Developing Analytic Programming Capability to Empower the Survey Organization

William E. Winkler (U. S. Bureau of the Census) and Michael Hidiroglou (Statistics Canada)

1. Introduction

In today's environment of powerful personal computers and software, developing new specialized software can be much easier. Accessing files and creating routine tabulations that once took teams of individuals months to produce can often be produced by a very good researcher or programmer in a few weeks. New statistical methods such as Fellegi-Holt edit/imputation, Fellegi-Sunter record linkage, confidentiality, and statistical methods such as bootstrap, multiple imputation, Expectation-Maximization (EM), and Gibbs sampling require both detailed knowledge of theory and the development of new computational algorithms. In the advanced statistical situations, there is much greater need for programming skills that are consistent with the analyses and theory. If we consider those successful software projects at different statistical agencies where these methods have been applied, it is likely that there is a key common characteristic. The common characteristic for developing specialized software is the project team or individual who understands the theory, the required analyses, and can do the programming.

The development of new specialized software raises several questions that range from the choice of programming methods to the support of the code and users. This software can be split into two main groups: research and production. Software for research purposes is initially developed by small groups who are conducting the research. This type of software is very specializing. It may only be understood by those using it. It is often 'thrown away' once the required research has been carried out. Some research software can serve as a prototype in the sense that it can demonstrate that theoretical concepts can be applied in practice. Changing research software into production software may require improvements in algorithm speed, complete rewriting into a different programming language, the addition of user interfaces and control information (statistics) needed for production processing, and documentation (possibly extensive). For survey production software, several issues need to be addressed, including the choice of problems to be programmed, the choosers, the management, and the support (development and ongoing) for such software.

To help clarify the exposition, we define an *analyst-programmer* as an individual who has detailed knowledge of the methods (possibly theoretical) associated with an analysis and who can develop the computational algorithms and associated computer code. An analyst-programmer might be a methodologist such as an economist, demographer, or statistician who can program. Or the analyst-programmer might be a programmer or informatician who understands theory and can develop new methodological applications. A *systems analyst* is a programmer or informatician that is often concerned with the development of general production systems. An analytic programming team is a group that collectively has the skills of an analyst-programmer. For instance, one individual might have theoretical knowledge of an imputation methodology and basic ideas about some of the computer code. A second individual might have advanced knowledge of data structures, algorithms, and code-writing techniques. The two individuals are able to communicate many ideas by being able to understand computer code. When we refer to a *methodologist*, we will mean someone who has significant insight into methods and theory. The methodologist may or may not have deep understanding of computer algorithms and coding techniques.

In this paper, we focus primarily on the need for and the skills of the individuals who would do analytic programming. Our view is that we are concerned with a group of individuals who are assigned to a technical team. The team is charged with developing (researching) new methods, creating a prototype set of software, or delivering a production system. We assume that the management review has taken place, the project has been prioritized, and very good individuals have been assigned. We assume that very good management support external to the group is available and that excellent hardware is at the group's disposal. The main issue is the skills of the individuals in the group. Why is one group able to perform successfully and another not? What are the analytic and programming skills needed? How can a statistical organization create a group that can move good ideas into practice? What are the programming and analytic skills needed among the individuals who create the production system? If a production system has been created, what are the skills needed among the analyst-programmers that may have to maintain or modify the system?

In the second section of this paper, we give straightforward examples of how analyst-programmers can improve practice and two large analytic programming project undertaken at our agencies. The third section describes good institutional practices that can be used when deciding on what projects will be undertaken. We provide an environment for developing analytic programming capabilities in the fourth section. The final section consists of summarizing comments.

2. Background

This section provides straightforward examples of how analyst-programmers can improve practice, highlights two large analytic programming project undertaken at our agencies, and provides further examples of the need for analyst-programmers. The straightforward examples suggest how easily moderate skills can be developed in better computer environments.

2.1. How an Analyst-Programmer can Improve Practice

This section consists of straightforward examples of how analysts (methodologists) can be more productive with elementary programming ideas and how programmers (informaticians) can produce more useful software by incorporating basic analytic ideas.

