Mansoura Engineering Journal

Volume 20 | Issue 4 Article 4

12-1-2021

A Framework for Improving Software Reusability within Quality Constraints.

Shawkat Guirguis
Institute of Graduate Studies & Research., Alexandria University., Egypt.

Mansoura Engineering Journal (MEJ), Vol.20, No.4, Dec.1995 E-24

A framework for improving software reusability within quality constraints

إطار عمل لتحسين إعادة إستخدام البرمجيات فى حدود متطلبات الجودة

Guirguis S.K.

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

ملخص

يتم فى هذا البحث دراسة سبعة من أهم النقاط المؤثرة فى عمليات تطوير أنظمة البرامج و تبحث هذه النقاط فى موضوعين أساسيين هما تحسين إنتاجية البرامج و رفع جودة المنتج النهائى.

بالنسبة إلى إنتاجية البرامج فقد تم إقتراح طريقة تعتمد على إستخدام السكريبت و التى تبنى بإشتراك كلا من محلل النظم و المستخدم النهائى و تستخدم فى تحويل متطلبات المستخدم إلى شفرة برامج أوتوماتيكيا وتخدم هذه الطريقة فى إنتاج نماذج أولية سريعة متضمنة إشتراك المستخدم النهائى فى التصور لضمان رضائه عن المنتج النهائى حتى قبل إستكمال التنفيذ.

أما بالنسبة إلى موضوع الجودة فقد تم عمل مكتبة برامج من مستويين: الأول يحتوى على مفسرات السكريبت و هو مستقل عن المستوى الثانى و الذى يحتوى على التنفيذ الفعلى للمكونات المؤدية.

هذا و يعرض البحث خمسة تطبيقات فى مجالات مختلفة تم تطويرها بواسطة النظام المقترح للدراسة جدواه.

ABSTRACT

In this paper seven of the most important issues in software systems development have been investigated. These issues address two aspects, namely: software development productivity and the final product overall quality.

For the productivity aspect, an approach has been devised relying on using scripts. Built as a result of an analyst-user collaboration, these scripts, upon interpretation, transform given user requirements into code. This prototyping approach, in addition to being rapid, involves the user in order to ensure his overall satisfaction with the final product.

For the quality aspect, a two-layered repository of thoroughly tested and verified reusable software components has been built. The first layer contains the kernels to interpret the scripts. This is independent of the physical implementation of the functional components in the second layer. Using the repository, the core data manipulations of virtually any application can be automatically generated.

Five applications covering five different domains have been designed and implemented to investigate the merits of the above described framework.

1. INTRODUCTION

It is noticeable that one of the most important goals of software systems developers is to be able to directly translate given user requirements into computer code. This direct translation aims at minimizing the development effort and time as well as achieving maximum user satisfaction. In this paper we are concerned with some of the key techniques and attributes that must be considered to fulfill the above mentioned goal, namely:

. Standardized user interface.
. Unification of data manipulation methods.
. Rapid software prototyping.
. Compilation of repositories of reusable software components.
. Software system maintainability and modifiability.
. Final product reliability.

Due to the nature of query/response and storage/retrieval information systems, the above described issues are likely to be directly realized to some extent. Other systems such as data acquisition are mostly domain dependent and therefore special coding is mandatory.

1.1. User interface

Design of the user interface is getting more and more stabilized through the implementation of well recognized and widely used techniques, e.g. pop-up and pull-down menus, context sensitive help screens, hot keys, windows ... etc. (See for example [1]). Employing such techniques ensures, to some extent, familiarity of the user with navigating through the system's various functions.

1.2. Data manipulations

Data manipulations generally constitute the greatest proportion of the code size and run time of many info mation systems [2]. For this consideration, recent programming environments include various built-in data handling mechanisms. However, some programming effort is generally required to customize data entry forms that include application-specific manipulations.

Although linking and data processing routines are special cases that must be handled separately, in the current proposed approach, these have been treated as modules that could be added during any stage of the system development. The latter

point is rather facilitated due to the extensive use of *scripts* that internally characterize the proper data type transformations, leaving the developer with only specifying the required data flows and specific algorithms. This attempt is likely to satisfy the three key features of an automatic programming system under data manipulations context [3].

1.3. Rapid software prototyping

In its early days form, it would have taken the developer quite some time to design the data entry layouts and the reporting formats to get the user's approval on a proposed software system. The prototype's essential role is therefore to assess the merits as well as the pitfalls of a proposed software system before actually going into the more complicated processes and details of designing and implementing the full system. These have evolved chiefly due to:

. the timing constraint normally imposed over software systems development, and
. the occasionally unclear software requirements specification.

The latter point is partially attributed to the user who is not acquainted with expressing his requirements in technically acceptable terms, as well as it is attributed to the analyst himself, who is sometimes not aware of the user's real problem. Several attempts are being made in formalizing users' requirements in acceptable technical terms for the benefit of both contracting parties (See for example [4,5]). Rapid software prototyping can aid in at least the following phases:

. getting the user to approve (or disapprove) a proposed software system in the earliest stage possible,
. if the system is disapproved, its design can be modified and proposed again until eventually its framework is accepted, and
. upon approval, the user is enabled to start feeding-in his data sets while the rest of the software system is still being developed.

This approach has a profound impact on reducing the software development life-cycle. Quality of the final product, however, must be maintained [6,7].

