APPLYING SOFTWARE ENGINEERING TO A GENERAL PURPOSE GEOGRAPHIC INFORMATION SYSTEM

Peter Aronson
Environmental Systems Research Institute
380 New York Street
Redlands, California 92373
(714) 793-2853

ABSTRACT

Over the past 14 years, ESRI has constructed three commercial general purpose Geographic Information Systems. The first two, PIOS and GRID, were constructed in the haphazard fashion typical of almost all past GIS development. The most recent, ARC/INFO, was from its initial design, constructed following modern software design and programming methodologies. In the course of the design, construction and maintenance of ARC/INFO, many lessons about the development of large software systems in general, and of GIS in particular, were learned.

INTRODUCTION

Most successful geographic data-processing software has not been designed by computer professionals (although there are exceptions, see Tomlinson, 1974). Such software has generally been designed by geographers or planners or foresters with limited programming background. This is due to the obscure nature of most geographic data-processing -- typically, no one outside those fields dealing with geographic data has had a sufficient grasp of its special requirements to produce useful tools to manipulate it.

ESRI's earlier GIS grew from such specific software packages (PIOS started as a simple package to digitize and to report overlay areas (Tomlinson, et al, 1976). They were not really planned as systems. The initial package was designed; then, whenever a new function was required, a new program or subroutine would be added to perform it. The systems simply grew. As long as a system fulfilled its current requirements, no matter how poorly, no overall system plan would be made.

In the winter of 1980-1981 the initial specifications for what was to become ARC/INFO were put together by Scott Morehouse. The original plan was for an arc/node digitizing system (Guevara, 1983), including an automatic topological "cleaning" program that would break line segments at intersections, snap closed undershoots and remove overshoots, and would feed into the PIOS system. This package was carefully designed, constructed and tested (and is currently still being employed after four years of heavy use).

At this time, dissatisfaction with PIOS prompted the next stage of ARC/INFO's development. A complete new design was

performed (actually, the design had begun with the design of the digitizing system, however, for internal political reasons, the processes were officially separate). The ARC/INFO Geographic Toolbox concept was developed (see below), then the development of the ARC/INFO full geographic information system began.

ARC/INFO DESIGN REQUIREMENTS

ARC/INFO was planned as a general purpose geographic information system, including commands for data input, analysis, output and management. It needed to be efficient to compete with special purpose systems. Most important, it had to be well designed and built.

That ARC/INFO be well engineered was vital for a number of reasons. First, because of practical limitations it would have to be built in stages, with capabilities being added as time went on. This would require that the system be carefully broken down into logical modules. Time constraints required that some problems be dealt with initially by simple, but non-optimum solutions, then later redone in a more efficient manner. For example, the original attribute handling after an overlay was done by a job control program, then recoded into FORTRAN a year later. This required that each module have well-defined inputs and outputs, as well as not generating side effects.

Second, the system would have to be easily expandable, both in the sense of adding new functions, and in increasing functional limits (such as the number of points in a polygon). When a system has a large and varied user base, there are sure to be additional capabilities required over time, not all of which can be anticipated at design time. For example, ARC/INFO did not have point-in-polygon overlay or point subsetting until Spring 1983 when they were requested by the State of Washington, Department of Natural Resources.

Third, it had to be portable. It was quite certain at the beginning that ESRI would at some point want to have ARC/INFO running on some machine other than PRIME, ESRI's current in-house development machine. PIOS and GRID were, at that time, available on PRIME, HP 3000 and IBM.

Finally, it had to be maintainable. Fixing problems in PIOS or GRID usually required either spending considerable time reading and comprehending obscure code, or writing a new program from scratch. This led to patched together systems where it was often easier to find some other way to solve a problem than to fix a bug.

ARC/INFO DESIGN PHILOSOPHY AND STRUCTURE

ARC/INFO was designed as a "toolbox" of geographic operators. ARC/INFO commands perform such operations as: overlay (Union, Intersection, Identity and Update), subsetting (Clip, Reselection, Sliver Elimination, Polygon Dissolve, Erase (reverse clip), and Multiway Split), combination (Mapjoin and Append), data input (Digitizing,

Generation, Grid-to-Polygon, COGO and Conversion programs), output (Plotting, Conversion programs and Report Generation), modeling (Theissen Polygon Generation, Network Analysis, INFO (relational database manager) and Corridor Generation) and data management (Librarian Map Library Manager). ARC/INFO commands can be thought of as statements in a geographic modeling language. Combined via system job control language (CPL on PRIME, DCL on VAX, CLI on Data General and EXEC on IBM CMS/VM) with INFO's relational programming language, complex spatial models can be implemented as relatively simple "programs".

