Software for Quality Evaluation Based on EFQM Excellence Model

Monica Leba, Andreea Ionica, and Eduard Edelhauser

***Abstract*—** In this paper we'll present the achievement of an informatics system based on software engineering principles. This system is intended for the analysis and evaluation of a real enterprise achievements and needs based on the interactions between the EFQM Excellence model criteria.

***Keywords*—**EFQM excellence model, quality evaluation

I. Introduction

EVER since the first computer was built, the software writing represented an important and difficult task.

For many years the hardware part was the main one in producing and using computers from the financial point of view. Now, the hardware part is well developed and the effort is concentrated on the software part.

Due to the complexity of software production, there was necessary to establish principles, methods and means that allow producing quality software, also ensuring a precise functioning.

All of these form up the software engineering.

The name of software engineering appeared for the first time in 1967 when a group of researchers from NATO invented the expression "software engineering" as a form to express their believe that it would be possible to solve the present software crisis adopting the engineering principles in software development process.

This crisis was characterized by the continuous development of low quality software, with high price and big achievement time period.

The classical definition of software engineering, given in 1969, is: "The software engineering represents the establishment and use of thorough engineering principles in order to obtain low cost software, reliable and that could function in efficient way on real machines"[5].

II. Problem Formulation

In order to elaborate a software application there must be followed a set of activities grouped on several levels.

The first level consists in requirements specification and analysis. The requirements specification can be achieved only if it is understood the structure and behavior of the system that is to be computerized, activity called system engineering.

It follows the design level, which contains both the conceptual design of the system and the detailed design of the program.

Then, based on this project there is done the coding and the implementation. The resulting code is tested at modules and at entire system levels.

After testing is done, the system can be delivered. It must be, then, maintained and eventually developed in time.

From these activities grouped on levels can be extracted the so called standard life cycle of the software development process.

This is also called the classic waterfall model and is presented in fig.1.

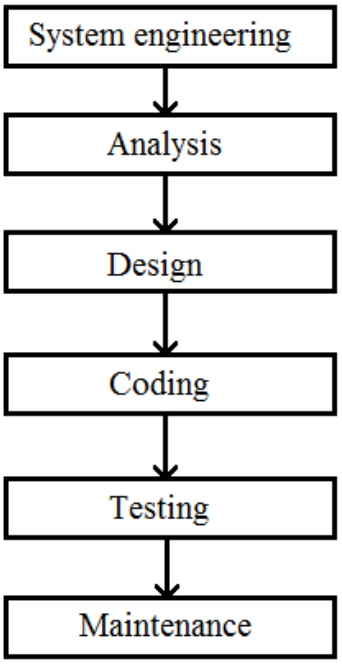

Fig.1 Classic life cycle

During this life cycle, there appear the following activities:

- System engineering: there are identified the requirements, are defined the functional interfaces

Manuscript received August 19, 2010.

M.Leba is with the University of Petrosani, Romania (corresponding author to provide phone: +4 0743172975; e-mail: monicaleba@upet.ro, monicaleba@yahoo.com).

A.Ionica is with the University of Petrosani, Romania (e-mail: andreeaionica2000@yahoo.com).

E.Edelhauser is with the University of Petrosani, Romania (e-mail: edi1ro2001@yahoo.com).

of the system;

- Analysis: there are defined the detailed requirements necessary to define the software performances, are identified the data flow characteristics and data structures;
- Design: The detailed requirements are translated into a series of representations of the system that describe the software build strategy. The design contains a description of the program structure and data structures;
- Coding: achieves the translation of the project in a form executable by the computer using an appropriate programming language;
- Testing: is an activity which verifies that each component of the software achieved is task correctly. Also, the testing certifies that the entire system satisfies all the requirements;
- Maintenance: is to reapply all the above activities for the resulted software. It could be necessary in order to correct a bug or to adapt the software to changes from the environment (new hardware, new operating system) or to ensure performances improvement.