1.4. Software reusability

Another very important issue to be considered is that whether the development process would start from scratch or rely on a repository of thoroughly checked reusable software components [8]. These components are verified and validated against test data patterns in a fashion similar to trouble-shooting in hardware component testing.

1.5. Maintainability and modifiablity

In the current context, a software system is considered maintainable when each of its components can be isolated and verified. It is considered modifiable when all alterations can be made at the system definition level rather than the component or the instruction level. One of the aims of structured analysis and design is to enable maintenance and modification of software systems in an easy manner. Techniques employed in Computer Aided Software Engineering are therefore appreciable (See for example [9,10]).

1.6. Reliability

It is generally important to be able to quantify reliability in order to judge the overall performance of a given software system. However, comparing two software systems is not a straight forward process. Lockhart [11] pointed that there is no such thing as a general purpose model of system reliability. Macro and Buxton [12] have associated reliability with compliance and modifiability. They have noticed that modifiability, and consequently reliability, are left with no real metrics. They have also concluded that the already known measures such as MTBF and MTTR, although successfully applied to hardware systems, are not adequate for software systems reliability evaluation.

On the road to quantify software reliability, Geist *et al.* [13] have been interested in N version, real-time software reliability estimate. In their study, they have used a graphical Petri net simulation tool that included arbitrary firing time distribution and correlated firing. The model has been used to estimate MTTF using mutation analysis applied to a simple model.

In this paper we introduce a performance index (θ), defined as the ratio between the probability of failure of a systematically built software system and its worst case value if the system is built from scratch. This approach has been used to assess the gained performance when relying on repositories of reusable software components.

2. AIM OF THE WORK

This paper suggests a framework for software systems development bearing in mind the above mentioned facts. Other complementary features that have been taken into consideration throughout the development process are also presented and discussed.

3. CASE STUDIES

In order to assess the merits of the proposed framework, the implemented software has been applied to five case studies. It has been used to generate five applications in five different domains:

1. Scientific: Questionnaire data manipulation and processing. Given the acronym (QUITE) standing for: Questionnaire Utilization in an Integrated Tabulation Environment.
2. Industrial: Machine Service Log and follow-up (MSL).
3. Commercial: Container Loading/Unloading Control (CLUC).
4. Managerial: Assets Control System (ACS).
5. Social: Social Development Organization providing aid to the needy (SDO).

4. IMPLEMENTATION

In this work a repository of software components has been built (Figure 1). It can be logically separated into two layers:

Layer 1:

Consists of the system kernels used to interpret the scripts. These scripts are designed to contain menu definitions and procedural flow. The system kernels are:

. Data manipulation program generation kernel
. Query formation kernel
. Report generation kernel

Layer 2:

Contains the lower level procedures, functions and macros to be expanded as appropriate by layer 1. It must be noted that the same data manipulation procedures and macros are re-used for query formation and report generation.

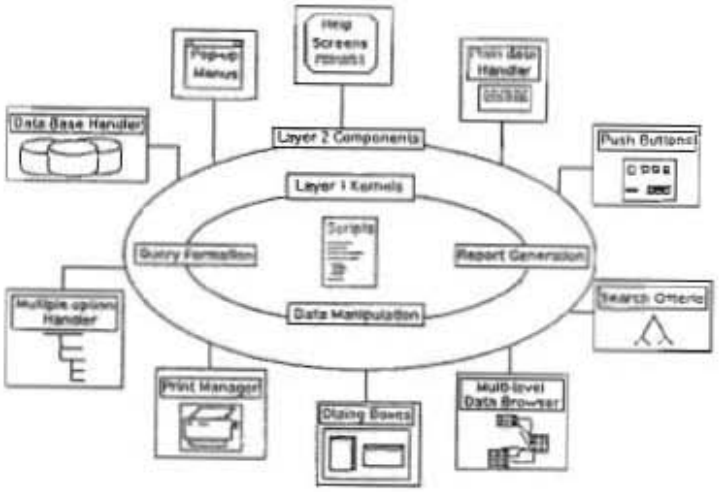

Figure 1. Logical overview of the proposed system

Layer 2 includes procedures to handle the possible data manipulation responses described in Table (I) given below. These procedures can handle plain, single option, multiple option and key entries.

Another feature has been added where a given response is designated a push button that activates a sub-process. This sub-process is typically a lower level data manipulation routine that handles transactions (or details) concerning the parent data base. This process is repetitive and logically unlimited. The system keeps track of the links between the parent, children and grandchildren data bases.

This layer separation in the repository guarantees that the mechanism of automatically generating programs is isolated from their actual physical implementation. This enhances the tailoring process to a great extent. For example: a user who requires to handle the employees' time sheet hourly rather than daily (e.g., in a typical payroll system) would need only some amendments to layer 2 that have no effect on layer 1.

4.1. User interface

The user interface is generated from the script that contains all menu definitions and their associated attributes. The interface employs standard techniques such as:

. Pop-up and pull-down menus.
. Push buttons
. Context sensitive help
. Hot keys

In addition, most data entry operations, other than plain entries, are performed by scrolling all possible responses on display. The user is asked to choose one. This speeds up data entry operation and eliminates human errors such as choosing erroneous codes.

4.2. Data manipulation

In order to handle various problem categories in almost a unified manner, common data manipulations must be recognized in terms of data types and possible user responses. Table (I) summarizes the most observed responses at the data entry level.