Better Survey Data Analysis. Most survey data can be messy in the sense that certain data points are in error. To help evaluate the quality of a survey data base, a methodologist might wish to perform a regression analysis. The skills the statistician brings are understanding of regression and, in many situations, the ability to do straightforward regressions and plots using SAS. At the simplest level, all that the statistician may have to worry about is a few outliers. Straightforward outlier situations are found in courses and books that talk about regression. What may not be straightforward for the statistician, is writing 10-30 lines of SAS code that separate the outliers from the rest of the data. If the statistician has not written outlier-separation code, then one procedure would be to get a coding example from another statistician and ask advice of the other statistician. If the first statistician understands a few ideas of coding in SAS, then the first statistician can quickly pick up the remaining ideas on his or her own. If the statistician must learn a few coding methods in SAS, then the completion of the regression analysis will take longer. What is often very time-consuming and relatively unproductive is for the statistician to wait for a programmer who can provide support in getting the appropriate subset of the data. Typically, a programmer will not be available for weeks or months. When a programmer becomes available, he or she may require the data be put in a special form (sequential file). Then a program can be written in a language such as C or FORTRAN to get the needed subset of data. It is far more efficient for the statistician to learn to write the 10-30 lines of SAS code. Learning the SAS coding can be far easier than learning many statistical ideas. It has the further advantage that the statistician can more easily determine what additional analyses are needed on the data.

Increased Flexibility and Speed of Analyses. Suppose that the statistician is performing a longitudinal analysis of several data bases over a period of years. Industrial coding has changed over the years, and the statistician uses a table to convert the codes in the data from the early years. To do this, he or she can write SAS code with a large variety of if-then-else rules. Since the statistician worked on another project with programmers in another area, he or she knows that the programmers have general software for reformatting files and for table searches and replacements. After obtaining the software, the statistician is able to complete an analysis much more quickly than if he or she had written the SAS code. Using the programmers' software was more efficient for the statistician because he or she was familiar with the analyses, the formats and structures of the files, and the specifics of the needed table replacements. The programmers were unavailable to help the statistician. In any event, the most efficient course was the one taken by the statistician.

Improving Imputation. Suppose that the programmer is working on an edit/imputation system. During development, methodologists suggest adding (i.e., provide formulas for) a new imputation module. The programmer notices that the main FORTRAN code is not very modular. It would be much easier to add in new imputation options if a large section of the code is restructured. He or she has worked with other programmers who write good modular code and has learned how to write such code. There is sufficient time to do all the needed coding. The programmer rewrites most of the code in a modular form and adds a module for the new imputation method. To program the methods better, the programmer obtains detailed advice from the methodologist. When the initial changed code is available, the programmer figures out a better way of displaying certain summary outputs for him or her and the methodologist's review. The methodologist explains that certain marginal subtotals must be preserved and how specific variables are used in a regression. The programmer discovers that, with one imputation method, certain marginal subtotals are in severe disagreement with the comparable marginal subtotals. He or she corrects the code. During the review, the programmer figures out a way of improving estimation with one of the other imputation modules. The improvement in the system (in this situation) was primarily due to the programmer learning additional analytic procedures well enough to improve the code. The methodologist was crucial for explaining the analytic procedures.

The first example tells us that there are straightforward programming practices that a statistician can learn to improve data analytic skills. The second example describes a situation where the statistician applied useful programs that programmers had created to provide flexibility and speed of analyses. The result was that it was much easier for methodologists to determine how well things were working. In the third, the programmer was able to eliminate some subtle errors because the programmer better understood analyses and was able to provide more analytic options by rewriting code.

2.2. Projects an Agency Might Attempt, Existing Skills, and the Creation of New Skills

Statistical agencies often need to edit and impute the data in individual survey data bases. Rather than write custom code from scratch for each survey, creating a generalized system that can be used on most or all surveys is far more efficient. An immediate benefit is that, if an agency has only a few really good analyst-programmers, then the skills and abilities of a few can be more efficiently used. Generalized systems have the additional benefit in that they raise institutional skill level. With a well-implemented generalized system, a user (applier of the software) will often produce better results (survey data) than if the user has custom code written for the particular application. This is because the software will often use the best available methods, have fewer errors, and have a better designed user interface.

