, TFU instrument testers conduct several mock interviews in the data entry program (DEP) to check that the instrument was modified correctly (to specification), and that other areas of the instrument were not inadvertently affected as a result of the code change. For example, modifying a block of code without regard to its existing parallel code in a separate file would cause inconsistent instrument behavior. If the initial instrument problem was not specification-related (for example, a problem discovered while testing another modification or problem), the same type of testing is administered once the problem is identified.

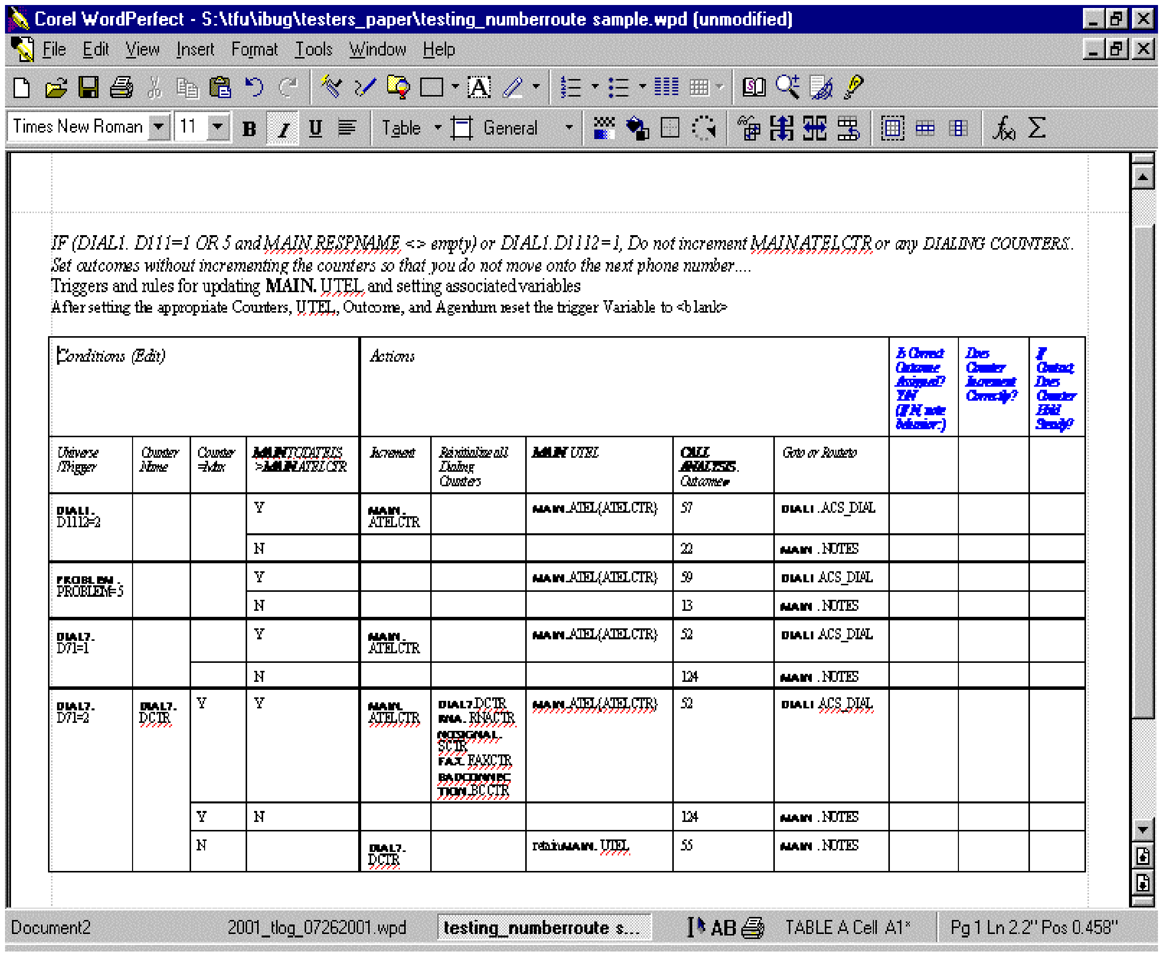
Once the instrument change has been successfully tested, that version is maintained in the separate test environment until it is deployed in the production environment (see Maintenance of Separate Environments, above).

