Challenges of Instrument Development

William E. Dyer, Jr., U.S. Census Bureau
Ellen Soper, U.S. Census Bureau

Bureau of the Census
4700 Silver Hill Rd Room 3640
Suitland, Maryland 20746

The Census Bureau's Technologies Management Office (TMO) Authoring Staff consists of a centralized team of programmers (a.k.a. authors) who develop Computer Assisted Interviewing (CAI) instruments using two different authoring languages, i.e., Blaise for Windows and CASES written for DOS. The Authoring Staff has written CASES instruments for years, but has only recently been developing instruments for Blaise. As more and more instruments are written for Blaise, new tools and standardized approaches are desired to help streamline project management, instrument development, testing and deployment.

The Bureau of Census (BOC) is a large organization that historically fields large complex surveys. The automation of large complex surveys introduces additional challenges to the process of data collection. The purpose of this paper is to identify some of the aspects of authoring that we perceive as being challenging and to review the approaches we have taken to deal with these issues. This paper will focus specifically on the challenges associated with project management, instrument development, instrument testing, and training of personnel.

PROJECT MANAGEMENT

Challenges

There are several challenges associated with managing a survey automation project. From a software development point of view, it is important to recognize the need to improve proper project management procedures to ensure all the tasks associated with programming culminate in a product that meets requirements within the established time frame and within budget. The project manager's role of getting a good understanding of the software requirements is a difficult challenge. More challenging is dealing with requirements that continually change throughout the lifecycle of a project. Effectively controlling change so that the project remains on schedule is an important issue and must be dealt with. The ability to establish and maintain good customer relations is also an important issue that should not be overlooked. After all, customer satisfaction is a key factor to success. How does one manage a project to a successful conclusion while at the same time ensuring the software developers receive what they need to perform technically, within schedule, and in concert with one another?

Approaches

How do we get good requirements?
A typical survey automation project will have internal (BOC) subject matter specialists who serve as the overall program manager, otherwise known as the sponsor. One of the sponsor's roles is to gather requirements from the client, such as Bureau of Labor Statistics or Bureau of Justice Statistics or another outside agency. The sponsor will then apply their survey automation experience to develop program or instrument specifications. The program developers work from these specifications to develop code that meets the requirements. It is important that the programmers and sponsors work very closely to identify

the requirements and any ambiguities in the specifications. It is critical to establish and maintain a common understanding of the client's requirements with the client and all other project members.

We have learned that specifications that were written with one software language in mind, e.g. CASES, should not be used as specifications for another software language, e.g., Blaise 4 Windows. The software language architectures are different which could result in limiting the instrument functionality. For example, designing tables that have arrays of arrays may be conceptually correct but impractical to implement. Some would argue that the ideal specification is developed such that it is not dependent on any software language. However, we have found that the best specification is one that is developed with programmer involvement that includes structures similar to the software language that will be used for development. We recommend the specification include information about the desired block structure, edit conditions such as checks and signals, and detailed logic statements within the rules sections, to name a few Blaise specific software attributes. A software language specific specification helps with the programming effort and improves the software development process.

How do we manage a project for success while keeping the customer satisfied?
Once the requirements are clear, it is important to create and disseminate a software development plan. The software development plan serves to educate the sponsors on the issues related to survey automation. In our experience, the more knowledgeable the sponsors are with regard to survey automation issues the better the project runs. The software development plan serves as a planning and communication tool that can be used to document and manage changes to the plan. A well-documented plan defines expectations for all project members and helps to establish the timeline. The plan should also include how communication will be handled. It is important to identify roles and responsibilities and determine who will have authority over what aspect of the project. When clear goals are established and communicated there is a probability the objectives will be achieved on time and within budget.

How do we deal with change control?
It is often the case that more than one person is assigned the task of developing instrument code. The coordination of programming activities can be a challenge without proper configuration management techniques. At a minimum, requirements, specifications, and program code need to be managed in order to maintain their integrity and traceability. The sponsor and the Authoring staff share in this responsibility. Several survey sponsors have developed specification databases to store metadata about the survey questions that will be used for data collection. The sponsor will populate the database with information that is useful for their own purposes such as edits used in post processing of the data as well as the attributes needed to provide an instrument specification. Changes to the instrument specification are maintained in this database and new specifications can be output at the press of a button.

