A Nontraditional Systems Analysis and Design: A Case Study

Bruce Mabis
Computer Science
University of Southern Indiana
Evansville, Indiana 47712
bmabis@usi.edu

Abstract

The termed *software crisis* was coined in 1968 by the NATO Software Engineering Conference for the myriad of problems in the development of quality software. The field of Software Engineering grew as a response to those problems and system analysis and design were recognized as important components of quality software. Since 1968, hardware costs have dramatically dropped and many software problems can be addressed through the use of *application utilities* - word processors, spreadsheets, data bases, etc. With the hardware and software in the hands of the users, program solutions can be created by the users, and thus, analysis and design may be done by "anyone."

We present a case study of the design of a system by an end user. The user was technology literate, but had no formal training on systems analysis and design. A comparison is drawn between this approach and classic system analysis and design. The benefits and problems with this modern approach are also considered.

This modern approach has the potential to provide systems solutions to many problems, but could lead to a different problem, a *new software crisis* – using modern hardware and software with ad hoc analysis and design. These system solutions are developed very quickly and cheaply, but many times without consideration for the users or proper data handling methods.

Introduction

Producing reliable, robust, cost-effective software systems has always been a problem for the computing industry. In 1968 the term *software crisis* was coined by the NATO Software Engineering Conference [1] for the myriad of problems in the development of quality software. In those days, computers were less common and more expensive; there were few programmers and analysts; every system was developed from "scratch." Even small projects took months to develop and were thus expensive undertakings.

Computer Science (CS) and Computer Information Systems (CIS) programs have responded to the problem in a number of ways. Early programming courses taught not only the particular language, but also how to develop quality programs; sometimes called program engineering. Both programs provided courses in Systems Analysis and Design and later in Software Engineering.

Students were taught a systematic approach to the development of computer systems. They learned the lifecycle of a system and how to approach the analysis, design, and implementation to produce reliable and robust software [2,3]. In many programs, students were expected to demonstrate their competence through a project courses before they graduated and joined an IT department and at the time, the only people developing systems were the IT departments.

However, computing has changed since 1968. Computers are now cheap and plentiful. Modern programming languages have been developed, providing a number of improvements, most notably ease in developing a graphical user interface.

A useful development, for creating software systems, has been the creation of *application utilities* – word processors, spreadsheets, data bases, etc. Now many "programming" solutions can be developed primarily by customizing these packages.

Furthermore, the combination of inexpensive hardware and application utilities has allowed the end-user to develop their own unique software applications. Users no longer need to wait for an IT department or consultant. Those who are interested may develop their own. Development has moved from the hands of the CS/CIS, IT expert into the general population.

The following is a study of one particular system developed in this new style: a technology savvy user implementing a system with application utilities. The chosen problem is simple and straight forward; one that could be approached by a sophomore/junior CS/CIS major. Names and identities have not been used to ensure privacy.

Case Study: Problem

A local public school system provides special education services to preschool children by sending itinerant teachers to service these children at various preschool and day care facilities. These teachers must periodically (yearly) conduct "case conferences," which provide progress reports using multiple state and local mandated forms. These multi-copy forms are preprinted, filled out by hand by the teacher, and then distributed to the parents and the student's permanent file. The file is intended to follow the student as he/she enters the public school system.

The problem, one faced in many systems, is the massive paperwork. The itinerant teacher must fill out several forms for each student. These forms provide spaces for the teachers hand written information, however information is duplicated on the various forms and the process is time consuming. Current estimates are approximately one half hour per student with 60 or more students per teacher. In addition, the teachers must find time in their already busy schedule to complete the forms.

The teachers and administration felt this was an ideal situation for an improved system. The goals of the system were to be (1) reduced time to prepare the forms for a case conference, (2) reduce re-entry of redundant information, and (3) provide more accurate records.

Case Study: Solution

The traditional approach [2] would be to turn the problem over to the IT department or an outside firm to analyze, design, and implement and appropriate solution. This would take time and be costly, and school systems rarely have extra funds.

The special education department had their own technology specialist. This person was trained as a special education teacher, but demonstrated an interest and aptitude in technology and was moved into an administrative position supporting technology in the special education area. This person had an advantage over a normal IT specialist: as a member of the department they were already familiar with the forms and the procedures. Therefore they initially had a better understanding of the problem than an IT designer.

However, there was no more real analysis. The itinerant teachers were not consulted either as a group or individually; i.e. there was no discussion with the end-users. The manual system was taken as a model and replaced with a computerized version.

The solution implemented was to create files that were word processing (Microsoft Word) templates for each form that must be completed. These files were distributed to the teachers, on a floppy or compact disk. The individual teacher then, knowing the forms needed, would selected the appropriate files for a particular student, filled in the appropriate fields in the template, and saved the updated copies of the files in a machine

readable form, usually on floppy disk. The final step was the case conference where the itinerant teacher(s), and other teachers or specialists, met with the child's parents.

During the case conference, the teacher may need to add additional comments to the forms, so the files were again edited and then the appropriate number of hard copies was printed for signatures. The files were to be kept in machine readable form (floppy disk) for each student. Thus a computerized record was formed for each student to be carried on through the educational process.