GEIS. To illustrate how analyst-programmers can affect development of a generalized edit/imputation system, we describe the creation of the Generalized and Edit Imputation System (GEIS) at Statistics

Canada. The first need was for an analyst-programmer to decide what theoretical ideas (if any) were available and could be translated into a usable prototype system. In creating the Numerical Edit and Imputation System (NEIS), Gordon Sande wrote code that integrated the Fellegi-Holt model of edit/imputation with several ideas from mathematics and operations research. For a statistical agency, a key feature of the development was having an individual who could realize widespread implications for improving survey practice by being able to develop workable code based on the Fellegi-Holt model. The type of knowledge needed by the analyst-programmer was knowledge of good methods in operations research and computer science and how to implement them in code. However, even more than the basic knowledge, Sande could learn new ideas and integrate them in the basic methodology. His initial work involved the use of Chernikova's method for finding extreme points in a region in R^n . Sande demonstrated that solutions to the Fellegi-Holt method could be implemented by a variant of Chernikova's method due to David S. Rubin.

To develop NEIS further, Statistics Canada created a team to enhance the speed of the algorithms, possibly refine code, create user-interfaces, rewrite the code from FORTRAN to C, and document the system. The team consisted of subject matter specialists, informaticians (programmers), and methodologists (mathematicians and statisticians). The completion of the new system (GEIS) took several years. The work is noteworthy because of the further skill development and system improvements that the team made. First, at least one individual (Yvan Davignon) was able to learn new methods for analysis and programming using the existing NEIS as the learning model. The result is exceedingly difficult C code that is remarkable in its clarity and organization. Methodologists (John Kovar, Iona Schiopu-Kratina and Jean-Marc Filion) were able to learn many new methods in mathematics and operations research. In particular, they refined some of the theory and developed a heuristic procedure that improved the overall speed by a factor of as much as 100 in some applications. The use of heuristic procedure is a good example of how real-world systems are often improved by individuals who understand aspects of the theory and work very closely with other individuals doing the code development. The team systematically developed new theoretical ideas and heuristic procedures in creating versions of the algorithms for which they performed speed comparisons on different data sets. Heuristic procedures are often used in testing various ideas and learning more about components of the basic algorithms. A heuristic procedure that involved a non-trivial reordering of one group of computations is associated with successive optimizations. This procedure yielded extreme speed improvements consistently.

There are two key aspects to be learned from the development. The first is that the individuals on the GEIS team, using NEIS as their model, were able to expand their skills and knowledge significantly. They taught themselves new theory and exceedingly difficult programming methods. The individuals who were in the group had good skills when they entered the group. The skills were often in areas that seemed very different from the analytic programming skills they created during their work in completing the project. The second is that the biggest speed improvement was due to a heuristic procedure developed by individuals who understood the methods and were looking very carefully at aspects of algorithm and code development. The team had effectively removed the isolation between analysts and programmers that exists in other development environments.

RECLINK. Statistical agencies often need to perform record linkage (computer matching) of names and addresses to create the frames for individual surveys. The Census Bureau's creation of its generalized RECLINK software has many parallels with Statistics Canada's GEIS development. The key differences that we emphasize are how the hardware situation affects the skills of the analyst-programmer and the need for individuals to reinvent themselves with different skills. Like GEIS, RECLINK began as a series of enhancements to generalized software created primarily by one individual, Mathew Jaro. The theoretical basis of modern record linkage is due to Newcombe and to Fellegi and Sunter. It uses extensions of odds ratios and the likelihood principle. The means of implementing this theory have primarily involved computer science. Jaro made dramatic improvements by introducing new methods of approximate string comparison for dealing with typographical error and a linear sum assignment procedure for forcing 1-1 matching. The RECLINK team primarily consisted of three analysts (statisticians Thomas Belin, Yves Thibaudeau, and William Winkler) and two programmers (computer scientists William LaPlant and Edward Porter). Their key tool for learning was the record linkage system created by Jaro. The initial needs were for better parameter estimation and for faster algorithms.