III. SOLUTION

The paper proposes a digraph model used as an assessment tool, that allows the understanding of the interactions between the EFQM (European Foundation for Quality Management) Excellence model criteria, for the systemic analysis and the evaluation of the real organization achievements and needs, because TQM (Total Quality Management) must be seen as a response to real problems.

Matrix representation is useful in analyzing the digraph model mathematically and for computer processing. In order to obtain the Quality Index which represents the performance of the system (organization) by a single number, the permanent of the matrix was calculated.

The permanent is a multinomial and a standard matrix function, which has been used and defined in combinatorial mathematics by Jurkat and Ryser (1966) [2].

Our research approach consists of developing a digraph model of the interactions between the EFQM Excellence model criteria.

This model is implemented as dedicated software to evaluate the overall system performance.

A. System Requirements

System requirements determination is the first activity to be done during the software design process.

This activity is very important because is the basis for all the other activities from the life cycle.

The requirement is defined as a characteristic of the system or a description of what the system can achieve in order to satisfy its purpose.

The requirements are specified using expressions like: "must do", "should do" and "will do".

The requirement can also be defined as a feature that the system must have or a constraint that it must satisfy in order to be accepted by the client.

Or, the requirement is a specification of the system in a way understandable for the user.

The system requirements determination consists in two stages: requirements capture and requirements validation. In fig.2 are presented these two stages.

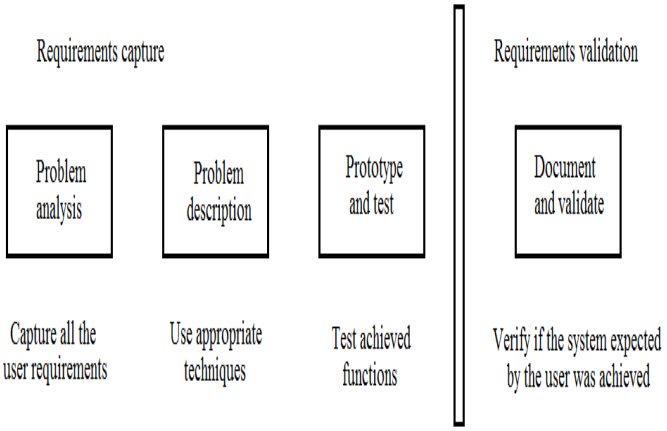

Fig.2 Requirements capture stages

For our system the captured requirements are as follows:

- The software must evaluate the current state of the system (organization) based on the EFQM excellence model criteria by computing the quality index;
- The criteria evaluation should be done depending on Enablers and Results;
- The software will allow the input of values for the Enablers and Results, but also some interdependencies between these;
- It will graphically display all the above values.

B. User Interface Design

The user interface design contains a logic design part and a physical design part.

For the logic design, there must be specified which are the data (attributes) visible for the user and how are these represented on the interface (for example, textbox, menu etc).

The physical design means the achievement of different interface models and validation with the client/user.

For this, there must be taken into account the followings: what information must the user transmit to the system; what information must the system transmit to the user; which are the elements of the interface useful for the user; what actions can the user do and what decisions can he take etc.

In fig.3 is presented the user interface designed for the developed application.

