# AN EVOLUTIONARY SOLUTION TO ANESTHESIA AUTOMATED RECORD KEEPING

*Alvin A. Bicker, PhD, John S. Gage, MD, and Paul J. Poppers, MD*

Bicker AA, Gage JS, Poppers PJ. An evolutionary solution to anesthesia automated record keeping.

J Clin Monit 1998; 14: 421–424

**ABSTRACT.** In the course of five years the development of an automated anesthesia record keeper has evolved through nearly a dozen stages, each marked by new features and sophistication. Commodity PC hardware and software minimized development costs. Object oriented analysis, programming and design supported the process of change. In addition, we developed an evolutionary strategy that optimized motivation, risk management, and maximized return on investment. Besides providing record keeping services, the system supports educational and research activities and through a flexible plotting paradigm, supports each anesthesiologist's focus on physiological data during and after anesthesia.

**KEY WORDS.** Automated record keeper, evolution, management of risk, object-oriented.

## PARADIGM OF FIXED MODEL ANALYSIS

In Science and Engineering it has been felt that fixed models of analysis and solution are the proper approach to problem solving. As larger and more complicated projects were attempted, conventional wisdom still held the fixed model as gospel: everything becoming known if studied long enough with adequate resources.

In anesthesiology, the introduction of technology coincided with the arrival of engineers, and the paradigm of fixed problem analysis and solution has dominated most applications of technology therein. Yet in anesthesia information systems, the lack of success of a system has usually been attributed to improper study of the problem, improper design of the solution, or improper implementation of the design. Virtually never have the underlying beliefs about the fixed-problem-solving approach been questioned.

### *Evolution of requirements*

Engineers believe that a changing problem description denotes an inadequate understanding of the problem and thus change is to be avoided. An engineer encountering this evolution phenomenon looks for ways to keep the problem from changing. Eventually most development projects impose a deadline past which changes are not considered. But the problem has not stopped changing; the engineer has simply enforced an artificial steady state in its description.

From the Department of Anesthesiology, State University of New York at Stony Brook, Stony Brook, NY 11794, U.S.A.

Received Mar 25, 1997; and in revised form May 29, 1998. Accepted for publication 9 Jun, 1998.

Address correspondence to Dr Alvin Bicker, State University of New York at Stony Brook, Stony Brook, New York 11794-8480, U.S.A.

## EXPERIENCES OF THE PROJECT

Information system problem solving is constrained by the model of the system and the model for analysis and programming. To construct an automated record keeper the first model is how anesthesia is conducted. Yet professional discussions about issues in anesthesia indicate that there are differences of opinion and areas of unknowns in this model.

The second model, the practice of analysis and programming, is less clear; yet many medical, computing, and management professionals assert the model is well known. It is complicated by rapid evolution of technology as well as the scarcity of information professionals who are experienced in the medical domain.

Fortunately this rapid evolution of technology has provided motivation and opportunity to see how technology can be used to support change [1]. A generation in computing is about 3 to 4 years, much shorter than that in other disciplines. The search for methods and principles to cope with this problem has produced new concepts as well as improved tools and strategies. No doubt more will arise in the future.

### *Automated record keeping project*

We began our automated record keeping project five years ago with the agreement that the problem was too big to be tackled in its entirety, and that regulatory and information system forces external to our environment would impose unpredictable requirements. We knew that the very process of design and programming would always provide new insights into problem and solution. Rather than artificially suppress these factors, we decided to incorporate and exploit them.

We adopted a partial-problem solving strategy: we would design a general architectural plan, and then build the solution step by step. Each step should thus fit into the architectural framework with minimal disruption of previously developed solutions. Object oriented analysis and programming tools were selected because they support complexity and change [2]. We also developed a risk management strategy that quickly and at minimal cost identified impractical concepts. We avoided modeling objects as fixed quantities. We further analyzed the essence of every component in a generic and abstract way to achieve general solutions.

### *Series of stages*

In constructing an anesthesia record keeper for our operating room with its particular anesthetic and monitoring equipment, we utilized commodity PC clone hardware and a relatively uncomplicated operating system (DOS). To support our programming, we chose the C++ object oriented language (Borland C++ compiler). Development proceeded in a series of stages, each with a definable set of goals that advanced the system to a higher level of functionality (Table 1).