To understand details of record linkage and the specific needs for improved parameter estimation, the statisticians had to learn COBOL, a few of the algorithms and data structures in the main matching code, and to work on different computer systems on which they had worked previously. The crucial difference was that the statisticians had come from research environments in which methodologists developed new algorithms and wrote code. They were willing to learn COBOL and new computer systems. The statisticians did not revert to the situations in which they were relatively isolated from programmers. From the computer scientists, they learned good coding practices and how to work on the new computer systems (initially a small IBM mainframe and two IBM PCs and later also a VAX and two Unix workstations). The computer scientists, in turn, learned many methodological details associated with the parameter estimation methods for which the statisticians wrote the code.

As work progressed, it became clear that RECLINK development primarily involved writing code for difficult algorithms and their associated data structures. The two main parameter estimation problems were best dealt with by the two statisticians who had strong backgrounds in the theoretical and computational aspects of the problems. In solving some of the parameter estimation problems, Belin developed a method for accurately and automatically estimating error rates and Thibaudeau developed new ways of dealing with dependencies. Belin wrote difficult code based on the EM and SEM algorithms, and Thibaudeau on Newton-Raphson and scoring. Winkler decided to help with most of the work which was computer science. Since the quality of name and address standardization needed improvement, he learned methods for parsing (separating components into pieces). He first helped write modules for improving name standardization and later wrote preprocessing modules that put some addresses in a form that could be better processed by the main address standardization routine. With the creation of good parameter-estimation methods, the best way of improving matching was to improve the quality of the inputs to the main matching program. Because the COBOL code was much too difficult to use for research purposes, he then decided to learn C and to write the main matching program in that language. In writing the new matching code, he had to learn details of the linear sum assignment procedure, the string comparators, and various sophisticated search-and-retrieval algorithms. By understanding the string comparators, he was then able to develop modelling procedures on how the values returned by the string comparators interacted with the parameters returned by the EM algorithm. By understanding and subsequently developing a new linear sum assignment procedure, he was able to understand better and improve overall matching efficacy. What is important is that understanding computer science and operations research lead to direct improvement in the decision rules of the statistical model of Fellegi and Sunter.

One hardware environment (traditional UNISYS mainframe) can be very helpful when computing resources are very minimal. The environment can also yield relative isolation of analysts (methodologists) and programmers (informaticians). With new computing environments, such an isolation can be counterproductive. Many traditional realms of the groups of programmers such as getting subsets of large files and producing tabulations can easily be performed by methodologists or individual programmers using packages such as SAS. Methodologists can easily do straightforward analytic programming tasks. What becomes clearer is that analytic programming needs for generalized systems involve much greater computer science abilities than were often needed in some traditional situations. Analysts need to understand more of the analytic uses of systems and to develop more much difficult data structures and algorithms. Programmers need to move theoretical ideas into practice and develop the broader vision that productive interaction with informaticians can yield.

2.3. Further Needs for Analytic Programming

Suppose that each statistical agency has a need to improve its methods of imputation. Modern imputation methods such has the EM algorithm, multiple imputation, Rao-Shao, and Gibbs sampling are known, almost uniformly, to improve the agency's ability to impute, model, understand, and explain their data. How does an agency create individuals who can perform these methods?

Suppose that one part of an agency has developed, as part of a specific production system, an analytic-programming product that does name standardization very well. Another part of the agency has a strong need for name standardization. How does the second part of the agency find out that the software exists? What skills are needed to understand the existing code, extract the essential parts, and modify it so it can be used? How does one analyse the inputs and outputs to improve the standardization?

An agency has a new analytic programming project and has successfully developed one group of analyst-programmers who have produced a generalized system that needs no foreseeable maintenance. How does the agency use these individuals on other projects? What analytic-programming skills are needed on the new project? How does the agency leverage its talents in training additional analyst-programmers? Which of the analyst-programmers can and need to develop specific new skills?

3. Good Institutional Practices

In this section, we present ideas related to how an agency might decide on what large or long-term analytic-programming projects to undertake. We also give a summary of the steps in a software development process.