Troubleshooting and Ad Hoc Testing

At times, we are unaware of instrument problems unless we are informed of TFU-related production problems by the staff in ACS TFU Unit of the Census Bureau’s National Processing Center in Jeffersonville, Indiana. Reliable lines of communication are fostered to facilitate this exchange of information.

Once we are given details about the instrument behavior, preferably with specific examples, testers at Headquarters then try to replicate the problem based on the information provided. This usually consists of specific keystrokes performed within particular types of cases to produce desired outcomes or pathing sequences (whether correct or incorrect). If a certain case is referenced where the problem was discovered, we have the ability to reference these particular cases directly for a given number of days within our Blaise production database archives. This further helps us to identify and emulate as much as the possible the environment(s) within which the problem was first recognized. If the problem is indeed instrument-based (which is the norm) and not interviewer-based, a request is made to our developers to rectify the code, where we would then in turn re-test to verify that the problem has been resolved. Interview-based issues may require adjustment to training procedures, which are handled elsewhere.

In similar respects, there also are problems that are discovered in-house that are handled in the same manner. Replication of errors and careful documentation is always performed before making requests of developers to alter code.


Corel WordPerfect - S:\tfu\ibug\testers_paper\testing_numberroute sample.wpd (unmodified)
File Edit View Insert Format Tools Window Help
Times New Roman 11 Table General
Document2 2001_tlog_07262001.wpd testing_numberroute s... TABLE A Cell A1* Pg 1 Ln 2.2" Pos 0.458"

IF (DIAL1. D111=1 OR 5 and MAIN.RESPNAME <> empty) or DIAL1.D1112=1, Do not increment MAIN.ATELCTR or any DIALING COUNTERS. Set outcomes without incrementing the counters so that you do not move onto the next phone number....
Triggers and rules for updating **MAIN.** UTEL and setting associated variables
After setting the appropriate Counters, UTEL, Outcome, and Agendum reset the trigger Variable to <blank>

Conditions (Edit)				*Actions*					***Is Correct Outcome Assigned? Y/N (If N, note behavior.)***	***Does Counter Increment Correctly?***	***If Contact, Does Counter Hold Steady?***
Universe /Trigger	*Counter Name*	*Counter =Max*	***MAIN.**TOTATELS >**MAIN.**ATELCTR*	*Increment*	*Reinitialize all Dialing Counters*	***MAIN.** UTEL*	***CALL ANALYSIS.** Outcome*	*Goto or Routeto*			
DIAL1. D1112=2			Y	MAIN. ATELCTR		MAIN.ATEL(ATELCTR)	57	DIAL1.ACS_DIAL			
			N				22	MAIN.NOTES			
PROBLEM. PROBLEM=5			Y			MAIN.ATEL(ATELCTR)	59	DIAL1.ACS_DIAL			
			N				13	MAIN.NOTES			
DIAL7. D71=1			Y	MAIN. ATELCTR		MAIN.ATEL(ATELCTR)	52	DIAL1.ACS_DIAL			
			N				124	MAIN.NOTES			
DIAL7. D71=2	DIAL7. DCTR	Y	Y	MAIN. ATELCTR	DIAL7.DCTR RNA. RNACTR NOSIGNAL. SCTR FAX. FAXCTR BADCONNEC TION.BCCTR	MAIN.ATEL(ATELCTR)	52	DIAL1.ACS_DIAL			
		Y	N				124	MAIN.NOTES			
		N		DIAL7. DCTR		retainMAIN. UTEL	55	MAIN.NOTES			

Figure 1. Sample Page of Numberroute Specifications Testing

Core Testing

Several circumstances require intensive, detailed testing procedures to ensure all aspects of the instrument are functioning correctly. Among these circumstances are: Blaise version upgrades, remote and on-site instrument deployments, and any instrument structure changes.

At a minimum, instrument upgrades involve the core testing of the five specification and data integrity verification activities outlined above in the core testing Test Plan. Refer to Figure 1 (above) for a sample testing document for Numberroute specifications. Additions to or expansion of current core testing areas may be incorporated as time and resources permit and circumstances require.

For version upgrades, a separate test environment is first upgraded with the new Blaise software version. Core testing is conducted by several testers, to divide responsibility (as some testers are more familiar with relative parts of the instrument), cross-check as needed, and minimize individual testing burden.

