



Practicing implementation of a large software system

S. Guirguis, M. El-Raey, E. Korany, S. Yehia

*Institute of Graduate Studies & Research, Alexandria
University, Alexandria, Egypt*

ABSTRACT

In this paper the problem of designing and implementing a large software system is addressed. Several considerations have been taken into account, namely : user friendliness, reliability, modifiability, maintainability and security. The case study was to model and automate the student management activities in Alexandria University, where the system has been applied to the student management section in the Faculty of Law as a model with characteristics general enough to be applicable to all faculties since the bylaws governing their activities are almost the same.

The paper describes the use of CASE tools to reduce the software implementation life cycle along with ensuring sufficient software quality.

One of the major topics which are most time consuming in software implementation cycle, is the development of data management routines. In this work, all data entry and management routines have been designed in an abstract and unified manner, where their functions are externally controlled via a set of script files. This allows for tailoring an unlimited number of applications without actually writing any programs. Also, the produced reports are intelligently assembled from another set of script files. This approach allowed for software reliability, maintainability and modifiability since the actual routines which are executed have been written in pure code and thoroughly tested through application to all possible data types and formats to be used throughout the system.

Since the system has to operate in a multi-user environment, several security levels have been introduced to avoid any expected misuses or conflicts

362 Software Quality Management

during operation. Bearing in mind that the users are the employees of the student management - with little or no experience in computing - the system has been designed to be user-friendly and self-contained.

INTRODUCTION

Computer Aided Software Engineering concepts and tools have been paid a lot of attention and interest in recent years because of the increasing demand on software developed for virtually all aspects of life. The long software life cycle has been and still is under investigation by several researchers. Several aspects of it need to be automated to shorten the duration of each phase and to reduce potential errors arising from different analyses and design styles and from the lack of a common vocabulary for communication between users and system developers.

For instance, the requirements specification would follow a certain formal method, understandable for both the user and the system developer. This would result in early elimination of possible conflicts and misinterpretations of the requirements by both contracting parties. Also, instead of building software systems nearly from scratch, the employment of repositories of reusable software components of proven quality would result in shortening the software development life cycle and additionally come up with a reliable package. Software testing and performance evaluation can then be accomplished using test data patterns analogous to those utilized in hardware trouble shooting.

Buckley and Poston [1] presented major outlines of methods and topics to be addressed for software quality assurance. They have restated the ANSI/IEEE standard for software quality assurance to be: a planned and systematic pattern of all actions necessary to provide adequate confidence that the item or project conforms to established technical requirements. They also reported the acceptance of two other definitions which are: (a) all those planned or systematic actions necessary to provide adequate confidence that a product or service will satisfy given needs, or (b) all those planned and systematic actions necessary to provide adequate confidence that an item or a facility will perform satisfactorily in service. The term *quality* has been defined to be: the degree to which software conforms to quality criteria which includes: economy, integrity, documentation, understandability, flexibility, interoperability, modularity, correctness, reliability, modifiability,

validity, generality, testability, reusability, resilience, usability, clarity, maintainability, portability and efficiency.

Requirements specification by formal methods have been investigated by several authors. Wing [2] considers a method formal if it has a sound mathematical basis. She defined a formal specification language to be a triple $\langle Syn, Sem, Sat \rangle$ where Syn and Sem are sets and $Sat \subseteq Syn \times Sem$ is a relation between them. The presented concept can be broadened by application to procedures rather than single instructions. This may result in a more general use and application by non-programmers. Roman [3] discussed the ways of expressing requirements where they fall into two general categories: functional and constraints. He showed that functional requirements involve modeling the relevant internal states and behavior of both the component and its environment. He also showed that constraints may involve: interface, performance, operating, life-cycle, economic and political constraints.

Potter and Trueblood [4] addressed a very important aspect which is the director of any data processing system and that's the data model. They presented three approaches to data modeling, namely: traditional, semantic and hyper-semantic. They discussed the three traditionally well known relational, hierarchical and network models. They concluded that traditional models are basically record-oriented models while semantic models are user-oriented ones since they exhibit more modeling flexibility without being constrained by a preset implementation structure. They introduced the term hyper-semantic data models which aim at capturing more of a given application meaning. Such data models focus on capturing the objects, operations, relationships and knowledge associated with an application.

The next step is to design and implement the procedures and functions of the software system. Hoare [5] gave an overview of some formal methods for program design. He based his ideas on the similarity between the design of small and large programs in that they both require a variety of formal methods and notations related by mathematical reasoning.