Generalized survey software is not cheap to develop and maintain. Consequently, many questions arise from the above steps if the ultimate product is to be generalized software.

3.1. The Recognition of Necessary Methodology

Suppose that an agency spends significant resources on projects such as sampling or estimation. It needs an individual (or individuals) who can develop methods (theories: Särndal et al., Fuller, Rao) that have the potential to yield practical solutions. If a good theoretical method for estimation exists, the agency needs an individual (or individuals) to develop a prototype system that shows that code can be developed for a generalized system such as the Generalized Edit and Imputation System (GEIS). How does an agency get (or create) individuals who can determine important ideas and carry them out? Finding the right individuals is a matter of circumstances. If one individual with the right profile exists, he or she can encourage the development of software in other than in his or her domain of expertise. Or he or she can learn new skills to facilitate development. The creation of new and untried methodology takes courage and imagination by individuals. Higher management needs to identify such individuals, encourage them, and create an environment so that their skills and insights are better utilized. These individuals will often understand the more advanced methodologies. They will either be able to write their own code or translate ideas into a set of specifications for an analyst-programmer to implement. On their own initiative, they will also have prototyped a good deal of the procedure themselves. The model of an individual coming up with new ideas and developing prototypes has led to several generalized systems at Statistics Canada. For example, GEIS was put together by rebuilding an earlier system called NEIS. On the other hand, the Generalized Estimation System, GES evolved to its present state by a more direct route. No prototyping and demonstration of the value of the concepts used in GES were needed. This was due to the better understanding of the need for and use of auxiliary data by theoretical statisticians and methodologists.

To demonstrate the practical validity of concepts, prototyping is also a key to success. Census methodologists at Statistics Canada have used this approach in creating estimation and imputation modules, even when the methods are not used for general system development. For the Census estimation system, a small group of methodologists wrote the prototype in SAS IML. In developing methodologies, computational feasibility was always their major concern. They validated their approaches in terms of data quality and computational efficiency by generating several versions of prototype systems. Systems analysts and programmers took the prototype and created the production system. They did not need to alter the complicated routines in the SAS IML code. They merely needed to write interface code to extract and then write data back to the Census data base. The development of the New Imputation Methodology (NIM) system proceeded similarly. Two methodologists wrote the first prototype. The second prototype was written by two methodologists and a systems analyst using FORTRAN. "Hybrid" methodologists and system analysts working on these projects were the keys to success. The ‘hybrid’ term means that the methodologists had an interest in programming and algorithms while the systems analysts had an interest in methodology and programming. Even though C is the official language for production systems, having a FORTRAN prototype showed operational feasibility and facilitated more general development. At an early stage in a project, the methodologists and systems analysts started working in a collaborative team. This collaboration carried through testing to the final product.

3.2. The Choice of Methodology to be Implemented as a Generalized (Specialized) Package

Given that required methodology has been identified, the choice of methodological problems that need to be computerized will differ among agencies. The choice depends on the agency’s need and its knowledge of methodology and systems. One common theme exists in Australia, Britain, Canada, France, the Netherlands, Sweden, and the United States. It is the requirement to develop software of a flexible and general nature that will process the following four major steps in a survey: (i) sampling, (ii) data capture and preliminary edit (iii) statistical editing and imputation, and (iv) estimation. Development of other tools such as for record linkage, confidentiality, checking automatic character recognition depends on the available expertise. Systems addressing the same survey step will differ among agencies. For instance, at Statistics Canada, the development of the Generalized Estimation System (GES) was very much driven by the requirement to do domain estimation and to take into account auxiliary data. At the U.S. Bureau of the Census, development of the generalized record linkage system RECLINK was primarily driven by the specific needs of the decennial census.

Topics of long-term interest at Statistics Canada include: (i) Generalized Systems (sampling, data capture and preliminary edit estimation, statistical edit and imputation, record linkage), (ii) the use of symbolic logic for developing a survey data analysis package, (iii) the possibility of using neural networks for imputation, and (iv) confidentiality. At the U.S. Bureau of the Census, the long-term projects are: (i) small area estimation, (ii) non-response imputation, (iii) administrative lists, and (iv) computer-assisted technologies.