Table I. Most common responses to data entry procedures

Common response	Data type				
	String	Integer	Real	Date	Boolean
Plain entry/update	√	√	√	√	√
Single option		√			√
Multiple options	√				
Key	√	√	√	√	

Most industry standard software packages are well equipped with procedures to handle plain entries/updates. However, the other possible responses are generally left to the programmer to handle, especially for multiple option and key entries.

In the implemented system single options, multiple options and key responses are handled as follows:

a. Single option entries

These are characterized by a limited set of possible responses, where an index (or corresponding code) is actually manipulated and stored. In order to minimize the chances of occurrence of possible data entry errors in such coded fields and to speed up

data entry process, fields of type single option are handled such that: possible answers are displayed and scrolled against the user until one is selected. The index or code of the selected response is internally transferred to the data file instead of allowing the user to enter the code himself. This approach goes in harmony with the user interface that uses pull-down and pop-up menus.

b. Multiple option entries

Multiple option entries can accept compound items that are mainly used in searching and reporting. Possible answers in multiple option fields are popped-up as help screens for data entry. A single code composed of 0's or N's for not selected and digits indicating the rank of selected options is supplied by the user. Also, while editing search classes, queries of the form Y12XN for instance can be formed where :

Y: means first option is selected (no matter what's its rank)
1: means second option is ranked 1
2: means third option is ranked 2
X: means ignore fourth option
N: means fifth option is not selected

This gives a noticeable flexibility in updating and analyzing data of this particular nature.

c. Key entries

Key option entries prompt the user with an identification of the data item under consideration. When appropriate, authorized users are allowed to interactively add new entries.

4.3. Reusable software components

4.3.1. Query formation

Queries are built the same way data are entered where all features of data entry and update are available. In a QBE manner, templates (typically the same data entry screens) are used for building such queries. The only difference exhibited is in the use of relational operators to build search classes. All relational operators (> , = , ... etc.) are displayed as (GREATER THAN , IS , ... etc.) and scrolled on the data entry line for easy selection and to enhance readability. The system then generates criteria upon which queries are performed.

4.3.2. Report generation

Report generation uses the same concepts of queries. Reports are built with the classical attributes: title, subtitles, column contents, formats and filtration criteria. The other added features include:

. Field contents are automatically obtained from *related* data files to replace codes of single options, multiple options and keys. This feature enhances the produced report readability.

. Filtration criteria are pre-designated and stored, however, they can be interactively altered through the use of query formation screens. The modified report attributes together with a user selected report designation can then be stored for later use. In this process an indefinite number of reports can be interactively designed and stored. This gives another dimension to the produced system flexibility.

4.4. Prototyping

The use of scripts facilitated generation of all data manipulation parts of the software system to the furthest extent. For instance, table (II) highlights the attributes of a typical script that are required to generate data storage and retrieval programs. The same script is eventually used to interactively form queries. Similar scripts are used to generate reports and menus.

The above described scripts are interpreted by the system kernels to generate the core data manipulation procedures upon which a user approval can be obtained. Since reading a script is naturally easier than understanding code, this prototyping approach is expected to involve the user in the early requirements specification phase. Eventually, an approval can be obtained on scripts rather than relying on screen demonstrations.

Table II. Typical attributes of data storage/retrieval programs

* File(s)
  - Key attributes (e.g., single/compound, uniqueness..etc.)
  - Field(s)
    . Identification
    . Title
    . Screen/Window coordinates
    . Expected user response
      Plain entry/update
        Format and range
      Single option
        Set of feasible responses
      Multiple options
        Set of responses
      Key relating other complementary databases
    . Restrictions and validation criteria
* Record handling mechanism
  - Single record editing mode
  - Full screen editing mode (browse-like mode)
* Push buttons
  - Sub-processes to activate
    . parameter passing
  - Restrictions and validation

4.5. Maintainability

Any problem that may occur can be traced back to either the kernels of layer 1 or the components of layer 2. Any changes in these two layers are mapped directly to the whole system. Therefore, maintainability is assured through reviewing a limited set of procedures and routines rather than the detailed software system. After corrections are made to the procedures, the whole, or the affected part of the software system, can then be regenerated in a compilation-like process.

4.6. Modifiability

This is also assured through the use of scripts. Any modifications are done at the definition level by updating the limited set of scripts that characterize the software system. This process can be performed iteratively and an updated software system can be immediately re-generated, until the user is eventually satisfied.

4.7. Reliability

Reliability (R) is defined in this work as the probability that the system will not fail $(1 - P_f)$. This can be given by equation (1) where $P(fail|P_i)$ is the probability that the whole system will fail given that process (i) of the Ω processes has failed.

$$R = 1 - \sum_{1}^{\Omega} P(fail \mid P_i) \cdot P_i \qquad (1)$$

Equation (1) is valid for single processor machines since no more than one process can fail at a given time. A lower limit of reliability can be given by: $(1 - \Sigma P_i)$. When using reusable software components the value of Pi can be individually reduced through appropriate verification. Also, in modular design, the number of redundant components is reduced to its lowest limit. The net result is an increase in reliability.