In large software systems development it is likely to adopt similar routines to perform a lot of similar functions. Obviously this requires the design of a set of universally usable routines to be repeatedly applied wherever needed. The term *universal* here



364 Software Quality Management

must not be taken literally because of the unlimited number of possible applications but it rather means conforming to the constraints and requirements of the currently developed system. Caldiera and Basili [6] discussed the software metrics to be used to identify and qualify reusable software components.

Provided that the user has joined the system developer during the requirements specification phase and approved it, it is likely that the system will perform towards user satisfaction where this is considered the ultimate target.

PROBLEM DEFINITION

It is required to automate the manual activities of a students' management section in a given faculty of Alexandria University. Since these activities are numerous, it is rather difficult to define them all in this paper. We would rather give a summary of the key activities in the following section, mentioned only by name without going into the full details of each activity.

For undergraduates, these activities include at least: student enrollment, study subjects determination, record tracking, examinations, scoring, transfer to higher grades, graduation, certificate issuing and graduate record tracking. For Diploma students, the activities also include management of the study in several specialized areas. For M.Sc. and Ph.D. students, the activities include keeping track of the supervisors committee decisions as well as the oral exam committee decisions. Other supplementary activities include statistics, honorary awards and scholarships, student transfer between similar colleges, fired students, modifications in the study subjects and courses and even expected changes in the university bylaws. It has been decided that there are at least 33 activities that need to be automated.

Each of the above mentioned activities is already assigned to a number of employees within a given faculty where there are usually major problems in the integration of the tasks of all employees, e.g. information on a given case or a student requires to be assembled from a number of records held by employees with various operating styles and may be using different formats. This has been always a time and labor power consuming process. Therefore, it has been decided to study the possibilities to automate most of the routine work done by a given students' management section as a step towards the



implementation of office automation technology within faculties of Alexandria University, as well as to investigate application of proposed CASE tools to a large scale problem.

Other activities which can not be modeled are those treated as special cases. These are subjected to faculty meetings and discussed individually until a decision is taken by voting. Accordingly, the data base is updated by direct intervention.

The faculty of law, Alexandria University, was in a transitional state towards automation of the students' management section and therefore it was a perfect candidate for applying the proposed study.

AVAILABLE HARDWARE

The system has been designed to be portable and therefore two versions have been installed and run on two hosts with two different hardware configurations and operating systems.

The first host consists of an IBM PS/2 Model 80 microcomputer with 2 MB RAM, a console, a 110 MB Hard disk for booting up using multi-user XENIX operating system, a 300 MB hard disk for programs and data, a tape streamer, a 1.44 MB diskette drive and a printer. The system is equipped with a serial port allowing for direct connection of up to 8 terminals where 5 of them are already installed. The terminals are intelligently Arabized through an internal firmware.

The second host is an IBM PC/AT microcomputer with 1MB RAM, 40 MB Hard disk, 1.2 MB floppy drive, a printer and a single user DOS operating system. This system is software Arabized.

SOFTWARE REQUIREMENTS SPECIFICATION

Following are the major outlines of the general software requirements specification:

- . The system will be operated by the employees of the students' management section with little or no experience in computing and therefore must be user friendly.
- . These employees may be trained on a number of systems and may be assigned to operate different ones upon management request. This situation may occur due to a lack in staff at specific periods for several

366 Software Quality Management

reasons, e.g. examinations control essential paper work, which requires most of the labor power of any faculty.

- . Report generation must be flexible enough to allow for the different styles and formats as expected to be existing between faculties. This requires that the user must be able to design his own reports with minimum effort.

- . Search in a specific data base must be enhanced by the use of a suitable filtration scheme.

- . Help with clear messages must be given where appropriate to aid the user while making decisions.

- . Errors must be trapped whenever and wherever possible.

- . Since the data items are enormous, easy to use and intelligent data entry screens are to be employed throughout the system.

- . The system is to model the actual activities of the students' management section and ruled by the University of Alexandria bylaws.

- . It must be easy to modify the system flow according to any anticipated changes in the university bylaws.

- . The system must be capable of handling both Arabic and English data management and report generation.

- . Due to the system nature, a hierarchical security system must be employed and equipped with the necessary tools for tailoring the operating conditions of both the hardware and software systems.

It is worth mentioning that the above requirements specification does not follow any particular order of importance.

DESIGN PHASE

The software system design phase has been divided into five sub-phases where each one deals with a specific class of routines, namely: security and setup, data entry and management, utility, application-specific and supplementary routines.

Security routines establish the usage level of the system for a given user. The system manager is responsible for configuring his system utilization by selecting which user is going to do what. A user is given a password which determines whether he will: use all or a specific terminal, have access to a given printer, have access to a given database, have access to a specific system operation or even an option within an operation. Then, a setup routine is employed to set the environment parameters and global variables.