3.3. The Four Stages of Software Development

If a project is being contemplated and has been decided upon, there are four stages to creating software:

1. Review and/or creation of theory and methods.
2. Creation of fast computational algorithms and prototype code.
3. Refinement of methods and development of production code.
4. Documentation and enhancement of user interfaces.

Many projects will begin at the urging of users of existing methods and software. The users will often be able to show how their productivity can increase with better tools. Other projects will begin after managers have found out that other agencies have developed certain tools. Some projects will begin when a methodologist has knowledge of new theoretical ideas that have potential application. A crucial aspect of many projects is that the methodologist have a very strong understanding of how computational algorithms are developed and computer code is structured. If this is not possible, then it may be necessary for a programmer to have a detailed understanding of the theory.

4. Creating and Enhancing Skills

In this section, we describe how an agency might create a situation in which it is easier to enhance the skills of the analyst-programmer, ideas on who an agency might quantify or certify new skills, and the required skills so that software might be passed within or among agencies.

4.1. Environment for Enhancing Skills of the Analyst-Programmer

The environment begins with individuals. Analyst-programmers need to show initiative in learning relevant new skills. They should be individuals with strong programming skills. The skills are not so much in the details of computer languages. They are more in thinking in a structured programming frame of mind. Analyst-programmers should have strong statistical or analytic skills, be able to extend existing methods and have a good intuition and grasp of concepts. If they organize in a methodical and meticulous manner, then they should be able to break larger problems into small components that are more easily solved.

The best way to find/develop analyst-programmers is to identify them at the recruitment level and steer them into a specialized career. This career path would differentiate them from pure systems analyst-programmers or /methodologists. Their specialty should be recognized in the career path. They should be able to be promoted within the specialty just as pure systems analyst-programmers or methodologists. The development implies that they get better at their job with time. They need to be rewarded via promotions. Existing analyst-programmers or methodologists can develop the required skills by better understanding each other's discipline.

How far do we decide the development of the analyst-programmer? What are the interactions of such individuals with specialized systems analysts, programmers, and methodologists? What should be the interaction between programmers, analyst-programmers, and methodologists to develop specialized software?

The success of the interaction depends on several factors: (i) Structure of the group, (ii) Management of the group, (iii) Stability of the group, and (iv) Computing environment.

The structure of the group and the interaction of its members. Methodology and systems (and users) should work together right from the start to develop products that are easy to understand and easy to maintain and enhance. An example of this approach is for methodology to develop required algorithms, prototype modules etc. in SAS to handle sophisticated statistical problems and then to turn the code over to systems. Methodologists need to prepare better specifications for developers that take into consideration their knowledge level: introduce ideas gradually; provide the motivation behind the ideas and reinforce these with simple examples; explain concepts first and then the formulas; and get developers to think in matrix structures rather than scalar thinking. The responsibility of finalizing system development would be fine tuning. This ensures that modules adhere to "standards" and fit into the framework of what already exists. At Statistics Canada, this co-development concept has been well demonstrated in the development of our sampling (GSAM), estimation (GES), edit and imputation GEIS), and data capture and collection systems (DC2). Having a small team on each project with team members working closely together is better. It is easier to manage and partition the work. A small team allows each member a greater role in how things are done. They should be physically located in the same area to foster team spirit and allow them to share ideas and concerns quickly. Systems analyst-programmers need to bring a "Systems Engineering" perspective to the task of creating generalized systems. Some of the best systems engineers are required to develop generalized systems. These systems have a much greater need than other software for solid construction because of the multiple platforms supported, the various (and sometimes unexpected!) ways in which the software is used, the long life and complex revision history, and the requirement for easy to use, yet flexible interfaces. Users should also be involved at the prototype level, and the users of these systems should figure prominently in the process that charts their evolution.