Although thoroughly tested and verified, re-usable software components may fail against special cases. Let us assume that the upper bound of probability of failure of such components is given by P_0, and the number of procedures in these components is denoted by σ among the overall system procedures Ω. Let us also assume that there exists an upper bound P_a denoting the probability of failure of externally added components. It is likely that $P_0 << P_a$. Assuming that $P_a = \alpha.P_0$ where α is generally >1, probability of failure can be written as:

$$P_f = \sigma P_0 + (\Omega - \sigma) \alpha P_0 \qquad (2)$$

Let us define (θ) to be the ratio between P_f given in equation (2) and its expected value in case when the whole software system would be constructed from scratch. This is given by equation (3):

$$\theta = 1 - \frac{(\alpha - 1)}{\alpha} \cdot \frac{\sigma}{\Omega} \qquad (3)$$

Equation (3) has been applied to the five case studies. Obtained θ values are considered indicators to how much the probability of failure is generally decreased when heavily employing thoroughly tested reusable software components.

5. APPLICATIONS AND RESULTS

Following, is a brief review of applying the framework described above to five case studies. The case studies have been chosen to cover five different application domains as mentioned earlier.

Without loss of generality, only the first application will be described in some details to illustrate the ideas behind the proposed framework. In fact, it was the first application that gave us the motivation to outline the suggested software development framework.

Application I. *Questionnaire data manipulation and processing (QUITE):*

Problem definition: A user who requires to perform some questionnaire analysis would need to use an industry-standard software package to carry out the required statistical analyses. This may seem to be the most efficient way to perform the analyses, but this is not always the case, since most of the available ready-made packages require specific structures for the input data to be maintained [14,15]. Analyzing combined text, numeric, boolean and date types of data is not always possible, and if available the user often suffers from a considerable burden with regard to defining the data types and their handling schemes. This may request that the user be extremely experienced in the use of a certain statistical package and may also include some programming effort to be done. Often, data type transformation routines are employed to convert from questionnaires' internally archived data elements to suitable formats for input to a specific statistical package. It has been decided to tailor such a software system using the framework described earlier.

5.1. Design phase requirements

The main objectives of the system design to be realized are summarized as follows:

1. As data structuring may constitute a problem for a given user, he is not subjected to the internal archive structure of the questionnaire design. Also the design must fulfill the requirements of being easily maintainable and modifiable according to specific user needs. The user is only required to specify his data items in an abstracted manner and

the software system is to generate the appropriate formats. Only plain text entries require the user to supply their maximum expected lengths.

2. Required to automatically, or at least easily, generate the data entry and editing subsystem according to the questionnaire description given in step 1. This saves the effort required in an essential, but tedious, phase. Also, it must have the advantage of being self modifiable according to any alterations made in step 1.

3. Required to include techniques for handling various types of data, especially for combined analysis of multiple option entries and for performing the appropriate data transformations. These data type transformations must be accomplished internally without any user intervention.

4. Provide tools to allow the user to synthesize data filters used to extract study-specific subsets of data, or more widely to identify search classes. These tools are expected to utilize natural language specification and simple relational algebra.

5. Allow the user to easily design and produce reports of analyzed data in a self-documented manner to reduce the effort usually associated with producing reports for timely information distribution.

In short, the overall objective is to create an integrated environment for questionnaire data archiving and manipulation, starting from the design phase of a questionnaire form to the actual production of analyzed data in an easily readable format.

It is worth noting that the above requirements are generally applicable to a wide class of information systems other than this specific example. This fact has been taken care of during the design and implementation phases. This issue resulted in an attempt to construct a repository of reusable software components. It also stimulated the need to identify which components are to be automatically generated and which are not. The rest of the case study illustrates this point.

5.2. System design approach

Design of QUITE software system is organized as follows:

I. Questionnaire build-up phase
II. Automatic generation of data manipulation subsystem

III. Analysis and tabulation phase, which consists of two sub-phases:
- III.1 Classification
- III.2 Application of analysis procedures with report generation facilities and automatically synthesized documentation.

Phase I, "Questionnaire build-up", is a user-friendly interface used to interrogate the user on the questionnaire entries, their attributes and, if applicable, their associated possible answers. The attributes of an entry may be either one of those described in table (III).

Table III. Data types used in the questionnaire analysis system

Data type	Description
Text	A plain text answer.
Integer	An integer number.
Real	A floating point number.
Date	A date expression.
Single option	Refers to an index value of 1 up to the maximum number of possible answers, *e.g. marital status.*
Multiple options	Where more than one answer are valid. In this type the user can weigh each answer with a number indicating its order, *e.g. preferred transportation facility.*
Key	Indicating a corresponding data item.

This data type classification can hold true for most information systems in use and therefore suitable handling mechanisms may be developed, validated and eventually reused whenever such data types are encountered.

Phase II, "Automatic generation of data manipulation subsystem", assembles the required data entry and editing procedures guided by the questionnaire description provided in phase I. It creates data files to hold collected data entries and customize a set of procedures to facilitate data entry and editing. Field names are coded internally and formatted to hold the corresponding data types without any user intervention to ensure ease of operation and to eliminate any possible errors typically encountered in manual archive structuring.

Phase III, "Analysis and tabulation", is then concerned with performing two main tasks:

1. It allows the user to filter some of his entered data to obtain subsets of the originally archived data if required. This data classification process is essential in specific studies.
2. It allows application of the analyzing procedures and synthesizes self-documented reports.