Additionally, change requests are stored in a centralized database system that both sponsors and authors access. This system is referred as the Change Request Tracking System. This system is used once the instrument enters into the testing phase of software development. Changes that are identified as a result of testing whether it is a new requirement or a problem identified in the instrument and are entered into the Change Request Tracking System. The time, version, nature of the problem and priority are all assigned. The software developers access the system to review the changes and to update the resolution of the change. The sponsors access the system to review the status of the change, enter new changes, or to update the existing requests. This facilitates making changes efficiently and ensures accountability on both sides. The change control processes should include a feedback loop when a request cannot be accommodated, is not clearly understood or cannot be addressed within the requested time period. Under these circumstances an author can make notes or provide explanations directly into the Change Request Tracking System. The change request database stores requests for modifications to the instrument that can be reviewed by the project manager and distributed for action. Time and date stamps are used to verify that the sponsors needs and requests are being attended to and that each project member is aware of

their responsibility. Deadlines for submitting changes can be documented, incorporated into the project plan, scheduled and measured. The Change Request Tracking System has allowed us to do research into the number and types of changes being requested and applied so we can identify opportunities for improvement. Having a way to document requests, prioritize and disseminate them to authors turned out to be a real time saver. This system also serves as an excellent reference for future or past projects.

Another method of change control is the use of software to archive different versions of the source code. Program files are stored in a version control application. Through proper use of the file check out/check in concept, program changes are captured, maintained, and problems with sharing program code are minimized. These files as well as shared source code can be placed under version control and locked by the person who has responsibility for insuring their integrity. Establishing ownership of files is necessary when multiple programmers are working on a project.

DEVELOPMENT

Challenges

A challenge in developing instruments for multiple surveys is ensuring programming efforts are coordinated and applied consistently across the board. It is difficult to foster a common understanding of the development tasks that occur across surveys for all team members. It is easy to develop the same functionality different ways when you have different programmers and specification writers working on the same and/or different projects. By having a common understanding of how requirements have been met, we can apply past practices to achieve continuity across instruments. Managing program files is also a challenge that affects a team of software developers who work on multiple concurrent projects. It is important to develop and organize a system that will work for all projects and programmers. The Census Bureau develops surveys, which are heavily customized to meet sponsor requirements. Sometimes we would like to modify the software default behavior requiring manual intervention by a programmer. Instrument preparation and release becomes a special challenge with the large volume of files that are required. It is essential that instrument builds are performed correctly, are consistent between builds and are completed according to established schedules.

Approaches

How is a common understanding of development tasks achieved?
We have taken several approaches that attempt to provide a universal understanding of development tasks. The Authoring Staff has approached the development of multiple Blaise instruments by organizing a team of programmers who will initially design and develop the instrument. At some point, the code is handed off to another programmer to maintain or enhance. This arrangement allows the same programmers to develop the more complex programming tasks consistently from one project to another providing continuity across surveys. The project manager is part of this team of initial developers. The project manager manages activities associated with developing complex instruments by working with software developers to create standards, libraries and procedures that affect similar tasks across one or more projects. It becomes easier to identify potential areas to standardize when the same resources are performing the tasks.

Incorporating new Blaise programming standards and procedures into our existing infrastructure improves future instrument development efforts. All programmers share information with one another in regular weekly meetings organized to provide updates on the project status. These meetings are an opportunity for software developers to share techniques or to discuss issues that relate to meeting the software requirements. Many opportunities have arisen that have led to development of code that can be reused. For example, we have developed external databases that hold fill data but are called using a

simple procedure. The same procedure is used throughout the instrument minimizing coding. Using lookup tables rather than hard coding fills allows us to change the external file without having to prepare the instrument.

To facilitate handing code from the initial developer to another programmer for maintenance it becomes important to create libraries of shareable code. A centralized library allows groups of programmers to coordinate activities and further simplifies the development process. We have found that a project can share many files without conflict as long as a single person with an assigned back up contact maintains the library. One of our more successful techniques to share information was to publish a "tips and pointers" document in the centralized library. This library also includes documents such as programming standards that include generic naming conventions such as those used to define Blaise types and block structures. This simple method for standardizing type and block names helps the programming staff to develop modules of code that could be worked by any member of the staff.

How are files managed?
Blaise 4 Windows offers many new features not found in DOS-based systems. As a result, it is often the case that there will be more files that need to be managed. Standardization of the development environment, i.e., directory structure, is one way to manage files and ultimately, improve the process of developing instruments. We have consulted with both Westat and Statistics Canada about how they have implemented the directory structure for Blaise instrument development and field operations. In combination with recommendations from several areas, we have a "recommended" directory structure that can be used for our instrument development from cradle to grave. It seems that there is no one perfect way to organize files. If the single directory approach is taken, large instruments become unmanageable. If a tiered directory structure is used, the program files are better organized but version control issues are more pronounced. Since some files are used strictly for development and others are used at runtime, we have attempted to organize somewhat by the nature of the file type being used. Source code modules, libraries, type definitions and procedures are subdivided into functional directories. In addition, files for bitmaps and lookup tables are put into a directory called 'externals' that can be set as read only at the directory level to enjoy better performance and simplify code management. We have split our efforts into a development area and a build area to keep from conflicting with each other's work and further layered the build to support multiple versions of the release over time.