Leadership of the group. The responsibility for managing an analytical-programming group should be centralized to ease the interaction between parts of an agency. It is not necessary that the manager of such a group develop a fixed set of procedures. He or she should understand the needs of each member of the team (programmer, analyst-programmers, and methodologist). Good individuals are attracted and kept by managers that emphasize the success of the current products. They promote the challenging nature of the work and allow group members to attend statistical computing oriented conferences such as SUGI. If a suite of related "systems" (e.g., GES->GEIS->-GES->CONFID) is being developed, one manager should be in charge for the overall methodology and another one for the systems development. This greatly helps the development of required interfaces between the different systems in the suite.

Stability of the group. The development of generalized or specialized systems requires time for initial development, maintenance, and further development. Developers of such systems may need to be committed for as long as several years. Stability is enhanced if the organizer/leader of the group can choose the members of the team. Often, the greatest amount of time is needed by individuals to learn the specific methods and programming techniques of a new area. If an advisor who was a member of a previous group is available, then group interaction can be facilitated. Problems (bugs) with systems can be fixed faster by those who developed the code. Enhancing an existing system requires intimate knowledge of the system, including the underlying reasons for choosing one type of data structures and coding methods versus another. Rotating individuals too soon out of the area is inefficient because it implies a complete re-learning of the system to attain the required in depth knowledge of the system.

Computing environment. The environment provided by Unix workstations with good windowing or by Windows 95 and Windows NT on IBM PCs is conducive to good analysis and programming. A key feature of the new hardware environment is relatively quick turnaround. The analyst-programmers can quickly see the results of a statistical analysis. The programmer can quickly compile and run programs. Modern debugging tools yield far superior results in comparison with results via previous methods. SAS is ideal because it is understood (possibly at different levels) by both programmers and methodologists. This understanding has the advantage of reducing the communication problems (between systems and methodology) and speeding up the development of statistically complicated components.

4.2. Certification of Skills

The goal in a statistical agency that is to be empowered with analytic programming is to have individuals who take the initiative in learning new skills. To demonstrate specific skill development, a certification program can be useful provided there is little overhead and that it can lead to useful improvements in skills. An informal certification might involve an individual and his or her supervisor delineating a set of skills in which the individual will learn and demonstrate proficiency.

Analysts (Methodologists)
Certification of proficiency in a subset of a set of basic statistical skills and a subset of skills with statistical packages. Skills in a programming language. Knowledge of algorithms, particularly statistical.

Programmers (Informaticians)
Certification of a basic subset of analytic skills. Knowledge of algorithms. Skills in using a statistical package. Experience in giving short talks, briefings, or seminars.

Good Practices

1. Having analysts learn programming in a statistical package such as SAS. Learning programming skills to do explanatory data analysis.
2. Having analysts and programmers work in teams.
3. Having seminars for programmers. This is a way of getting new skills or knowing that acquiring them is possible.
4. Having managers that will set aside time so that analysts can learn programming skills and programmers can learn analytic skills.

Statistical Example. The statistician can do a messy data analysis in a straightforward statistical situation using SAS.

Programming Example. The programmer can write code for a straightforward binary search or simple parsing and string comparison methods.

4.3. Sharing Software Among Agencies (Or Within an Agency)

This section begins with the three methods for getting needed software. It ends with the skills needed by the analyst-programmer if software is to be shared effectively or off-the-shelf software is to be used .

Getting Software that Meets an Agency's Needs

There are three methods of getting software. They are; developing it with in-house, contracting out, or purchasing general-purpose off-the-shelf software. Evaluating options depends on whether the agency has analyst-programmers who can effectively estimate cost, portability, and knowledge considerations. Buying software may be cheaper if it meets an agency's requirement, even if the software requires the purchase of new hardware. If needed software is not commercially available, doing an entire development can be very expensive. Understanding whether outside software can benefit a project requires almost identical skills to understanding whether in-house generalized software can be used on many surveys. One impediment will be the same. Users will want specific features in existing software. The analyst-programmer will need to determine what is really needed. The determination may mean performing a comparative analysis of results produced by the generalized system and the specific survey system.

Contracting out is another possibility. For example, Statistics Canada is exploring the use of symbolic algebra in survey estimation by having an outside consultant build the software using Mathematica. The intent is to mesh the resulting code with GES to build a survey data analysis package. Contracting can be efficient if the statistical agency clearly understands the specific statistical service to be performed and the contractor does not have to spend significant amounts of time developing individuals with the needed skills. Contracting can be very inefficient if the agency does not have the time and the skills necessary for specifying the contractor's work or monitoring and evaluating deliverables such as software and analytic products.