Other data integrity tests are conducted on, for example, Blaise input file and Manipula routines by computer specialists, to make sure that all data are being read in properly. Checklists and testing tables based on core testing specifications are developed to insure that testing procedures are completely thorough. Also, core testing is conducted whenever plans are made to deploy an updated TFU instrument. This helps to ensure that what is incorporated into the production environment is fundamentally sound.

An abbreviated version of the core testing procedures is performed for situations such as: limited instrument enhancements (for example, updates to the appointment block, or to CEDIT case disposition algorithms), and code modifications that may affect other blocks in instrument (for example, cumulative outcome table updates and revisions to core instrument [acsform.bla] code). Depending upon the degree of instrument change, the abbreviated core testing may range from four core activities to as few as two. When it is determined that an exhaustive test is not necessary, the core testing is refined to focus on only the essential areas of concern.

Testing Log

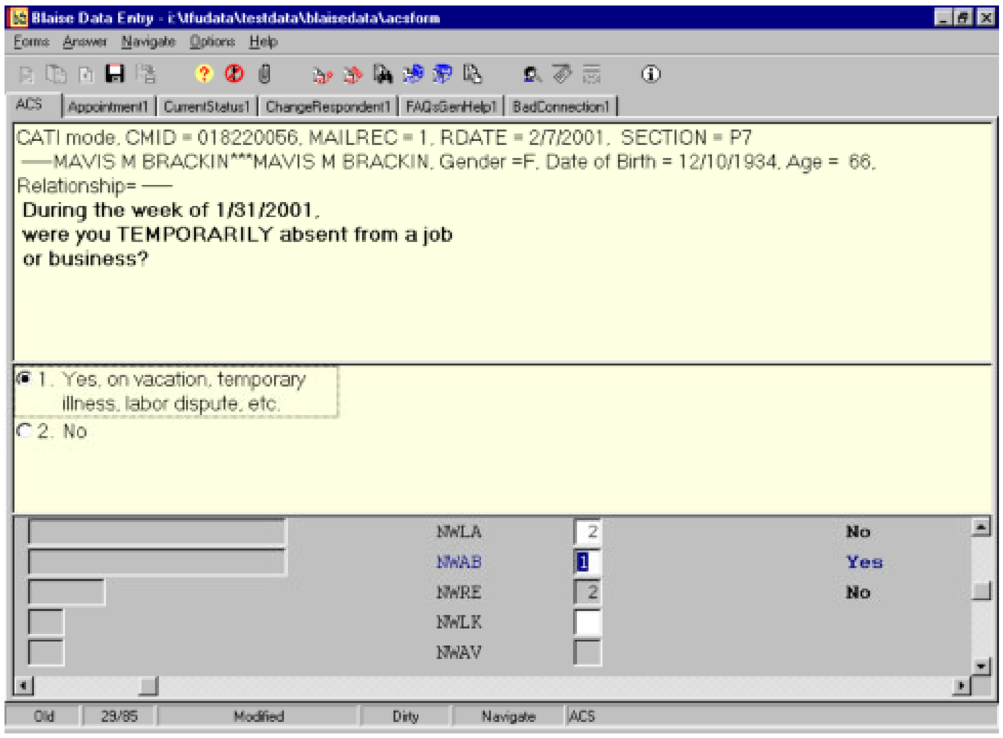
To maintain an efficient method of organizing each testing occurrence, we developed a testing log (Tlog) which we maintain and circulate to everyone involved with the TFU operational maintenance. The Tlog is essentially a running historical table that documents the following information: when an instrument problem is first discovered or reported to us, a detailed description of the problem, its correction status, to whom it is assigned, and whether or not the problem has been fixed.

Planned Enhancements to Testing Procedures

Although our testing plan is relatively standardized, it still lacks systematic methods for testing certain code fixes. As mentioned earlier, our experiences testing the Blaise TFU instrument have underscored the fact that code modifications may inadvertently affect other instrument functions. Central to planned enhancements to current testing procedures is the development of a TFU Testing Checklist that testers can reference and utilize in every testing scenario. Certain code fixes (for example, changing an error flag value) will be assigned specific instrument functions which are dependent upon the code fix (for example, checking active signals, error status blocks, and relevant LookSee fields). Instead of having to know or determine which instrument functions to test for a specific code fix, a tester will be able to refer to a complete listing of which aspects of the instrument are involved for all routine core testing situations. As the ACS operation expands in 2003, so will the number of cases that become eligible for TFU. A testing checklist will help conserve resources, minimize testing time, and help to ensure that the proper functions are being covered in the testing.