A simple example of such a program could take as input a map (coverage in ARC/INFO terminology) of an area within a map library, and produce a report of the estimated total value of state-owned forest stands and a map of those stands shaded by value per acre. The procedure would go something like this:

```
PROCEDURE STATE-FOREST-VALUE (AREA)
/*                                                          */
/* This procedure produces an estimated value report and    */
/* an estimated value per acre plot for a specified area.   */
/*                                                          */

EXTRACT from Map Library FOREST coverages: FOREST-STANDS,
SOILS and OWNERSHIP by AREA.

/*                                                          */
/* Overlay FOREST-STANDS with SOILS and OWNERSHIP to        */
/* produce a coverage that contains all the information     */
/* required to calculate cost and owner.  Remove those      */
/* polygons not owned by the state.                         */
/*                                                          */
IDENTITY FOREST-STANDS with SOILS to produce COST

INTERSECT COST and OWNERSHIP to produce VALUE

RESELECT VALUE so that OWNER = 'STATE' to produce
STATE-VALUE

/*                                                          */
/* Create products.                                         */
/*                                                          */
Model STATE-VALUE with INFO to produce Estimated Value and
Estimated Value per Acre (ESTVAL/AC), and produce a summary
report of Estimated Value and add ESTVAL/AC to STATE-VALUE.

Generate a plot of STATE-VALUE shaded by ESTVAL/AC.

Clean up (Delete FOREST-STANDS, SOILS, OWNERSHIP, COST,
VALUE and STATE-VALUE.

END
```

This adaptable structure is mirrored inside ARC/INFO as well. ARC/INFO consists of about eighty FORTRAN 77 programs linked in various fashions by job control language. The CLIP command, for example, consists of nine FORTRAN programs (not all of which would be used by any single execution)

linked together by about (on the PRIME) 60 lines of job control language. Many times new ARC/INFO commands have been created simply by linking these programs in new combinations ("As above, so below").

THE ARC/INFO SOFTWARE LIFE-CYCLE

Most commercial general purpose software systems are in a continuous state of development. Operating systems, for example, are re-released at a consistent rate, often with fairly large changes in functionality coming every year or so. A general purpose geographic information system, like ARC/INFO, is also in such a state. There is typically a new release every four to six months, and each release is sure to contain one or more new commands, alteration of two or three existing commands, and a reasonable number of fixed bugs. As a result, ARC/INFO has its own internal development process.

The software life-cycle is one of the basic concepts in software engineering. The waterfall life-cycle model shown in Figure 1 is based on one by Barry W. Boehm (Boehm, 1981) and modified for the ARC/INFO development process. This model differs from the usual model in that it starts from requirements (as a result of ESRI's operating procedure discussed below) and the incorporation of phototyping as separate steps.

Step by step the ARC/INFO life-cycle consists of:

1) Requirements. Requirements for ARC/INFO come from two sources: software modification request forms and the systems group's section of the yearly company plan. The software modification request forms (commonly called "bug sheets") are filled out either by in-house personnel (usually from the production group or from the systems group) or by the software support group by user request. These consist of reports of problems in the software ("bugs") and of requests for enhancements. The yearly company plan contains those enhancements that the management feels would be useful in selling new systems or in satisfying an important requirement of current users.

2) Feasibility. This step is performed by the system architect. The requirements are examined for practicality and necessity, and those that can be performed acceptably by existing software are usually rejected at this point. There is a distinct effort made to minimize the amount of code in ARC/INFO, when it can be done without sacrificing significant functionality. Code that is not written does not have to be maintained. Those requests that would require excessive programmer time are usually tabled or rejected at this point (however, occasional major system additions are made, such as Network Analysis and Map Library management packages added early in 1985). Those modifications deemed practical and desirable are then passed on to the design stage.