How are instruments prepared and released?
Building instruments in Blaise, even with the Blaise control center software tool, is a complex process with a lot of pieces that need to be arranged to make all the parts work together as an integrated instrument. There are advantages in having build and release procedures to control the process of preparing instruments. The key to success when preparing an instrument for release is to remember all the steps, the order that they need to be performed and all the details involved with setting the ModeLib, key bindings, help files, Manipula programs, data model properties, etc. It is unrealistic to expect one person to be solely responsible for this process, so we captured the steps required and trained others in the ways of building instruments in Blaise. This was a major departure from what authors have been expected to do for other languages. To this day we are having ongoing discussions about the best ways to implement the build procedure. We have found there are some manual steps required in the preparation process of Blaise instruments that cause unexpected results if not properly performed. Each time an instrument data model is renamed, which is a Census Bureau requirement for each release to production, many configuration items need to be manually reset from within the Blaise Control Center. For example, accept the hidden setting for enumerated fields is a manual step that if not selected will cause enumerated values to scroll off the displayed area. It is our desire to minimize these types of manual changes.

Repeatable results can be achieved by using the same procedures each time the instrument is prepared and released. Since there are so many things to remember when performing an instrument build, we have

made an effort to automate portions of the process. Having the same person do the build every time is impractical so we prefer the team approach of performing tasks. Making check lists helps, but falls short of ensuring the repeatable results we are seeking. Our effort starts with having an experienced programmer, who is familiar with Blaise requirements and the project constraints, mentor other staff in the procedure required to build an integrated instrument. As the steps become well defined we use a combination of written release procedures and automated procedures whenever possible. This benefits us in many ways; we train programmers, provide better documentation and improve the automated release procedures that can be adapted to suit future projects.

As one can see we use a variety of methods and procedures, ways to share code and engineer for code reuse to get the most out of our development efforts. By adopting standards whenever possible for everything from naming conventions to screen design we can have a number of authors working simultaneously on a very large survey. Having both written and automated build procedures allows us to cross train our staff and repeatedly produce quality surveys.

TESTING

Challenges

A centralized programming staff faces the challenge of ensuring instruments are thoroughly tested in an efficient and timely manner. Testing of instruments is a shared responsibility between the programming staff and the sponsors who develop the specifications/requirements. Meeting sponsor expectations of implementing changes quickly is a difficult but necessary task. It is important to develop a strategy of how an instrument will be modified, released and tested. Since we develop multiple instruments with multiple iterations of testing, it becomes necessary to consider the time it takes to make instruments available to the sponsor so they can conduct their testing in a timely fashion. The way the instrument is prepared and packaged, how it is delivered, and on what platform it is run are factors to consider. The TMO Authoring Staff has developed a system that allows sponsors to test the instruments; however, maintaining and supporting this centralized test system for users on numerous platforms, at remote locations, and for multiple software languages introduces additional challenges. For a system that was originally developed to aid software developers in the testing and debugging of instruments, maintaining and supporting this application across many sponsors is a challenge. The primary responsibility of the software developer is to develop the survey instrument. Supporting and maintaining an application that meets everyone's requirements can be time consuming. However, with that said, the centralized test system has reduced the time required to get the instrument to the sponsor or tester.

Approaches

What is our strategy or life cycle model of software development?
Perhaps the most difficult challenge we face is the issue of testing. Often overlooked and misunderstood, testing should be given more time on the schedules, should be better organized during all phases of the project and could become a focal point for improving our processes. Finding and correcting errors early is paramount if an organization wants to have instruments delivered on time and error free.

There are various life cycle models, which integrate well into the paradigm of software development of instruments. Most project managers and programmers are familiar with the classic Waterfall Model, the Iterative Model, and Incremental Model as diagrammed in Whitten, Neal, 1995. Managing Software Development Projects. The typical software development project at the Census Bureau is a combination of these models. Let's briefly review each software development model and discuss how our software development of instruments affects our approach to testing.