Data entry and management is considered the most vital subsystem since it consumes most of the programming effort as well as the run time in a data storage and retrieval information system as the one under consideration. Therefore the design of such subsystem required a careful decision whether to design a data entry and management routine for each file in the data base, simply because they have different structures and applications, or to design a universal set of routines with suitable control parameters to suit the entire system utilization. The latter option has been accepted in view of the enormous number of data base files to be manipulated as well as it seemed appropriate to employ CASE concepts at this particular stage. The design criteria of such subsystem will be focused while presenting the implementation phase. In fact this was the first subsystem to be implemented throughout the software life cycle since its gained characteristics have been employed all over the design and implementation phases of the other subsystems in addition to its extensive use in building them too.

Utility routines have been designed to act as building blocks of larger procedures and programs. For instance, routines to trap errors and display appropriate messages have been designed. Also, in a multi-user system, it is expected to have access collisions. Therefore, all data base update operations are performed via a special routine whose function is to detect and avoid these collisions with no user intervention as far as possible. Other routines such as those used to delay a process, give a warning message, initialize the printer for operation, Arabize a line of text for compatibility between a terminal and printer, etc., have also been designed.

Application-specific routines have been built according to the special nature of the problem. For instance a routine which transfers students between grades on success, keeps a student in the current academic year data base in case of failure, performs graduation procedure, or fires a failing student after exhausting all given chances, is an application-specific type of routines. The most important fact about these routines is that they are mainly designed and built based on the same ideas and concepts established by the data entry and management routines and in effect they are mainly assembled from the previously built and checked procedures. That is, improving, updating or even changing a procedure or a routine would in essence map the performance changes to all related tasks. This enhances the idea of reusable software components and consequently ensures



368 Software Quality Management

that if a given piece of software is well designed and thoroughly checked would become a solid break in quality assurance.

Supplementary routines are actually smaller versions of the above subsystems to aid manipulating supplementary data such as : governorates, graduation grade names, student status cases, etc.

DATA CLASSIFICATION

Data items have been classified into two main categories:

. *Static data* : those items which are never or rarely changing, e.g. student name, address, sponsor, ... etc.

. *Dynamic data*: those items which are always amended to a data base according to the occurrence of certain situations, e.g. student enrollment suspension and resumption due to medical reasons for instance, excuses presented by a student for not attending an exam, ... etc.

According to the previous data classification two sets of data entry and management routines have been designed to deal with each class. The one concerned with dynamic data has essentially the same characteristics of that concerned with static data except that it is equipped with an extra search scheme to locate key items in the static data base for which dynamic data are to be manipulated.

IMPLEMENTATION PHASE

This section gives an insight view of the concepts, techniques and tools used to develop the data entry and management subsystem, since it is the heart of the data storage and retrieval information system under investigation. This model subsystem is sufficient to present the concepts and techniques used throughout the design and implementation phases. All other subsystems are built following the same principals and techniques.

Data entry and management subsystem is menu driven with all menus and control mechanisms are externally assembled and stored as *script* files. The routines are designed bearing in mind that they will be reused wherever possible and therefore written in pure code. All the required control is passed through a set of script files describing what data base file(s) to be used during the execution and how.

This menu driven system is easily configurable by simply updating the *menu* description files in such a manner that, if required, an entirely different system flow can be externally designed and controlled.

Other controlling script files are given parameter names : *f*, *fdx*, *fky*, *fhd* and *frp*. The parameter *f* determines the data base file on which to perform manipulating operations, *fdx* is its index file name - if applicable-, *fky* is its key description to be used in search and insert operations, *fhd* is a header file describing the screen format and functions to be used for data manipulation and *frp* is a script file containing a set of pre-designed report forms represented as a set of descriptors to be followed during report generation. At run time these script files are interpreted to determine the strategies for handling data base files.

The following flow diagram (Figure (1)) illustrates the operations of data entry, update, delete and report generation. It is worth noting that authorization setting may cause certain options to be skipped or inhibited for some users. Also, preparing a printable file means performing all necessary translations to assemble a terminal compatible image; this is essential for handling Arabic characters and numerals and for selecting printing fonts and sizes.

The script file *fhd* is employed extensively in the data entry and management subsystem as well as in other subsystems. Its role is to map the actual data fields to be manipulated and their attributes to the actual functions to be performed. For example it describes the data entry screen format and the titles and subtitles to be used. It is used in constructing filtration screens and design of search criteria. It enables or disables certain data entry features to suit the data type to be handled as described below.