## PRELIMINARY FINDINGS

Results show that an evolutionary problem solving approach can produce an information system of increasing complexity that keeps pace with, or exceeds those available from commercial sources. It has met new demands not conceptualized originally. It provides a medical record printout that can replace a manual record. Through its flexible multivariable plot paradigm it offers new perceptions of data to support decision making in complex situations. Because evolution is under local control, new concepts can be added.

Unexpected difficulties were experienced. The DOS 640 k memory limit became a problem. We were able to raise this limit by using the Virtual Object Overlay Manager (part of Borland C++), which oversees the swapping of currently inactive objects out of RAM memory so that others can be swapped in and run on demand. Our monitoring program exceeds 900 k in size and runs with no apparent speed handicap.

In addition, the system's overall architecture underwent numerous minor changes and several major ones as requirements evolved. The object orientation of the design facilitated this evolution. Because data and processes are bundled into classes and not scattered randomly, much functionality is protected during design changes. Objects evolved substantially and they continue to evolve. New properties are added; old ones get redefined. When we identify common functionality in several classes we move this into a single base class and then re-integrate it via inheritance. This is a powerful tool to eliminate code redundancy.

We have also realized the efficiencies of code reuse with the object model. The data storage/retrieval and graphics display subsystem is incorporated without alteration in the monitoring system, the review and analysis system, and the computer-assisted instruction system. This has allowed us to achieve a common user interface across the three vital domains of research, education, and clinical activity. Commercial designs

*Table 1. The stages of evolution encountered during the development of the record keeper*

| Stage | Activity |
|---|---|
| I | Construction of the communication interface to the anesthesia machines (N.A. Drager Narkomed 3, 4, & 2C). Added data storage and tabular printout features. |
| II | Addition of an interface to the physiological monitor (Siemens 404), followed by graphical display of data and capture of notes, and discrete variable information, a general case review facility utilizing the same graphics system, and interfaces to an infusion pump (IMED Gemini) and the EMG monitor (Datex). |
| III | Adaptation of the review capabilities into a computer assisted instruction (CAI) paradigm that supports the formulation of multiple choice question and answer rationales at selected time points of an anesthetic procedure, allowing clinical data to be utilized for teaching [3]. |
| VI | Adaptation of the monitor to study a new anesthetic (Sevoflurane). Interfaces for four new monitors (Ohmeda 7800 ventilator, Datex Capnomac, Nelcor 2000 pulse-oximeter, and Omega 1400 NIBP), a custom tabular printout and an event marker facility were added. |
| V | Extension of the CAI system to construct a didactic program from a collection of short segments (snippets) of clinical cases. This program is able to focus on a particular physiological management issue. |
| VI | Expansion of the system to capture all data needed for a printout of an official medical record. Demographics, physician data, medical history and diagnoses, anesthetic management, and fluid replacement are entered by keyboard. |
| VII | Adaptation of the communications front end to work with the Medical Information Bus (IEEE standard 1073) prototype hardware and software (ILC Data Device Corporation) to replace RS232 communication links to monitors [4]. Demonstration of the Plug and Play and the Virtual Medical Device features of MIB. |
| VIII | Adaptation of the system to a laptop computer with a single serial port by moving the communications front end to an outboard DOS-based communications concentrator. |
| IX | Addition of configurable customization options to support anesthesia delivery during electro-convulsive shock therapy [5], including multiple variables per plot. |
| X | Addition of control features to the plot subsystem so as to save the current graphics setup along with the data set for a given case. The ability to define graphic templates by name for automatic setup of the plots for different patients was also incorporated. |
| XI | Incorporation of an anesthesia-event-knowledge base viewable simultaneously with monitor data by means of a split screen format [6]. |

generally do not incorporate support for research and teaching.

## SUPPORTING EVOLUTION

Because the expense of building computerized medical information systems has been high and the ability to manage such projects is low, the trend has been to purchase ready-made systems. When one includes the requirement for responsive evolution to the user's needs, it is doubtful that this approach is cost effective.