Waterfall Model

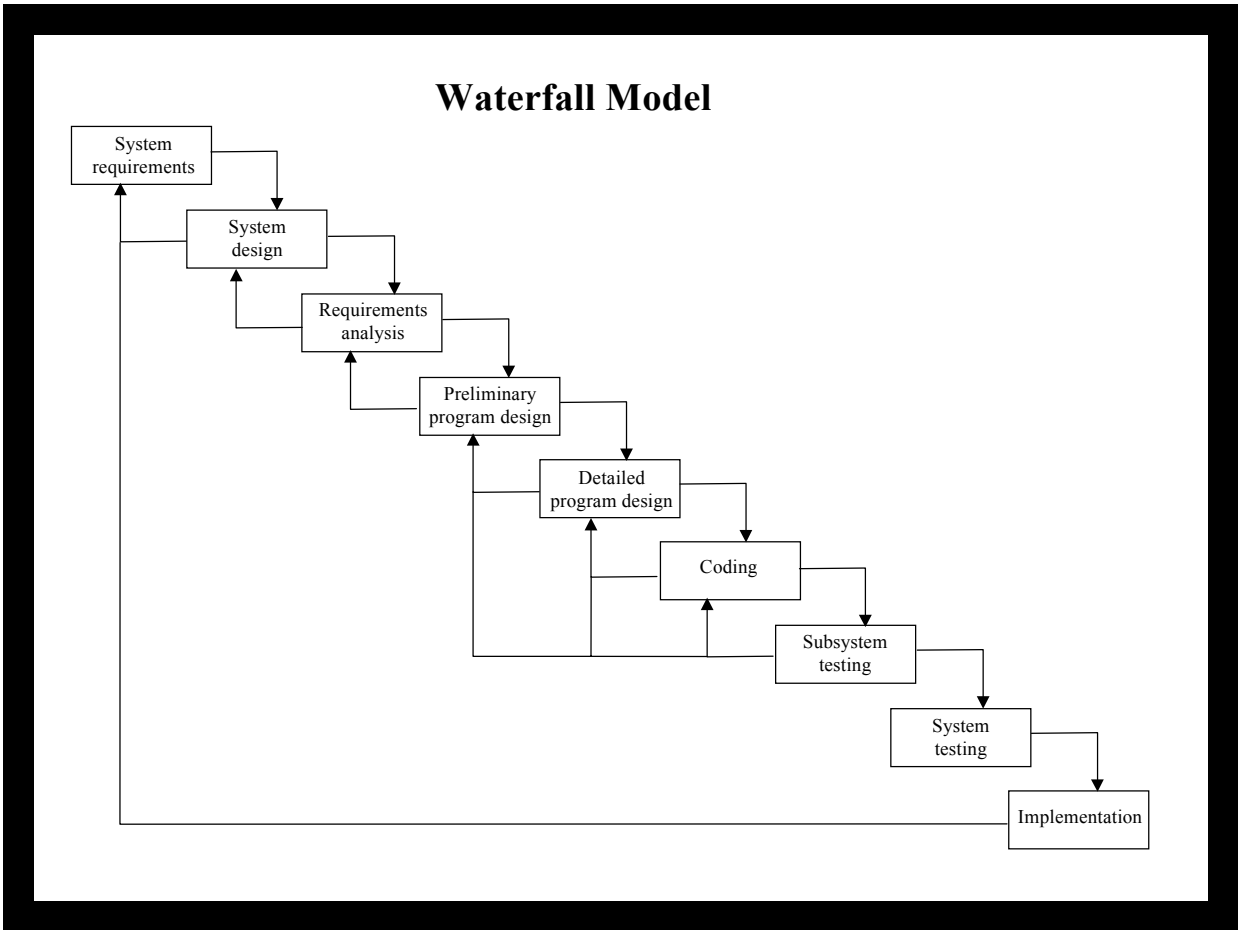

Figure 1

The classic waterfall model of the software development life cycle is very methodical. Typically, requirements are frozen once they are initially agreed upon and changes to the requirements are minimal. Also, testing is done later in the process. This model is especially good for software development when the requirements are well defined and well known up front. See Figure 1.