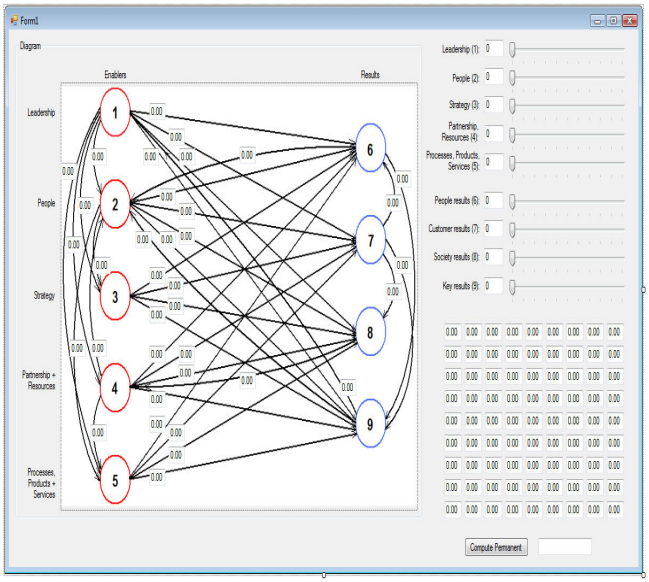

Fig.3 User interface

As can be seen, the user interface has in the left part the digraph graphical representation of the interactions between the EFQM Excellence model criteria, divided in Enablers and Results.

The user can introduce the interdependency values in the text boxes that are located on the edges between the criteria. In the upper right part are represented the values of each criteria, that can be introduced by the user either in the text boxes or using the sliders at the right.

All the changes done by the user in the digraph and in the criteria weights are visible also in the adjacencies matrix in the bottom right corner.

The user can obtain the overall system quality value by pushing the button Compute Permanent and the result is visible in the text box at the right.

C. *Software design*

Generally speaking, the software design represents the activity that produces a software model, based on the specified requirements. The design has the following characteristics:

- It is a creative process;
- The solution design for the same problem is limitless;
- It is a process of modalities (how) versus requirement (what) evaluation.

The design is both the creative process of transforming the problem in solutions and the description of this solution.

The software design can change during the software development, as the designers better understand the problem and as the requirements can be modified.

While the requirements describe what the system should do, the design process and the product describe the way the system will do.

The conceptual project tells to the client what the system will do, representing the "what" part of the solution.

Once the client approves the conceptual project, this is translated into the technical project, which is a more detailed document that allows the designers to better understand the necessary hardware and software to solve the client problem.

This represents the "how" part of the solution.

The conceptual project has the following characteristics:

- It describes the system limits, entities, attributes and relations;
- Gives answers to questions like: where come the data; what happens to the data inside the system; how will the system look for the users; what options will be available for the users; how the reports and screens will look like;
- It is described in the client language and does not use technical terms;
- Describes the system functions;
- Is independent of implementation;
- Is linked to the requirements document.

In other words, it allows to the client to understand what the system will do, explaining the characteristics observable from the outside.

The technical project has the following characteristics:

- Describes the hardware behavior and the functions;
- Describes the data structures and the data flow;
- Describes the software components architecture;
- Design the software structure.

This represents a technical view upon the system specifications. For the designer it is important the technical project, having the model from fig.4.

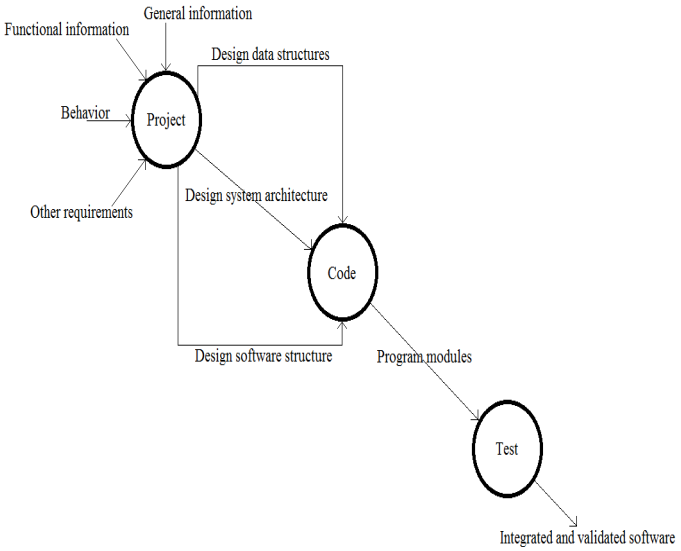

Fig.4 Software design model

The inputs of the model are the requirements that can be: general information, functional information, behaviors and so on.