5.3. Characteristics of the user interface

The user interface has been designed bearing in mind that:

1. it must be user-friendly through the use of numerous help messages and pop-up menus for easy selection. Also, most data entries are being performed by selecting possible answers using only the arrow keys.
2. search criteria are being built using simple relational algebra where all operandi and operators are displayed in natural language rather than mathematical notation.
3. the user is able to customize the automatically generated reports by selecting whether he needs an automatically synthesized description of search classes, user defined captions, percentile calculations and their decimal digits, or a combination of all mentioned options.

To fulfill the above requirements of the user interface, a number of routines have been selected from the repository to minimize the clerical work required in data entry of hundreds of questionnaire forms with tens of entries. In case of data entry and/or update the user can select any questionnaire form and then navigate through its entries until one is selected for editing. The system keeps track of the data types under consideration and then calls the appropriate data manipulation procedures.

While specifying the required analyses, the user is asked to customize search classes. A user defined caption is entered for the class and then a series of conditions are edited, in natural language, to form the filtration criteria for the class.

5.4. Implementation overview

Figure (2) illustrates the user view of the proposed software system. The user interacts with the system at three stages:

1. requirements specification to enable the system to build-up the questionnaire form and its entries.
2. entry of collected forms using the automatically generated and/or assembled data handling procedures.
3. analyses specification through user defined search classes using natural language and simple relational algebra.

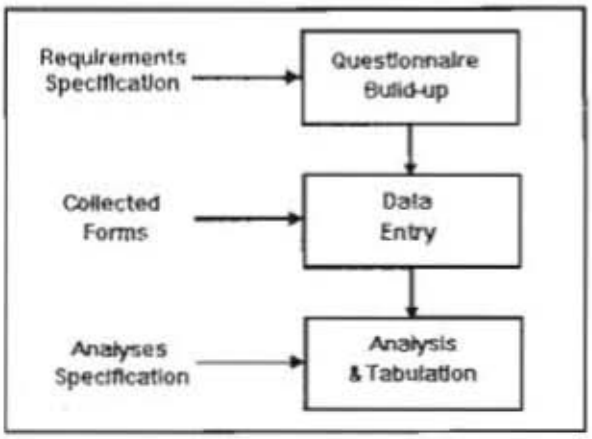

Figure 2. User view of QUITE software system

Figure (3) gives a schematic of the main components of QUITE software system.

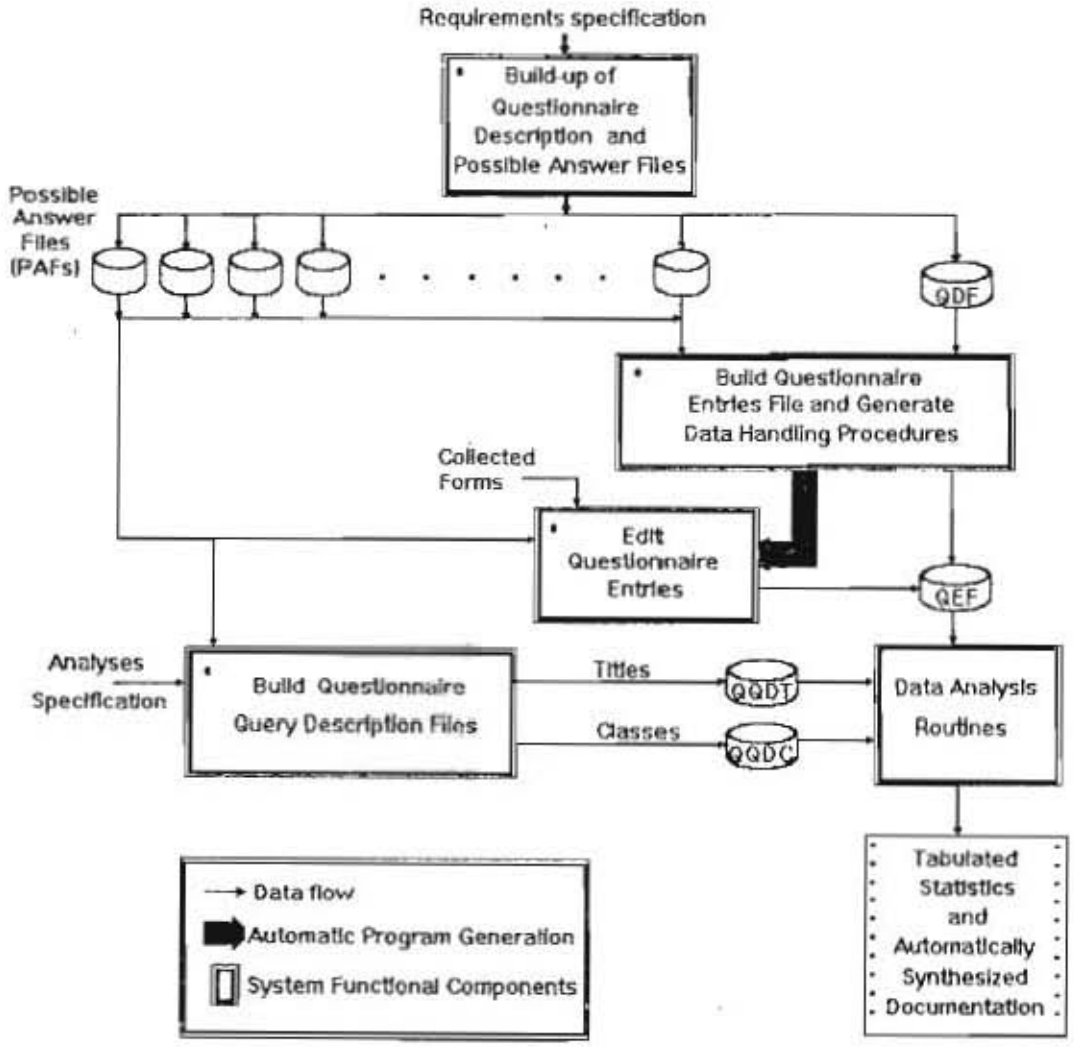

Figure 3. Main components of QUITE software system.
(Automatically generated or repository taken procedures)*