Iterative Model

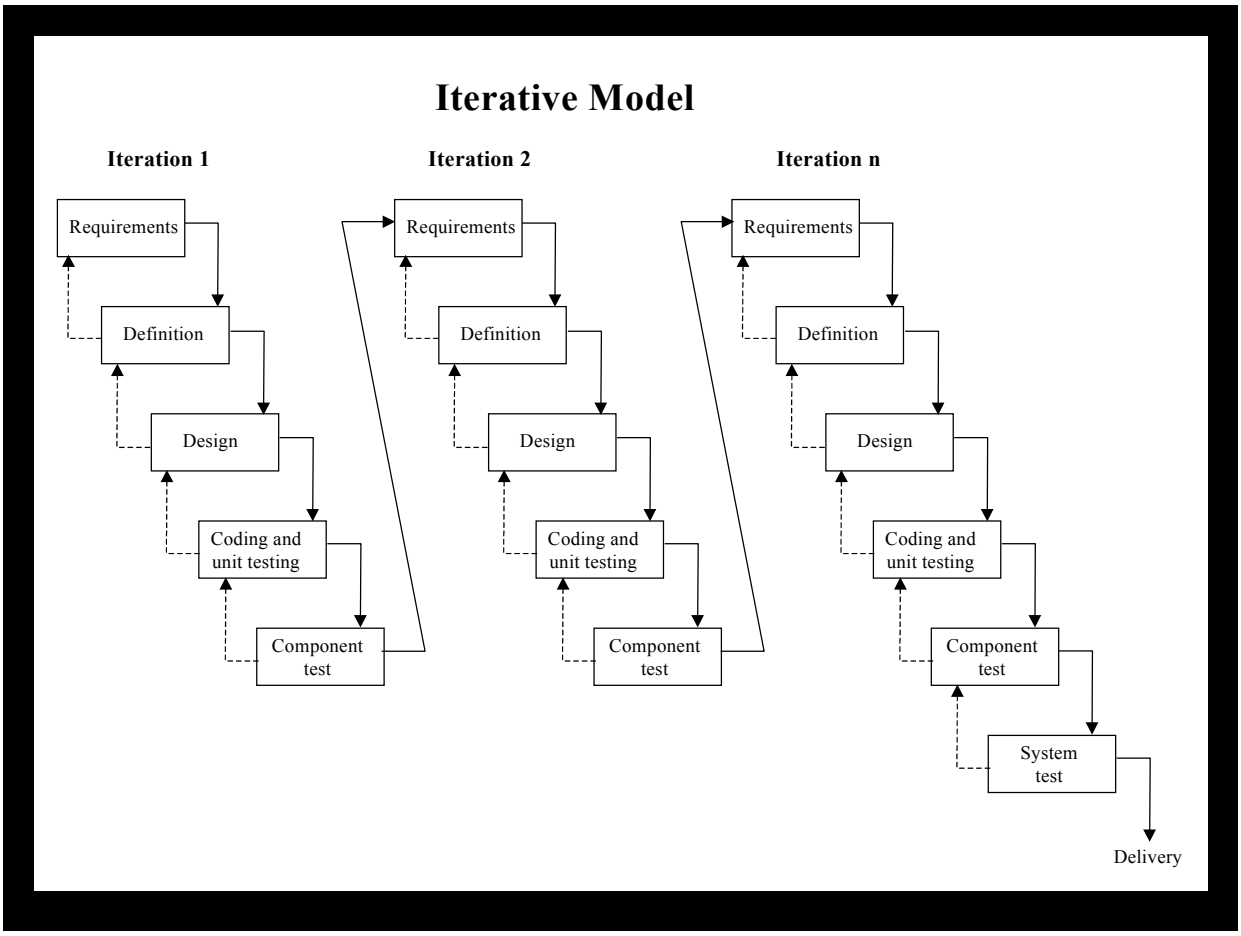

Figure 2

The iterative model of the software development life cycle allows for multiple iterations of the classic waterfall model. Each time an iteration is completed, this model allows for a user test of a baseline product (i.e., Acomponent test@ on the diagram). The comments that result from the user test are added to the existing requirements to develop the new requirements for the next iteration. This model is especially good for software development when the requirements are not well defined up front because it allows for the discovery of new requirements as you go through the development cycle. See Figure 2.

Historically, this was the software development life cycle used for instrument development. Instrument components were developed in a linear fashion and added as the next software baseline. Components were not often developed concurrent to one another because it was difficult to isolate the code once the components were integrated. We still use this model for most of our DOS based instrument development efforts.