3) Product Design. This task is performed by one or two development programmers, usually acting under the super-

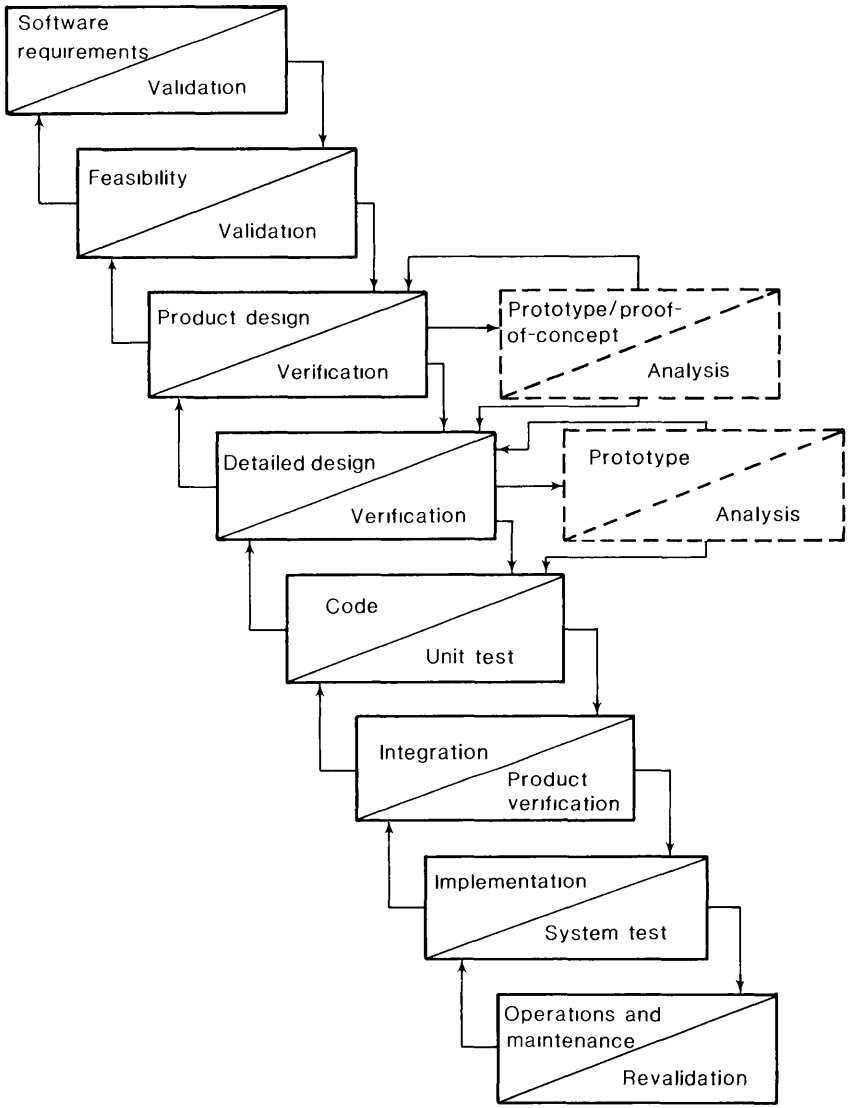

FIGURE 1 - The ARC/INFO Software Life-Cycle

(based on Boehm, 1981)

vision of the system architect. Performed at this time are the four major general design tasks: user interface design, initial algorithm selection, data flow design, and major module specification.

The user interface is usually described in terms of an entry in the command reference manual. Inputs, outputs, prompts and subcommands are fully described. This design also serves as a rough draft for the user documentation and on-line help files.

Boehm placed algorithm selection under detailed design, but in GIS development, where the algorithms are not always well understood, it has been found necessary to have at least an initial approach to an algorithm before proceding any further. At this stage, algorithms can often be compared via analysis using computational geometry (Guevara, 1983). If this proves inadequate, then prototyping may be required (see Step 4 below).

A data flow design shows each program or operational module in a command and each file/data structure in and out of each program. It looks at programs/modules as operations on files/data structures. Since most ARC/INFO commands are sets of programs linked operationally by files, a data flow design can be very helpful.

Defining subroutine modules might seem out of order at this stage, but in ARC/INFO the careful design of the low-level subroutine modules is considered as important or even more important than that of high level program sections. These subroutine modules, each based on a single data structure (such as variable record-length, random access files or coverage boundaries, or the user terminal), are the basic building blocks of ARC/INFO. As well as structuring the code and performing information hiding, these modules supply the primitives used in the construction of geographic operators. This approach has allowed ARC/INFO to be programmed in FORTRAN (selected for portability), without any problems due to the limited data structures available, as all major data structures are accessed via subroutine calls (70%+ of ARC/INFO code (exclusive of comments) consists of subprogram calls and control statements).

4) Prototyping/Proof-of-Concept. This is an optional stage, used as an aid in algorithm selection. It often helps in choosing an algorithm to write a small program to perform comparative testing. This process can be invaluable for selecting among competing algorithms when mathematical analysis is inadequate. The current sets of sorts used in polygon overlay and the search algorithm used by the arc editor were both selected in this fashion.