According to the requirements specification, the system produces a Questionnaire Description File (QDF) and a set of Possible Answer Files (PAFs) for entries of single, multiple and key option types. The above files are then used to build a Questionnaire Entries File (QEF) to hold collected data and customize data entry procedures to suit the used data types. The routine customization process is adaptive in the sense that any changes made to the questionnaire description file are mapped directly to the data entry procedures.

In questionnaire analyses specification, queries are stored as search classes and then used in the tabulation processes. They are assigned user defined captions (QQDT) for easy retrieval and for use in tabulation, and a query description is stored as a set of conditions formed in relational algebra and stored internally as codes (QQDC) but displayed in natural language form. The analyzing routines then use the query descriptions given above to form criteria upon which search classes are formed in addition to automatically synthesized natural language documentation. Finally, tabulated forms are produced containing all or part of the above information as selected by the user.

Figure (3) emphasizes the automatically generated or repository taken procedures. The automatically generated subsystem consisted of 989 procedures and functions (Functional Components (FCs)) while the extra added procedures and functions consisted of 567 FCs. The percent automated part is therefore: 63.56%.

5.5. Illustrative example

A hypothetical questionnaire form has been designed to test the performance of the implemented system. The test is mainly directed to investigate the merits of building software systems using the above described concepts. Inherent capabilities of the system are highlighted for assessing the gained benefits of such automatically generated systems. It must be emphasized here that figure (3) shows the functional component clusters with captions related to the application example for illustrative purposes. However, these components are applicable to a wide class of similar information systems.

A set of 1000 questionnaire forms has been stored. Data values have been generated using a random number generator, bounded by meaningful limits, chosen according to the entries under consideration. Some human intervention was some times essential to force the data set to illustrate certain conditions. Table (IV) contains the chosen questionnaire entries and their attributes.

Following are some automatically generated output forms as samples of what could be obtained using the proposed system. Table (V) illustrates ten user defined search classes (shown only are user defined captions).

Table IV. Questionnaire entries and their attributes

Entry	Attribute	Possible answers
Employee name	Text	
Employee Sex	Boolean	MALE FEMALE
Birth Date of Employee	Date	
Salary plus Incentives	Real	
Marital Status	Single-option	SINGLE MARRIED DIVORCED WIDOW
Number of Children	Integer	
Transportation facility to reach firm	Multiple-options	PRIVATE CAR COMPANY BUS PUBLIC BUS UNDERGROUND WALKING BICYCLE MOTORCYCLE

Table V. User defined search classes

1	Male employees
2	Female employees
3	Employee name starts with SANDRA
4	Birth date later than 1-Jan-70
5	Single or divorced employees
6	Has more than 2 children
7	Preferred facility
8	Salary is less than $400
9	Salary is between $400-600
10	Salary is greater than $600

Table (VI) illustrates a typical system output produced from tabulating four examples of occurrences of selected classes. While first and second sections of table (VI) illustrate classes composed of one condition, the third and fourth sections display multiple condition classes.

Table (VII) illustrates a system output that shows two examples of occurrences of a certain class in another. Table (VIII) gives cross tabulated classes while table (IX) gives the same table but with resetting the percentile calculation option.

Table VI. Occurrences of a certain class

Occurrences of Class : 1 Male employees

Synthesized Description :

Class : 1

Employee Sex IS MALE
Occurrences of Class : [1] Male employees = 572
Percent Occurrences = 57.20 % Total Cases = 1000

Occurrences of Class : 4 Birthdate later than 1-Jan-70

Synthesized Description :

Class : 4

Birth Date of Employee GREATER THAN 70.01.01
Occurrences of Class : [4] Birthdate later than 1-Jan-70 = 326
Percent Occurrences = 32.60 % Total Cases = 1000

Occurrences of Class : 5 Single or divorced employees

Synthesized Description :

Class : 5

Marital Status IS SINGLE OR Marital Status IS DIVORCED
Occurrences of Class : [5] Single or divorced employees = 500
Percent Occurrences = 50.00 % Total Cases = 1000

Occurrences of Class : 7 Preferred facility

Synthesized Description :

Class : 7

Transportation facility to reach firm IS SUCH THAT :
PRIVATE CAR IS NOT SELECTED
AND COMPANY BUS IS SELECTED
AND BICYCLE IS NOT SELECTED
AND MOTORCYCLE IS NOT SELECTED
OR Transportation facility to reach firm IS SUCH THAT :
PRIVATE CAR IS RANKED 1
Occurrences of Class : [7] Preferred facility = 667
Percent Occurrences = 66.70 % Total Cases = 1000

Table VII. Occurrences of a class in another

Occurrences of Class : 3 Employee name starts with SANDRA
IN Class : 4 Birthdate later than 1-Jan-70

Synthesized Description :