The key question is "*What* should be bought from vendors?" It is generally agreed that hardware, operating systems, network operating systems, relational databases, word processors, and spreadsheets are fairly sophisticated and mature entities, available at commodity prices. But for medical applications, it is difficult to obtain mature components at commodity prices, especially ones that support evolution. It seems wiser to buy the *tools* of analysis, design and programming that empower us to construct the evolutionary components, reserving the maximum potential for evolution *within* the organization.

### *Development methods that support evolution*

Object-oriented analysis, design, and programming tools have produced a fundamental change in the approach to building a system. In the pre-object era (a.k.a. procedural programming) there were no boundaries within a computer program's memory space. All variables were in a collective pot along with the program instructions; unwanted, unplanned interaction easily occurred through addressing and naming errors. Various naming methods and partitioning schemes were attempted, but success depended too much on human

discipline. Efforts to isolate components from one another eventually led to the concept of encapsulation.

Today's object languages provide a logical encapsulation of the inner working of an object and rigorous control of the interface with the rest of the system, very similar to what the biological membrane provides for the cell. Just as nature uses cellular constructs to support complexity, object programming can significantly improve the result of investments in information technology. The most important fundamental object concepts are the following:

**Data abstraction**, through the collection of variables and processes associated with individual objects, supports the modeling of real world objects into structural design units, called classes.

**Encapsulation** creates tight logical boundaries around each of the objects created from a given class, and it eliminates undesireable interaction between objects. Communication between objects occurs through tightly defined *interfaces* across these boundaries.

**Inheritance** allows one object to inherit characteristics from another, rather than recreate that function and incur the evolutionary overhead of redundant design.

Even well designed objects evolve over time, acquiring additional features; sometimes they break into several sub-objects as new understanding is achieved. Programmers sense a continued investment in the objects of a system, which is lacking in the develop-and-leave experience of traditional procedural methods.

Programming languages are themselves evolving at a rapid rate. Object orientation in the strict C++ sense is powerful and very detailed. Recently, Java has emerged as a "safer" evolution of C++, designed to support Internet and intranet information paradigms. Component software programming systems, such as Visual C++ & JBuilder, have attracted interest because many of the modules one needs are already available as software components. In addition, much programming can be done with the built-in visual drag and drop paradigm. These systems include interfaces to data modeling tools, data access to most relational databases, and a visual programming paradigm for GUI applications.

A number of conclusions and suggestions support evolutionary program development.

1. A team of creative users and creative engineers utilizing rapid prototyping methods can achieve a highly efficient process in minimal time.
2. Success in evolution benefits from a strategic approach to managing risk. A new feature that is strictly additive offers minimal risk since it can be detached during development and testing, leaving the prior system intact. One that requires redesign of an existing module is riskier since the system requires modification. When several existing modules are involved, the risk is even greater.
3. Exploit the object paradigm to achieve clarity in architecture and operation. Objects and architecture evolve as work progresses. Objects and interfaces must be carefully defined to restrict subsequent impact of change and enhance testing.
4. Everything programmers build will one day have to be modified. When redesigning or extending the features of an object, programmers can either read the earlier imbedded notes or re-derive the logic by reading the code. In the evolutionary model, undocumented programming quickly becomes expensive.

In conclusion, by adopting an evolutionary problem/solution model, we have achieved considerable success in building an automated record keeper for the anesthesiologist. In addition we have gained valuable insight into the anesthesia information domain that will allow us to further enhance and expand the system.

## REFERENCES

1. Hammer M, Champy J. Reengineering the corporation. Harper Business, 1993
2. Booch G. Object oriented design. Redwood City (CA): Benjamin Cummings, 1991
3. Gage JS, Bicker A, Poppers PJ. Computer assisted instruction: From the clinic to the classroom. Proc. 14th Internat. Symp. Anesth. Intensive Care. Rotterdam: Erasmus University, 1994: 49–50
4. Kennelly RJ, Wittenber J. New IEEE standard enables data collection for medical applications. Proceedings – the Annual Symposium on Computer Applications in Medical Care, 1994: 531–535
5. Gage JS, Litman SJ, Bicker A, Poppers PJ. Automated record keeping for electroconvulsive therapy. Br J Anaesth 1995; 74; 24
6. Gage JS, Bicker A, Poppers PJ. Automated record keeping to advance anesthesia education. Proc. 16th Internat. Symp. Anesth. Intensive Care. Rotterdam: Erasmus University, 1996: 128–129