The outputs represent three categories of design schemes, as follows:

- Data structures design;
- System architecture design;
- Software structure design.

D. Data structure design

The first activity during the data structure design is the selection of logical representation of the objects identified during the requirements and specifications definition phases.

This selection process contains the algorithmic analysis of the possible structures in order to determine the most efficient design.

For our application we have chosen to use an adjacencies matrix to represent the digraph.

This is the standard and most common way to work with graphs.

Using the adjacencies matrix, there can be extracted many information from the digraph.

E. System architecture design

The objective of this is to develop a modular program structure and to represent the control relations between the modules.

This design puts together the program and the data structures by defining the interfaces that allow the data flow in the program.

The architecture design offers a holistic view of the software.

The architecture design of our application is represented by the algorithm block diagram from fig.5.

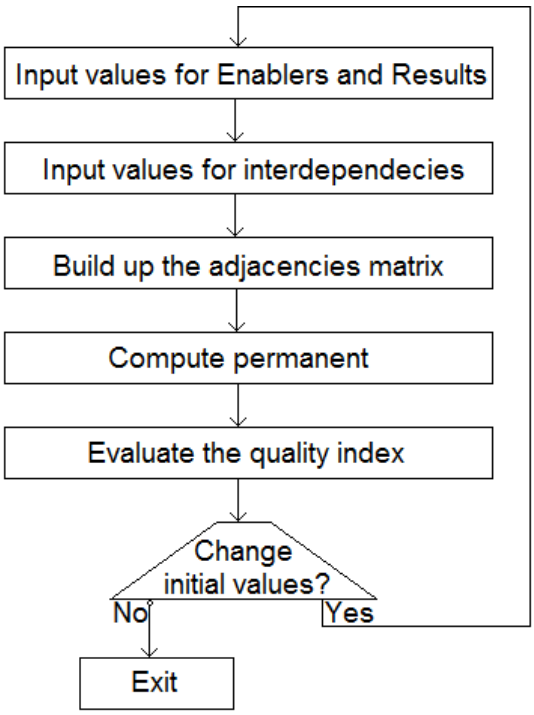

Fig.5 Algorithm block diagram

F. Software structure design

After designing the data structures and the system architecture, it is necessary the precise specification of the software structure. This means:

- The design paradigm (structured or object oriented);
- Graphical user interface;
- The programming language/environment.

For our application we have chosen the structured design paradigm, as it is an application that uses simple and few data.

The programming environment chosen is Visual Basic.NET, as it allows easy design of the graphical user interface from fig.3 and also easy implementation of the algorithm from fig.5.

G. Testing

The testing consists of the followings:

- Planning the tests;
- Design the tests (can be white box, black box, regression);
- Implement the tests;
- Achieve unit, integration, system and acceptance tests;
- Evaluate the tests;
- Debugging.

Testing is defined as the process of finding errors and the last verification of the specifications, projects and code.

The purpose the testing is to guarantee that all the elements of the application work good together, the system functions as it was said and satisfies the performance criteria.

Testing can also be defined as the process of finding the differences between the specified (expected) and the observed (existent) behavior of the system.

So, the testing purpose is to design tests that will systematically find the errors. The testing is usually done by the designers that were not involved in the system implementation.

It is impossible to achieve the complete testing of a complex system.

Also, many times, the systems are delivered without being completely tested.

In fig.6 is presented the information flow during testing.

Testing can be dynamic or static.

The dynamic testing deals with the product behavior verification and observation. This includes code execution.

The static testing deals with the analysis of the static system in order to discover possible errors. This does not include code execution.