370 Software Quality Management



Figure (1) Flow diagram of data entry and management subsystem

Data manipulation is simplified for the user by allowing data fields to fall into one of possible three main categories:

1. plain text, numeric and date fields, where an entry is essentially filled using the keyboard, e.g. a student name, address, marks, birth date,... etc. User keying in is unavoidable with this data type.
2. single option fields, in which the user navigates through a limited set of *possible* selections using the arrow keys until an answer is chosen, e.g. selecting a student grade (*Distinction, Very Good, ... etc.*). This serves several advantages:
 - a. it saves time of entry since the actual code will be internally handled and is *transparent* to the user. This is very important if the designer decides to add other selections, exchange certain codes or change the coding scheme completely.
 - b. it eliminates possible human errors of choosing other illegal codes which may be not recognizable by the rest of the system or even may cause unavoidable conflicts in the application-specific type routines.
3. coded fields, where possible responses are stored separately in a key indexed file and accessed through a code field, e.g. names of governorates. The user is asked to give a code and the system interrogates him until the answer is confirmed. In this field type it is possible for the user to encounter a new entry to be added to the list, and therefore *some* users may be given the authority to interactively add new entries.

When interpreted, the script file *fhd* would designate the appropriate function or routine to be applied to a given data field. This gives another dimension to flexibility and modifiability, since the effort to change a data field format and handling scheme would only be done once in *fhd* file and immediately mapped to the rest of the system, wherever it is used.

The script file *frp* contains a set of predesigned report forms where the user is allowed to use,

372 Software Quality Management

add or manipulate them interactively through filtration screens.

A filtration screen is invoked upon user's request and a search criteria is internally constructed for the user while he fills a set of data fields conforming to the previously described categories and selects the appropriate ones of the possible relational operators (>, <, =, ... etc.) using the arrow keys, e.g. suppose he selects:

```
Student name = Sandra
Birth date   > 1970.02.05
```

then the internal search criterion will search for a student with a first name 'Sandra' and a birth date later than February 5, 1970. It is worth noting that throughout all data manipulation and filtration processes the user is not subjected to the physical field names as internally stored.

In Figure(1) all similar tasks are actually performed through one reusable routine or procedure. This gives a chief flexibility in modifying, enhancing, or even replacing a given routine without affecting the operations of the others. For instance, to add a new data item the system would insert a blank record in its proper place and then calls the same update routines to carry on the operation. This means that modifying one of the update routines would affect both data entry, search and update operations simultaneously.

CONCLUSION

In this paper the problem of designing and implementing a large software system has been investigated. The use of CASE concepts added a great deal of flexibility in the system design and implementation phases and therefore resulted in favorable enhancements both for the user and for the developer. The time span of the software life cycle was noticeably shorter than that if we would have used traditional system analysis and design methodologies especially in view of an even larger to specify problem like the one presented above.

One of the greatest merits of using such concepts is that the software design is *general* enough to be adapted to other similar systems without actually doing any programming. Also, user satisfaction seemed to be achievable through the employment of user friendly set of routines controlled by script



files that can be easily modified and maintained. Also, system portability became a less worrying issue. The system exhibited greater reliability due to the nature of pure code routines, which could be individually tested against all possible sorts of data to manipulate in relatively smaller experiments rather than testing the overall system behavior with huge sets of data.

ACKNOWLEDGMENT

The authors wish to acknowledge the cooperation of Prof. Dr. M.Z. Abou-Amer and the staff of the students' management sections in both Faculty of Law and Institute of Graduate Studies & Research, Alexandria University for providing all the facilities and data to conduct this study.

REFERENCES

1. Buckley, F.J. and Poston, R. 'Software Quality Assurance' *IEEE Trans. Software Eng.*, VOL. SE-10, NO. 1, pp. 36-41, January 1984.
2. Wing, J.M. 'A Specifier's Introduction to Formal Method' *Computer*, pp. 8-24., Sept. 1990.
3. Roman, G.C. 'A Taxonomy of Current Issues in Requirements Engineering' *Computer*, pp. 14-21, Apr. 1985.
4. Potter, W.D. and Trueblood, R.P. 'Traditional, Semantic, and Hyper-Semantic Approaches to Data Modeling' *Computer*, pp. 53-63, June 1988.
5. Hoare, C.A.R. 'An Overview of Some Formal Methods for Program Design' *Computer*, pp. 85-91, Sept. 1987.
6. Caldiera, G. and Basili, R. 'Identifying and Qualifying Reusable Software Components' *Computer*, pp. 61-70, Feb. 1991.