Class : 3

Employee name IS SANDRA
Class : 4

Birth Date of Employee GREATER THAN 70.01.01
Occurrences of Class : [3] Employee name starts with SANDRA
IN Class : [4] Birthdate later than 1-Jan-70
IS = 47 OUT OF 326
Percent Occurrences = 14.42 % Total Cases = 1000

Occurrences of Class : 1 Male employees
IN Class : 5 Single or divorced employees

Synthesized Description :

Class : 1

Employee Sex IS MALE
Class : 5

Marital Status IS SINGLE OR
Marital Status IS DIVORCED
Occurrences of Class : [1] Male employees
IN Class : [5] Single or divorced employees
IS = 285 OUT OF 500
Percent Occurrences = 57.00 % Total Cases = 1000

Table VIII. Cross tabulation of classes

Synthesized Description

Cross Tabulation of Classes :

Class : 1

Employee Sex IS MALE

Class : 2

Employee Sex IS FEMALE

VERSUS Classes :

Class : 8

Salary plus Incentives LESS THAN 400

Class : 9

Salary plus Incentives GREATER THAN OR IS 400 AND
Salary plus Incentives LESS THAN OR IS 600

Class : 10

Salary plus Incentives GREATER THAN 600

Salary is less than $400
Salary is between $400-600
Salary is greater than $600

Class	Definition	8	%	9	%	10	%	TOTAL	%
1	Male employees	71	7.10	423	42.30	78	7.80	572	57.20
2	Female employees	113	11.30	308	30.80	7	0.70	428	42.80
	Total Incidents	184	18.40	731	73.10	85	8.50	1000	100.00

Table IX. Cross tabulation with resetting percentile calculations

Salary is less than $400
Salary is between $400-600
Salary is greater than $600

Class	Definition	8	9	10	TOTAL
1	Male employees	71	423	78	572
2	Female employees	113	308	7	428
	Total Incidents	184	731	85	1000

It is worth mentioning that all tables described above along with their synthesized descriptions are automatically generated to emphasize the feature that the output is self-documented.

The system has been put to practice by applying it to study the social impacts of the Sea Level Rise phenomenon on people living in vulnerable areas [16]. Two districts in Alexandria city had been chosen for the study, namely: Amreya and Semouha. A questionnaire form has been designed and the proposed software system has been used to manipulate the collected forms. The results obtained via report generation and query formation kernels have been successfully used in the study.

5.6. Generalizing the approach

The same procedures have been used to generate the other four systems by using the concept of script files to describe data flows. Other complementary functions that had to be constructed were added to fully characterize a given software system.

Table (X) summarizes the number of automatically generated functional components and their ratios to the total in each system. The ratios reflect how much data manipulations are in such systems. It is noted that the number of functional components has been increased in the latter systems along with increasing ratios of automation. This is attributed to the several amendments and refinements done during implementing the first software system. However, it must be noted that in (MSL) where domain dependent computations were required the value of (σ/Ω) was less than that of the others.

Table X. Automatically generated versus total functional components

System	σ	Ω	σ/Ω %
QUITE	989	1556	63.56
MSL	1794	3483	51.51
CLUC	2781	4322	64.35
ACS	5088	6373	79.84
SDO	4944	6022	82.10

Figure (4) illustrates the effect of α on θ. As α increases due to either refinement of the repository components or, on the contrary, the course production of complementary components, the value of θ decreases to a lowest limit of (1-σ/Ω).

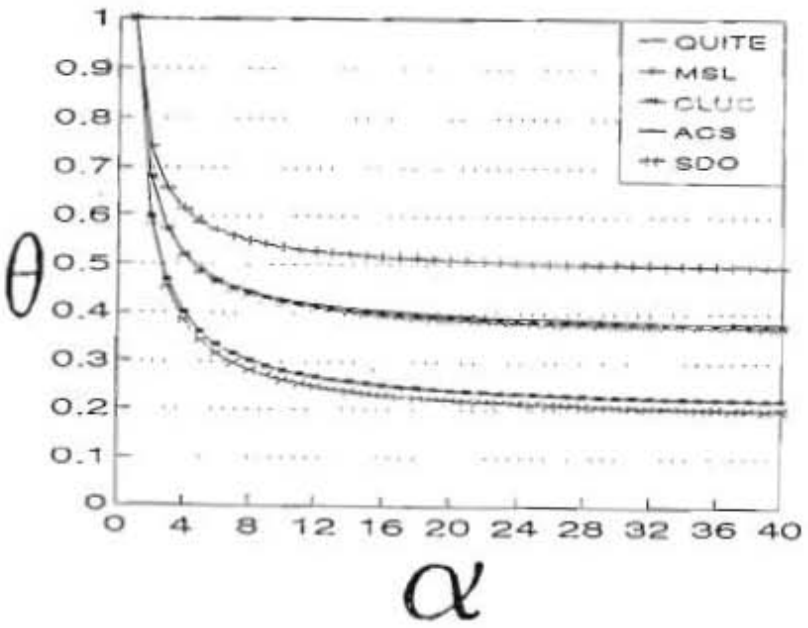

Figure 4. Effect of α on θ

6. CONCLUSION

The aspects of software development productivity and the final product overall quality have been investigated in this paper. Seven key issues affecting the two aspects have been examined.

Relying on a repository of thoroughly checked reusable software components, had a profound effect on reducing the software development life cycle. The concept of using scripts greatly facilitated rapid software prototyping where a user approval has always been obtained within a considerably short time.