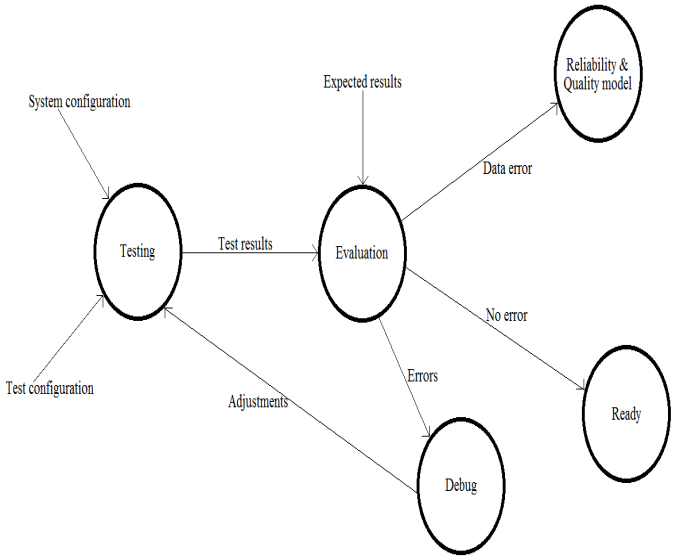

Fig.6 Information flow during testing

In fig.7 is presented the test process.

Errors' testing has as objective the discovery of errors in the program.

A test is said to be successful if it makes the program to act in an abnormal way.

The tests show the presence and not the absence of errors.

Only the exhaustive tests can prove that a program has no errors, but the exhaustive testing is not possible.

The test data are inputs created especially for testing the system.

The test cases are inputs for testing the system together with the outputs expected for these inputs in the case the system works according to the specifications.

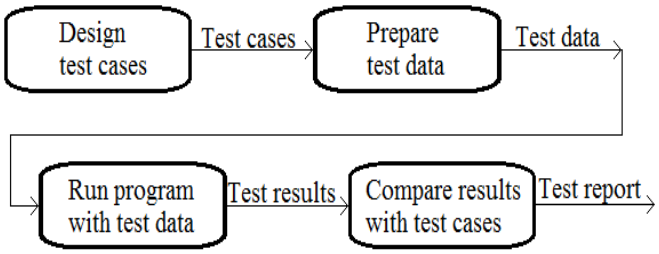

Fig.7 Test process

The application was tested for several cases.

First, there were tested the boundaries, that is the minimum and the maximum values possible. The results were as expected, as shown in fig.8.

Then, there was established an initial state, as in fig.9.a, and a state with raised accomplishment degrees both for Enablers and Results, as in fig.9.b.

As can be seen the software functions as expected.