Analysis of Case Study

The resulting system had several advantages over the previous manual system. The material was no longer hand written. This meant no chance of going back to a form at a later date and finding that information was unreadable. The material was kept in a machine readable form, so in theory, it could be reused in subsequent years.

A change in the design of the forms could be implemented quickly, with minimal cost. The file containing the particular form could be edited, or replaced if there were major changes, and redistributed for the cost of a disk.

The system was developed quickly and cheaply. The only personnel involved were already part of the staff. The system was implemented using existing equipment and software.

Unfortunately, there are a number of defects in the resulting system. The teachers involved were itinerant and many were not provided their own portable computers or printers. They were expected to find a computer at the facility they were serving or use the desk-top systems at their home office, which they visited only once or twice a week. This was a major problem when, during a case conference, updates were needed to the forms. The teacher had to find a computer to make the changes, print the updates, and then return to the conference. Many times a second meeting had to be scheduled to sign the updated forms.

There was no training planned or provided for the end-users, the itinerant teachers. Many of the teachers had limited, or minimal, computer skills. They were expected to not only understand the working of the word processor, but also expected to be able to load a file from one location device, edit, then save it on another device. Furthermore, since they were itinerant, they may have to use several different computer systems. There was no part of the system designed to handle backing up the critical data stored on an unreliable floppy disk. There was no plan for error recovery.

The new system had three goals; the first was to reduce the time to prepare the forms. In fact, it took longer to create the forms for an individual student! The estimated time increased from half an hour for the hand written forms to about an hour for the word processing system. Typing speed accounts for a portion of the delay, but the new system

required opening multiple files and saving them on different media. While this is not a difficult task, it may be time consuming and stressful for a user who was uncomfortable with computers to begin with. Furthermore, there was no training to do this and no systematic approach to renaming and organizing the new files.

The second goal was to reduce the re-entry of redundant data. This problem was not even addressed. The computerized forms were identical to the paper forms and therefore required all the re-entry the manual system required. The teachers had to re-enter data - such as name, birth data, etc. – repeatedly on the separate forms. This presented the same problems that the manual hand written system had. Furthermore, in the manual system the teacher could lay out the forms and copy the redundant information from one form to the others, thus ensuring some consistency. In the computerized system, teacher worked on one form at a time and except for the most technologically savvy, could not compare the redundant data for consistency.

The third goal was more accurate records. It does not appear that the computerized system provided any advantage over the manual system, although it appears to be no worse than the manual system.

As a final observation: the newer system should have taken advantage of the potential to reduce data entry errors. The new system by default took advantage of the word processor's spell checking. However, additional information could have been checked. Dates and ages could have been checked for reasonableness and consistency. Scores on standardized tests could be checked for validity and consistency.

Conclusions

It is fairly obvious that the designer in this example did not follow a traditional analysis and design approach. That, in itself, does not mean the approach is wrong. It illustrates the way many systems are being developed. Even well-planned, well-staffed projects can fail [4].

Whether we, as computer professionals, like it or not, this is the wave of the future. Sales people are developing their own customer databases. Doctors are setting up their own networks and developing customized applications. Any technologically savvy individual can now create a new system. When such a system is used only by the individual, then any effects of the system only involve that individual. However, if the system affects others, directly or indirectly, then we may hold the system to a higher standard. We don't want inaccurate patient records in a doctor's office and we don't want inaccurate records in student records.

In the case study, the designer was a knowledgeable user and therefore understood the problem area. The designer didn't need to consult the users to understand how the forms were being created and used. However, this also meant that the other users had no input

into the design process; they had no chance to voice concerns or evaluate proposed solutions.

In the case study, the designer used the software product (word processing) that they knew best. While any designer does that; most professional systems designers have had a reasonably wide set of training and experiences. This designer was comfortable with word processing and little else. Thus, they didn't explore all the options available. A simple change to using a database package, like Microsoft Access, could have produced a much more useful system; one in which the teachers use one standard "form" for input of all relevant information and then produce the necessary reports. However, if the designer has had no experience with a database package then they have no way of knowing the option exists.

As noted before, the resulting system suffered from a number of other deficiencies. There was no method of error handling; there were no procedures for backups and failure; in fact, there were no procedures at all; the system provided no attempt at data consistency.

We are entering a new *software crisis*. The original crisis was concerned with the development, by professionals, of reliable and robust software in a cost-effective manner. The new crisis does not deal with cost, since the investment in a computer system is minimal. Rather the crisis is with the production of reliable, robust software systems by anyone. More and more software, like the example case study, is being created by individuals whose only credentials are an interest in technology.

References

[1] H. C. Lucas, Jr., *The Analysis, Design, and Implementation of Information Systems*, Fourth Edition, Mitchell McGraw-Hill, New York, 1992

[2] P. Naur, B. Randall, and J. N. Buxton (Editors), *Software Engineering: Concepts and Techniques: Proceedings of the NATO Conferences*, Petrocelli-Charter, New York, 1976.

[3] S. R. Schach , *Object-Oriented and Classical Software Engineering*, McGraw Hill, Boston, 2005

[4] E. Oz, When Professional Standards are Lax, The CONFIRM Failure and its Lessons, Communications of the ACM 37, 10 (October, 1994)