Incremental Model

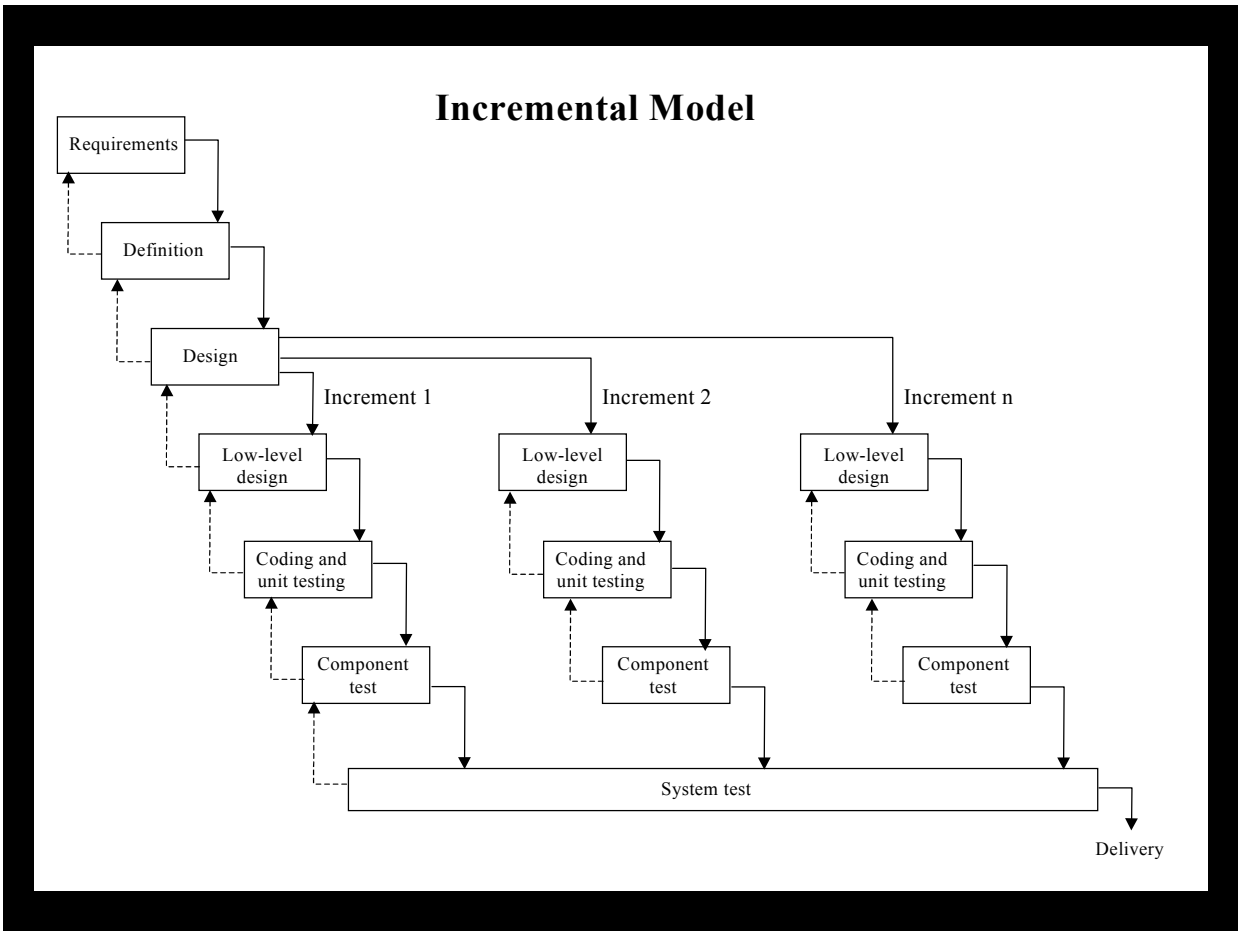

Figure 3

The incremental model of the software development life cycle is essentially a compressed version of the classic waterfall model. Typically, requirements are frozen once they are initially agreed upon and changes to the requirements are minimal. Once an overall design is completed, the development effort is divided into modules or increments so that lower level design, development, and testing can be done concurrently on the modules. The modules are then brought together at the end of the development cycle to be integrated and system tested prior to delivery. This model may be especially good for applications where there are natural divisions for coding purposes (e.g., client server applications). See Figure 3.

As we began developing with the Blaise 4 Windows software, we learned that the software has certain organizational flexibility we could use. Specifically, the Blaise 4 Windows software allows the code to be divided into components or modules that could be easily integrated together and removed apart again for further refinement. However, this life cycle model doesn't demonstrate the entire picture of what we do, so we invented a new model as presented in Figure 4.