Unifying data manipulation, query and report generation proeedures in a manner that is coherent with the user interface, resulted in both reducing the required development time and enhancing the overall working environment. The same data entry screens have been used for query formation and report generation. In query and report formation only some matching patterns are entered for retrieving single events or use the very same pattern to generate a report. The above described approach requires the user to get acquainted with a limited set of screen designs that are used in three different ways. On the other hand this approach substantially reduces the programming effort and time.

To some extent, the above described functional components are of comparable size and complexity, except for the MSL application. The development time of each FC is therefore approaching an average value of (T) given by ($T=\Sigma t_i/\Omega$) where t_i is the development time of FC_i. As a rough estimate, it can be concluded from table (X) that the development time has been reduced to range between 17.9-48.49% with an average of 31.73%, i.e. about one third of the required development time in classical systems design. However, the exact estimate is dependent on each individual system according to its nature.

It is evident that maintainability has been practiced on a limited number of components rather than implementing a full system from scratch. Also, modifications are generally reduced since they are applied to a very limited set of scripts rather than the actual code.

Performance of the final product has been related to the level of automation in the development process. It is concluded that the continuous refinements made to the repository components would lead to decreasing the probability of failure P_f with a rate proportional to ($1/\alpha^2$).

All of the above mentioned issues are always aiming at reducing the time consumed throughout the software development life-cycle and obtain as early results as possible from the proposed software system with reasonable quality.

One of the points which seems to be missing in this context is what's accepted to be a formal requirement specification. It is noticed that this may be successful on the level of single programs or functions, and could be understood by software developers. However, on the level of systems this needs a great deal of research and review in order to involve the ultimate target: *the end user*.

Also, metrics are being applied to code segments in order to quantify their performance [17,18[. Although metrics are also required on the level of systems, they are rather complicated and cumbersome to be practically applicable. For this issue, we suggest a potential solution and that's to apply appropriate metrics to the original scripts rather than the fully implemented software system.

REFERENCES

[1] Coats R.B. and Vlaeminke L (1987) Man-Compnter Interfaces, Blackwell Scientific Publications.

[2] Guirguis S.K., El-Racy M., Korany E. and Yehia S. (1993) Practicing Implementation of a Large Software System. Proceedings of Software Quality Management, SQM93, Southampton, U.K., 360-373.

[3] Rich C. and Waters R.C., (1988) Automatic Programming: Myths and Prospects. Computer, 40-51.

[4] Beiten E.A., Partsch H.A., Tuijnman D. and Volker N. (1992) How to Produce Correct Software - An Introduction to Formal Specification and Program Development by Transformations. The Computer Journal, Vol. 35, No. 6.

[5] Semmens L.T., France R.B. and Docker T.W.G. (1992) Integrated Structured Analysis and Formal Specification Techniques. The Computer Journal, Vol. 35, No. 6.

[6] Blackman M. and Jeffreys M. (1993), Quality Systems by Prototyping. Proceedings of Software Quality Management, SQM93, Southampton, U.K., 385–400.

[7] Buckley F.J. and Poston R. (1984) Software Quality Assurance. IEEE Trans. Software Eng., Vol. SE-10, No. 1, 36–41.

[8] Caldiera G. and Basili V.R. (1991) Identifying and Qualifying Reusable Software Components. Computer, 61-70.

[9] IEF (1990) Information Engineering Facility, Technology Overview, Texas Instruments Incorporation.

[10] IEF (1990) Information Engineering Facility, Methodology Overview, Texas Instruments Incorporation.

[11] Lockhart R. (1993) On Statistics in Software Engineering Measurement, Software Quality Journal 2, 49-60.

[12] Macro A. and Buxton J. (1987) The Craft of Software Engineering, Addison Wesley.

[13] Geist R., Offutt A.J. and Harris F.C. (1992) Estimation and Enhancement of Real-Time Software Reliability through Mutation Analysis, IEEE Trans. Computer, Vol. 41, 550-558.

[14] Norusis M.J. (1985) SPSS/PC+ for the IBM PC/XT/AT.

[15] SGC (1986) Statistical Graphics System. Statistical Graphics Corporation.

[16] Dessouky S.M. (1993) Remote Sensing and Geographic Information Analysis of Climatic Impact over Alexandria. M.Sc. thesis, Institute of Graduate Studies & Research, Alexandria University.

[17] Garlick F.J. (1993) Planar Similarity - A New Synthetic Metric. Proceedings of Software Quality Management, SQM93, 503-516.

[18] Bently W.G. and Miller E.F. (1993) Ct Coverage - Initial Results. Software Quality Journal 2, 29-47.

Appendix A. *List of abbreviations*

ACS	**Assets Control System**
CASE	**Computer Aided Software Engineering**
CLUC	**Container Loading/Unloading Control**
FC	**Functional Component**
MSL	**Machine Service Log**
MTBF	**Mean Time Between Failures**
MTTF	**Mean Time To Fail**
MTTR	**Mean Time To Repair**
QBE	**Query By Example**
QDF	**Questionnaire Description File**
QEF	**Questionnaire Entries File**
QQDC	**Questionnaire Query Description Classes**
QQDT	**Questionnaire Query Description Titles**
QUITE	**Questionnaire Utilization in an Integrated Tabulation Environment.**
SDO	**Social Development Organization**