Communicating Ideas

The fundamental concept is that analyst-programmers communicate ideas, good practices, and methods via code. There are two facets. The first is that the analyst-programmer can understand well-written code and run it on his or her computer system. The second is that he or she can write and disseminate code that others can understand and learn from. An analyst (methodologist) might write a prototype system and provide it to the programmer (informatician). In this way, he or she would demonstrate that a theoretical concept could be moved into practice. The prototype might be written in SAS. If the programmer needs to write a system that can be used for production, then the programmer might have to write new code in a language such as C or FORTRAN. If the code is poorly structured, has slow algorithms, or does i/o poorly, the programmer might rewrite portions of the code or give the analyst examples of code that performs similar functions that are well written.

Building Skills to Understand Code

The basic skills needed for understanding and writing good code are built in a manner similar to the manner in which a theoretician progressively learns more advanced ideas. The analyst-programmer begins with basic ideas such as simple i/o, straightforward data structures, and elementary do loops. He or she progresses to more advanced i/o and data structures and to elementary algorithms such as tables searches, sorting of simple structures, and approximate string comparison. Depending on the need, the analyst-programmer might proceed to advanced hashing methods, tree-based searches, sophisticated parsing, integer programming, set covering, random number generation, replication, and ways of sampling for different types of probability distributions. Once the analyst-programmer has built high-level skills, reading difficult code written by others will be more straightforward for him or her. Understanding of well-written code corresponding to difficult algorithms is similar to understanding of the details of a mathematical proof. To extend theory, theoreticians must be able to understand the details of existing methods and develop details of the new theory. Analyst-programmers, at all levels of expertise, need to understand details of existing code and to create the details necessary for new code. A crucial advantage that the analyst-programmer has over the theoretician is that the analyst-programmer can run versions of segments of code, look at sample inputs and outputs, and step through all details using a debugger.

Generality of Knowledge

The analyst-programmer wishes to write good software that might be used by others in the statistical agency. To do this effectively he or she might have to know several different computer systems, several programming languages, and be familiar with commonalities and nuances of software written in several different portions of the statistical agency. Understanding of coding methods increases when an individual has done serious analytic-programming work in two or more areas of an agency. The biggest increase may be in the appreciation of the similarities of concepts of two sets of code. For instance, the analyst-programmer might be familiar with ratio edit systems for two surveys. The functionality of the core algorithms would be very similar. Possibly only the user-interfaces are different. In this situation, the two systems could likely be replaced by a generalized system. Analyst-programmers can take the best logic and user interfaces from specific systems in producing the generalized system. If a small subgroup of an agency does good training of programmers or has many good examples of coding methods, then it is likely that anyone joining the subgroup would learn good methods. The way to institutionalize the good methods is to have relatively standardized training methods and many examples of good coding techniques. Learning an additional programming language is straightforward for the analyst-programmer if the individual is a good programmer in another language. Learning a different language or different computer system is actually much easier than it is to learn good programming ideas and difficult algorithms. Learning two or more languages and coding a few algorithms shows the basic similarity of the differing languages.

5. Summarizing remarks

There are two critical aspects for recognizing good methodology. The first is to have individuals who understand agency practices and can look for and understand new ideas. The second is to have individuals who translate them into a set of specifications on how the new ideas might be incorporated into practice. The best specification would consist of a prototype system that demonstrates the feasibility of many of the concepts. Implementation requires that individuals have a variety of both analytic and programming skills and the capability of rapidly learning new skills. Some individuals might be hybrids having several skills to recognize, develop, and implement new ideas on the computer.

Analytic-programming can empower a statistical organization. At one level, creating individuals who can undertake the most difficult project is useful. At another level, having systematic ways to improve the analytic and computer skills of individuals at a statistical agency is advantageous. In all situations, the ideal is to have individuals learn many ideas predominantly on their own and to have analyst-programmers who acquire new skills and pass them onto others.