Combination Incremental/Iterative Model

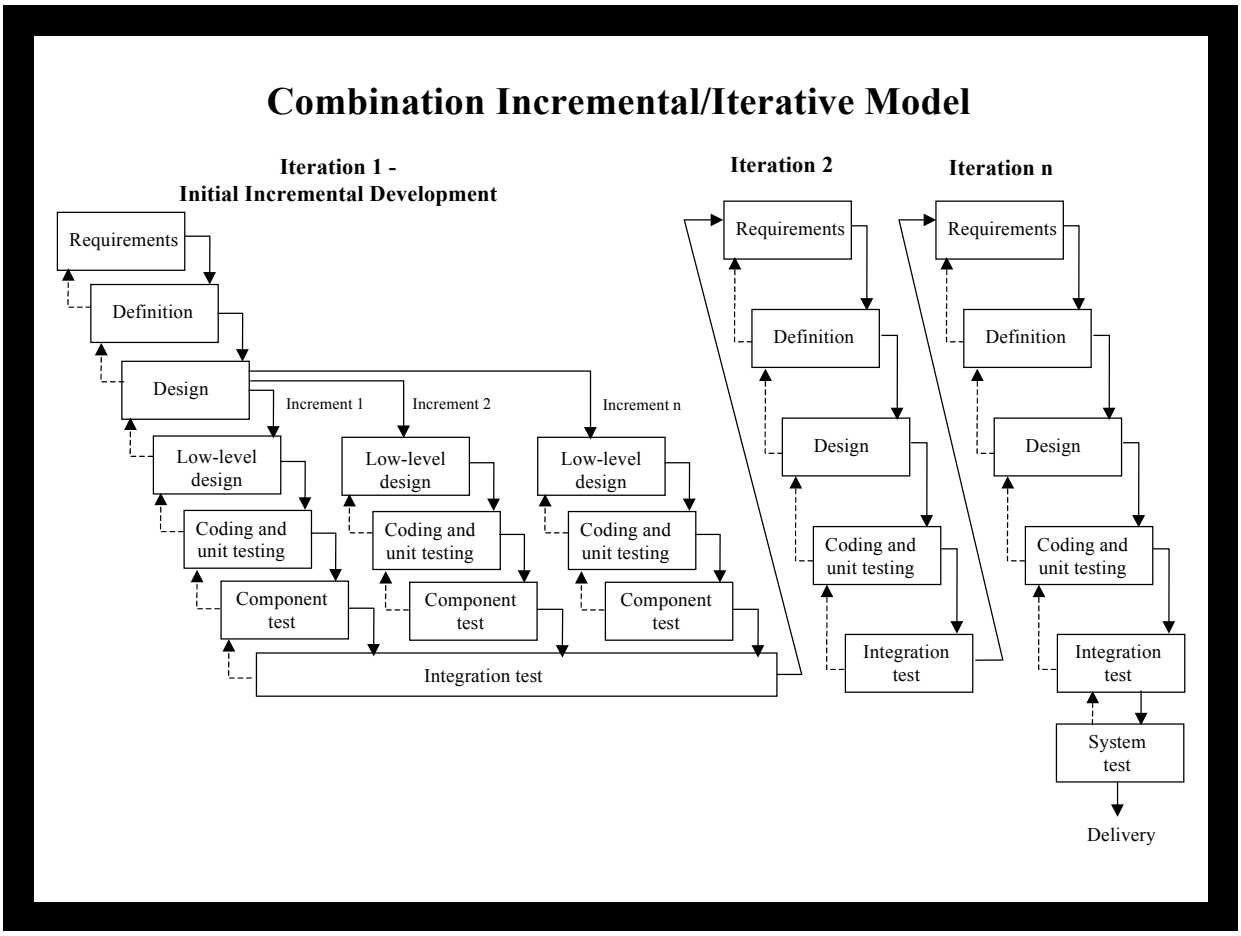

Figure 4

This combination of the incremental and iterative models of the software development life cycle allows for an initial development cycle that is incremental or partitioned into modules for development followed by multiple iterations of the classic waterfall model. Requirements are frozen once they are initially agreed upon. Once an overall design is completed, the development effort is divided into modules or increments so that lower level design, development, and testing can be done concurrently on the modules. The modules are then brought together at the end of the initial development effort to be integrated. Each time an iteration is completed, this model allows for a user test of a baseline product (i.e., Aintegration test@ on the diagram). The comments that result from the user test are added to the existing requirements to develop the new requirements for the next iteration.

This model allows for an initial modular and thereby, rapid approach to development followed by planned and controlled modifications to the system. This model is especially good for software development when the requirements are not completely defined up front because it allows for the discovery of new requirements as you go through the development cycle.

The combination software development model demonstrates clearly the work that is needed to adequately test instruments. Instrument testing is a very large and important task often not considered in developing project schedules. The TMO Authoring staff now includes this life cycle model in software development plans as a mechanism to help others understand the development and testing process.

How are instruments made available to the sponsors?
Software applications that are used by others, especially ones that have a graphical user interface, must be designed with the end user in mind. They should be intuitive and easy to use, be beneficial, and be designed so that they can be supported. To support the development, test and release of multiple instruments, the TMO Authoring Staff developed our own 'Tester's Menu'. The Tester's Menu is a system that runs on multiple platforms, i.e., Windows 2000, NT and 95, and allows users to test instruments developed in CASES and Blaise.

The primary responsibility of the TMO Authoring Staff is to develop instruments. The support and maintenance of the Tester's Menu is secondary. It was important to design the Tester's Menu to meet the basic needs of software developers and instrument testers. It was also designed so that it is easy to deliver, install, support and maintain.

The application driver is installed via e-mail using a program written in WinBatch, "The Batch Language for Windows" from Wilson WindowWare. The application resides on the server, whereas, the client's desktop receives only one executable necessary to provide the connection to the server. To change the functionality of a button or other menu function, an executable on the server side can be swapped without having to reinstall the application on every user's PC throughout the Census Bureau. The Tester's Menu system is being used at headquarters, regional offices, and the National Processing Center in Jeffersonville. Currently, the system supports several hundred users.

In an effort to make this menu as flexible as possible the program attaches to resources only when needed and lets them go when the user exits the application. It does not demand specific network or configuration modifications by other divisions, which makes this application easy for users to install. For Windows NT computers we use the set environment variable to drive DOS applications without having to modify the config.sys or autoexec.bat files on the user's PC. In fact, this software is so easy to use it installs the start icon on the user's desktop in all cases including dual boot systems.

Another important benefit of the design of the Tester's Menu system is the ability to dynamically make changes to the menu pick lists. A flat ASCII file is input to a stock Blaise manipula program eliminating the need to recompile source code to update the Blaise menu pick lists. The menu pick lists, e.g., survey name, are made from reading the directory structure. Adding, hiding or removing a directory makes an instrument available for use without additional programmer intervention, support or maintenance.

We answered the challenge of testing numerous instruments by building a testing system that is available for everyone to use. It provides a common ground for all testers of all instruments. The driver was programmed to have error-trapping built-in with useful messages given to the user if resources are unavailable or menu selections are not present. By keeping it simple to maintain and making it easy to install we build confidence that the products we make are reliable and useable.