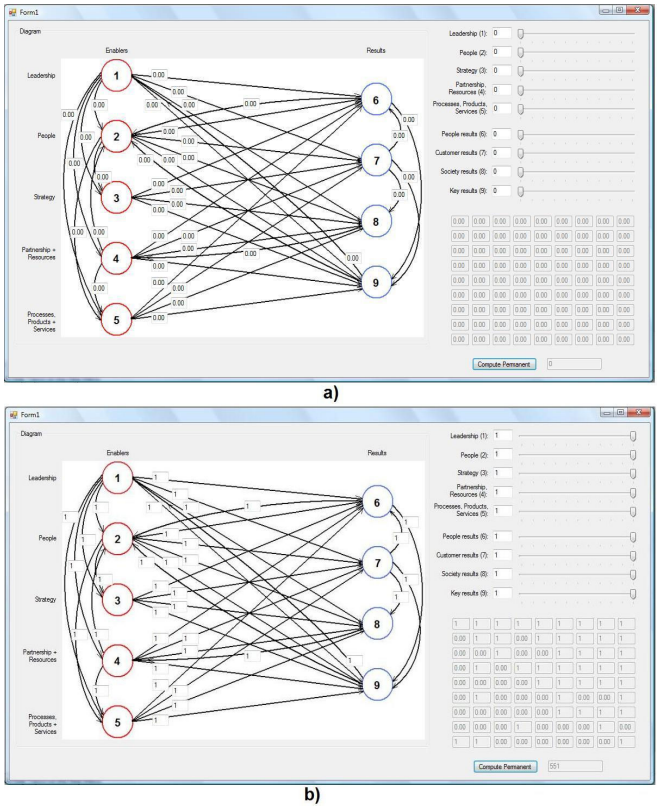

Fig.8 Boundaries test: a) minimum values; b) maximum values

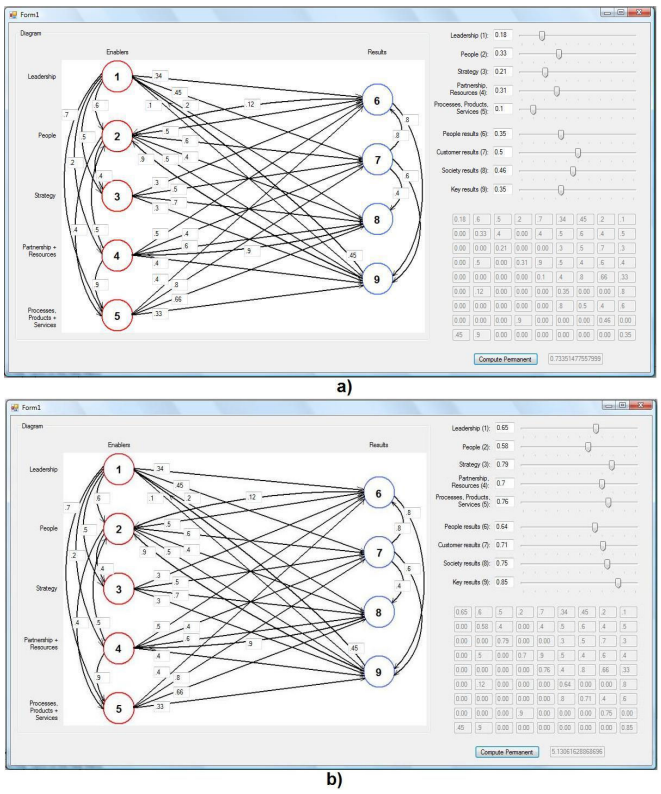

Fig.9. Test states: a) initial; b) raised accomplishment degrees

IV. Conclusion

The software developed in this paper was designed exclusively based on software engineering principles. It was intended as a computer implementation of the EFQM excellence quality model to evaluate the performances of an enterprise.

The software can be further developed in order to estimate both the values of Enablers, Results and of the interdependencies based on the results of some questionnaires.

References

[1] Grover, S., Agrawal, V.P. and Khan, I.A. (2005) "Human resource performance index in the TQM environment", International Journal of Management Practice, vol.1, pp 131-151.

[2] Ionica A.C., Edelhauser E., Leba M., "The Quality Digraph Model - A TQM Assessment Tool", RMEE (Review of Management and Economic Engineering) first Management Conference: Twenty Years After - How Management Theory Works 16th - 18th of September 2010, Technical University of Cluj-Napoca, Romania.

[3] Pop E., Leba M., Pop M., Sochira B., Badea A., "Software Based on Logic Neural Networks for Digital Controllers Design", Proceedings of 8th WSEAS International Conference on CIRCUITS, SYSTEMS, ELECTRONICS, CONTROL & SIGNAL PROCESSING (CSECS '09), ISBN 978 960 474 139 7, ISSN 1790 5117, Puerto de la Cruz, Tenerife, Canary Islands, Spain, pp. 168-173, 2009

[4] Pop E., Padurariu E., Leba M., Ciodaru D., "Software Engineering Approach on Administrative Management", Proceedings of 8th WSEAS International Conference on CIRCUITS, SYSTEMS, ELECTRONICS, CONTROL & SIGNAL PROCESSING (CSECS '09), ISBN 978 960 474 139 7, ISSN 1790 5117, Puerto de la Cruz, Tenerife, Canary Islands, Spain, pp. 186-191, 2009

[5] Ian Sommervile, *Software Engineering*, 7th edition, Addison Wesley, ISBN: 978-0321210265, 2004.