5) Detailed Design. A complete algorithm definition is produced. Pseudo-code is generated for all the main program sections. Both file formats and module contents are fully defined.

6) Prototype. This optional stage is used when the detailed design will not, or cannot, be carried past a

certain point. This usually indicates that there is not enough information available to complete the design. Often, building a simple, quick version of the product will raise all of the design questions, allowing them to be answered before the final coding. This is particularly useful when constructing a function that no one involved in the development has experience with, and for this there is little or no published information available. The Map Librarian, for example, went through this stage.

7) Coding. A complete ARC/INFO command is constructed and tested by the development group. This is installed into the active code set, along with any changes the installation requires in other existing code.

8) Integration. The command reference manual, user manual, training notes and programmers manual are updated for changes. The software support group performs independent testing of the new/modified command(s). At this stage, alterations in the operation of the software being tested may be requested by the software support group. An alpha release is made to inhouse users.

9) Implementation. The new software is officially included into the next beta release, and listed in the installation notes. After beta release, if no new problems are reported it is included in the normal release. Finally, it is then installed on all systems and trained.

10) Operation and Maintenance. As the software is operated, problems or shortcomings are discovered and transcribed onto software modification request forms. This cycle continues until either the software is entirely replaced or the function it performs is no longer needed (the former has happened to the original line-overlay command OVERLINE, the latter will happen to the ARC to PIOS conversion command ARCIDC).

PROGRAMMING PROVERBS FOR GIS PROGRAMMERS

This section is a collection of small tidbits of advice for prospective programmers of geographic information systems (as well as being my tribute to Henry F. Ledgard). For the most part, these simply reiterate the material discussed above, but in a more succinct form.

Always minimalize code. The more code you have, the greater the maintenance load. Whenever possible, use existing code in preference to writing new code. Deprogram (reduce the amount of code without reducing functionality or readability) whenever the opportunity arises.

Construct useful subroutine packages. Low level functions will be used again and again. Putting them in coherent packages allows simple reimplementation, isolation of system dependent code, and information hiding.

Any special case that can occur, will occur. Because of the vast quantities of information involved in geographic and cartographic data processing, GIS procedure algorithms

must be particularly robust, as any possible special case will arise in short order (and typically in a vital data set).

When in doubt, test. If the choice between two algorithms is unclear, submit your case to the computer. Write test programs for each case, and run them through a variety of typical and extreme cases to learn their behavior. Then choose the algorithm on the basis of experimental results.

GIS procedure algorithms will have unexpected characteristics when coded. GIS procedures are often not well understood. Formal analysis can only tell so much. A host of practical issues involving machine precision, special cases, and so forth, are sure to arise in the coding. This characteristic, combined with the above lesson, leads to the requirement that GIS require extensive prototyping before acceptance.

A general purpose GIS needs to be made of flexible building blocks, to allow new functions to be easily added. Since a general purpose GIS is not locked into a single set of operations, from time to time a user will require some capability that has not been hitherto present in the system. It is necessary that such capabilities not be difficult to add.

CONCLUSIONS

The design and development methodology of ARC/INFO presented above was the result of a combination of the deliberate application of software engineering methodologies and of trial and error. Even given a complete set of methodologies and techniques, it is not a simple matter to apply them. Long ingrained procedures must be removed, and new habits learned in their place. The procedures described here, simple as they are, can seem unwieldly and overly slow to an overworked programming staff. There is often a strong temptation to skip methodology and "just fix the problem". However, the consequences of ignoring methodology in differing versions and bad releases will drive home the need to follow a well defined procedure.

Geographic information systems are, by their nature, large and complex systems. If they are not well built, they will simply not work well. Software engineering is the logical application of methodologies such that software products and systems are well constructed. Therefore, usable geographic information systems need to be constructed following the principles of software engineering.

REFERENCES

Boehm, B.W. 1981, Software Engineering Economics, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Guevara, J.A. 1983, A Framework for the Analysis of Geographic Information System Procedures: The Polygon Overlay Problem, Computational Complexity and Polyline Intersection, Unpublished Ph.D Dissertation,SUNY at Buffalo.

Tomlinson, R.F. 1974, Geographic Information Systems, Spatial Data Analysis, and Decision Making in Government, Unpublished Ph.D. Thesis, University of London.

Tomlinson, R.F., Calkins, H.W., Marble, D.F. 1976, Computer Handling of Geographic Data, The Unesco Press.