Testing Log

To maintain an efficient method of organizing each testing occurrence, we developed a testing log (Tlog) which we circulate to everyone involved with the TFU operational maintenance. As mentioned, the Tlog is essentially a running historical table that documents the following information: when an instrument problem is discovered (or reported to us), a detailed description of the problem, its correction status, to whom it is assigned, and whether or not the problem has been fixed.


Blaise Data Entry - i:\tfudata\testdata\blaisedata\acsform
Forms Answer Navigate Options Help
ACS Appointment1 CurrentStatus1 ChangeRespondent1 FAQsGenHelp1 BadConnection1

CATI mode, CMID = 018220056, MAILREC = 1, RDATE = 2/7/2001, SECTION = P7
—MAVIS M BRACKIN***MAVIS M BRACKIN, Gender =F, Date of Birth = 12/10/1934, Age = 66,
Relationship= —
During the week of 1/31/2001,
were you TEMPORARILY absent from a job
or business?

● 1. Yes, on vacation, temporary illness, labor dispute, etc.
○ 2. No

	NWLA	2	No
	NWAB	1	Yes
	NWRE	2	No
	NWLK		
	NWAV		

Old | 29/85 | Modified | Dirty | Navigate | ACS

Ex.1

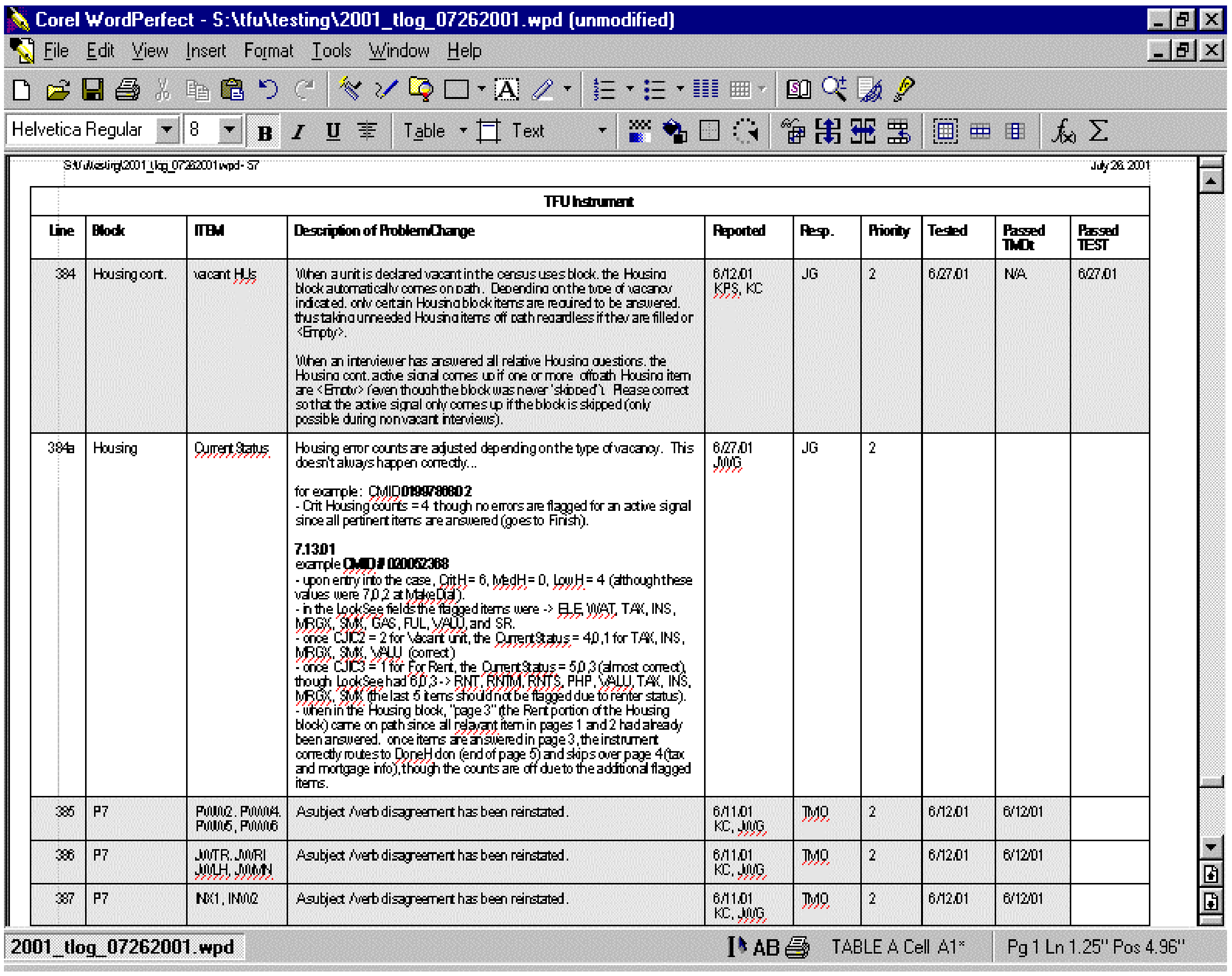
When first entering a 1 in NWLA, then a 2 in NWRE - this correctly brings NWLK on path. I then back up and change NWLA to 2, which correctly brings NWAB on path. When NWAB = 1 the instrument should go to WKL. Instead it goes to NWLK which remained on path because of the NWRE 'keep'= 2.