TRAINING

Challenges

Training is an ongoing challenge. Designer, programmer and interviewer alike need access to training for several purposes: initially in how to do their jobs, to stay current with new developments in industry, and may even require re-training due to advancements in technology and/or survey methodology.

Our office is involved with finding new and better ways to introduce subject matter specialists, sponsors, project managers, programmers and field interviewers to different types of software design and implementation. Part of our task is to educate the sponsors and our staffs about how design decisions and coding techniques may adversely impact development issues such as instrument performance. Translating the sponsor's instrument design into a machine version, which can be used by the interviewers, is both challenging and rewarding for the authors. Part of the challenge is enabling good communication to share knowledge, skills and abilities. Education is mutually beneficial to all parties involved.

Developing an interface that has a common look and feel for our field interviewers that will also work with different CAI software is an important way to minimize training. Interviewers need to collect data quickly and efficiently. The development and implementation of standards allows the interviewer to learn the instrument faster, gain confidence in using the instrument, and adjust to using other instruments more easily.

Approaches

How is training addressed from a software development point of view?
Training is important during all phases of the instrument development process. Teaching programmers how to use the Blaise Control Centre and other tools is essential to accomplish the task of coding and building instruments. Educating our sponsors and subject matter staff on new methods and techniques is a wise investment. We also benefit by making sure the methods used to collect data across numerous types of instruments are consistent.

Having a well-documented and standard approach to interviewing simplifies the training of field interviewers. Standards for screen design and function keys assignments gives us the opportunity to create instruments that look and feel the same for field interviewers regardless of which interview is conducted. Training is simplified when the interviewer knows what to expect from survey to survey. Our challenge is to develop good CATI/CAPI Standards for GUI instruments. A GUI Screen Standards group was formed to develop, prototype, and user test screen standards for the Consumer Expenditure Survey. This was the first CAPI survey to be implemented using a Windows-based software language, Blaise 4 Windows. Decisions about screen colors, fonts, layout size, and function key assignments were made based on current and past experiences, prototyping efforts, performing usability tests with interviewers, and collaborating with other agencies that use Blaise 4 Windows.

Predefined standards for screen design go a long way toward minimizing unpleasant movement when navigating the instrument. By identifying the common elements we want to use across all surveys we can train the interviewers what to expect. The use of pull down menus or the speed bar and inclusion of some or all of the functionality of hot keys can be decided, defined and documented in advance simplifying not only the instrument design process, but also the training required.

Function key definitions are very much like the standards for screen design but on a smaller scale. Each instrument should have similar definitions so that interviewers can expect to use the same keys for the same functions across surveys. Every effort was made to keep keys the same between our CASES

instruments and the Blaise instruments. Many of the functions such as “exit”, “notes”, “help” are standard across surveys and should have consistent key mappings across surveys. Optional function keys have been set aside for special functions that do not apply across all surveys giving maximum flexibility.

The Blaise windows menu (.bwm) file gives us the ability to bind keys to meet our needs. With the freedom to easily define hot keys comes responsibility. It would be unwise to have every programmer and every survey sponsor make up their own key definitions. We have taken the approach of identifying keys used by CASES and those used by Blaise and reconciling the two to support a generic approach for all surveys built and supported by the Census Bureau.

Training issues are understandably part of our industry. As surveys migrate from one authoring language to another, as new computer based interviewing methods are developed, and as new developers, programmers, managers and field interviewers are hired they will need training. To simplify this task as much as possible we establish effective communication, develop and implement training opportunities and work to continually develop and enhance standards for CAI.

SUMMARY

The Technologies Management Office Authoring Staff are dedicated computer professionals tasked with developing methods and procedures that are generic and can be applied across all surveys, which make our Computer Assisted Interviewing instruments of better quality and more cost efficient. Our office supports standardization to the extent it is helpful in achieving The U.S. Census Bureau’s mission, “To be the preeminent collector and provider of timely, relevant, and quality data about the people and economy of the United States. “

Our staff acts as a human interface and turns ideas and subject matter specifications into procedures and programs for use by Computer Assisted Interviewing technologies. Our work requires us to interact with a host of other people, organizations and systems. The successes we have achieved would not be possible without our clients, sponsors, system developers, testers, trainers, and field interviewers.

Windows based design is well suited to modern machine capabilities and for a variety of techniques and features for data collection such as multi lingual text, voice, graphical and video presentations. With this array of opportunity naturally comes complexity. Our job is to meet the challenges of developing these complex instruments while balancing the sponsor wants, with what the software language and computing resources can do.

The approaches we are using to surmount these challenges have been and will continue to be incorporated for use in future survey automation efforts. We continue to address the issues and implement procedures to streamline project management, standardize instrument development, facilitate testing and improve training. Our objective is to continually improve our processes, while producing instruments that meet our sponsors’ needs on time and within budget.