Figure 2. Sample Diagnostic Graphic for Troubleshooting

Each instrument problem is given a unique sequence number to easily refer to the item. If a fix of the item is attempted but not sufficient, the testing result is dated and documented for that item.

This document serves as an excellent reference for all concerned parties and is easily disseminated upon request. Newly added Tlog items are given a priority number of 1, 2, or 3 (with 1 being the highest priority) to help the developers prioritize their code modifications. For instance, an item that is not detrimental to the success of a TFU interview, such as certain infopane typographical errors, could be given a priority 2 or 3, whereas an item that details an improper skip pattern would be given a 1. It is also possible to give higher priority to otherwise less important items if an instrument deployment is planned and all relevant Tlog items must be completed or fixed by a certain date.

At times, the statement of an instrument problem in the Tlog may not be clear enough for a developer to understand. Often testers and developers discuss a problem or may even meet so that the tester can demonstrate the problem or provide supplemental information to further clarify the problem. Figure 2, above, is a sample diagnostic attachment from a mail message sent from a tester to a developer.


Corel WordPerfect - S:\tfu\testing\2001_tlog_07262001.wpd (unmodified)
File Edit View Insert Format Tools Window Help
Helvetica Regular 8
Table Text

S:\tfu\testing\2001_tlog_07262001.wpd - 57 July 26, 2001

TFU Instrument

Line	Block	ITEM	Description of Problem/Change	Reported	Resp.	Priority	Tested	Passed TMDt	Passed TEST
384	Housing cont.	vacant HUs	When a unit is declared vacant in the census uses block, the Housing block automatically comes on path. Depending on the type of vacancy indicated, only certain Housing block items are required to be answered, thus taking unneeded Housing items off path regardless if they are filled or <Empty>. When an interviewer has answered all relative Housing questions, the Housing cont. active signal comes up if one or more offpath Housing item are <Empty> (even though the block was never 'skipped'). Please correct so that the active signal only comes up if the block is skipped (only possible during nonvacant interviews).	6/12/01 KPS, KC	JG	2	6/27/01	N/A	6/27/01
384a	Housing	Current Status	Housing error counts are adjusted depending on the type of vacancy. This doesn't always happen correctly... for example: CMID**0189786802** - Crit Housing counts = 4 though no errors are flagged for an active signal since all pertinent items are answered (goes to Finish). **7.13.01** example **CMID# 020052368** - upon entry into the case, CritH = 6, MedH = 0, LowH = 4 (although these values were 7,0,2 at MakeDial). - in the LookSee fields the flagged items were -> ELE, WAT, TAX, INS, MRGX, SMK, GAS, FUL, VALU, and SR. - once CJIC2 = 2 for Vacant unit, the CurrentStatus = 4,0,1 for TAX, INS, MRGX, SMK, VALU (correct) - once CJIC3 = 1 for For Rent, the CurrentStatus = 5,0,3 (almost correct), though LookSee had 6,0,3 -> RNT, RNTM, RNTS, PHP, VALU, TAX, INS, MRGX, SMK (the last 5 items should not be flagged due to renter status). - when in the Housing block, "page 3" (the Rent portion of the Housing block) came on path since all relavant item in pages 1 and 2 had already been answered. once items are answered in page 3, the instrument correctly routes to DoneHdon (end of page 5) and skips over page 4 (tax and mortgage info), though the counts are off due to the additional flagged items.	6/27/01 JWG	JG	2			
385	P7	PWW2, PWW4, PWW5, PWW6	A subject /verb disagreement has been reinstated.	6/11/01 KC, JWG	TMO	2	6/12/01	6/12/01	
386	P7	JWTR, JWRI JWLH, JWMN	A subject /verb disagreement has been reinstated.	6/11/01 KC, JWG	TMO	2	6/12/01	6/12/01	
387	P7	INX1, INW2	A subject /verb disagreement has been reinstated.	6/11/01 KC, JWG	TMO	2	6/12/01	6/12/01	

2001_tlog_07262001.wpd TABLE A Cell A1* Pg 1 Ln 1.25" Pos 4.96"

Figure 3. Sample Page from Tester's Log (Tlog)

The Tlog has been invaluable in helping us document testing progress. It also serves as an archival reference for identifying new or recurring problems which may be similar to past instrument problems. Above is a sample page from our Tlog (Figure 3).

Version Control

Standard version control procedures involve nested folder locations, naming conventions, date-time stamping, and archiving of source code and data, regardless of whether manually done or whether versioning library software is used. This allows for ease of maintenance; code delivery tracking; comparison and troubleshooting of code, data, and instrument behavior; as well as for version rollback, should that be necessary.

In order to trace any versioning issues with current and newly incorporated code, we have found it necessary to maintain an orderly and structured network storage of code delivery and implementation. This has also allowed us to maintain a continuously updated historical archive of code. Additionally, analysis of dated code archives in conjunction with dated Tlogs has allowed us to evaluate past issues and solutions that were or were not utilized successfully for those issues with current issues that may be identical or similar. This has proved invaluable on a number of occasions, particularly as the issues and problems have become more complex.

Deployment

There are two major types of deployments: instrument upgrades and software upgrades. Ideally these two types of deployment are done separately in order to avoid conflating sources of possible problems or outcomes on the production instrument.

All instrument upgrades require regular core testing, as mentioned earlier. For Blaise software upgrades, in addition to those tests described earlier, this core testing is typically augmented by a variety of other system-level tests prior to deployment. This combination of internal and external testing is necessary to assess fitness in the current production environment before implementing wholesale conversion to the next software upgrade. Existing network and hardware constraints, combined with the instrument paradigm (that is, the logic and organization of the production instrument in question), may be critical to assessing real-time production performance. This may involve performance of transparent remote (for example, via PCAnywhere) or actual on-site testing of the physical production environment as a final step. The type and extent of external system testing of the instrument functionality is determined by the particular network and server configuration and will involve coordination with the systems technology staff.

Including thorough viability testing into the “scheduled time to next upgrade” is critical to the seamless functionality of your production instrument. Also, it is typically desirable to minimize upgrades (that is, a few times yearly), allowing the maximum amount of time on the front end to assess upgrades and instrument integrity in a baselined test environment before installing in the production environment.

Diagnosis and Response

Optimization testing during development and modification of a Blaise instrument will often reveal pressing issues, or specification oversights or nonconformities that must be resolved to maintain minimum required functionality. Optimization testing, both core and ad hoc, are typically also one reliable source of diagnosing inefficiencies or inconsistencies that do not threaten performance but should be corrected to improve or maximize functionality. We have implemented both a tracking log and a set of core testing procedures, as well as a structured yet flexible system of testing within the appropriate environment using appropriate data and measuring tools, for speedy resolution of both straightforward and thorny issues.

Summary

Testing is critical to the functionality of a production TFU Blaise instrument. Core testing procedures based on rigorously maintained specifications, along with coordinated versioning and archiving, are critical for baselining instrument functionality. Additionally, procedures must also be developed to accurately track, diagnose, test, and resolve real-time issues and spontaneous problems in a timely manner. Regular implementation of both core and problem-oriented testing results in better conformity to specifications. It also allows for preemptive treatment of problems before they reach production and also for enhancement of instrument functionality through diagnosis of inefficiencies.

Acknowledgements

Amanda Foster Sardenberg is a computer specialist who also performs processing maintenance and production instrument management for the ACS TFU project.

John W. Gloster is a survey statistician who also maintains and updates specifications and procedures for production instrument management on the ACS TFU project.