A Proposal of a practical approach for quantified quality software evaluation during the development cycle

AZARIAN Armin (1,2) – SIADAT Ali (2)
SIEMENS(1) AG A&D AS AP SE 3 IC - Laboratoire LGIPM
Ecole Nationale supérieure des Arts & Métiers
84 Siemensallee 76187 Karlsruhe – 4 Rue Augustin Fresnel 57007 Metz
GERMANY - FRANCE

Abstract: - A large number of developed, acquired or purchased software tools do not respond to the users' requirements and expectations which had been at the origin of the project. This is mainly due to two reasons: firstly because the users' requirements are not well identified or formalized, secondly because the software and tool evaluation is not robust enough or does not have a minimum required quality [1]. The authors attempt to propose a new approach in order to assess and quantify the quality of the software evaluation process. The theoretical approach is based on elaborating a matrix (A_{nm}) of software functionalities versus user's scenarios. The norm of the columns and lines vectors of this matrix may be considered as a quality indicator of the software. The authors have applied such an approach on one case study.

Key-Words: - Software Evaluation, Practical Verification Validation, Development, Functional Decomposition

1 Introduction

1.1 Overview of software developing

Most Software-projects statistic points out that these projects are late, over budget, lacking functionality, or are never delivered [1] [2]. The CHAOS study [3] reveals that only 16,2% of software projects are completed on time and on budget in small companies and around 31,1% of projects are canceled. Effective requirement management has allowed much progress to control quality, costs, schedule, functionalities and exhaustiveness. Also, if in a software project all requirements are adequately and exhaustively fulfilled, it doesn't imply that the developed software tool does satisfy the user's needs and expectations in terms of functionalities or performance. The following reasons can explain these cases (leading to unsatisfying software products) [4]:

- Users don't understand what they want.
- Users won't commit themselves to a set of written requirements.
- Communication between users and engineers or development team is slow and generates misunderstanding, misinterpretations…
- Users often do not participate in reviews or are incapable of doing so. And engineers or developers do not understand the user's view and needs.
- Users don't understand the development process.

These ways inhibit strongly user's requirements engineering for software [4].

Software analysis and evaluation is an essential and well established activity for improving software development and the architecting community of the software systems. The development effort, the time and costs of complex systems are considerably high, and nowadays there is an increasing need for a practical formalized and comprehensive evaluation method that encompasses all factors which affect the software's functionality and usability to achieve system's quality. The following section presents a brief summary of the existing evaluation methods.

1.2 Existing evaluation methods

Evaluation techniques are activities of evaluators which can be precisely defined in behavioral and organizational terms [5]. Evaluation techniques are usually classified into two categories: the **descriptive evaluation techniques** and the **predictive evaluation techniques**, both of which should be present in every evaluation:

- **Descriptive evaluation techniques:**
  - Behavior based methods:
    They contain observations, "thinking aloud" and video-confrontation method. The result of these methods is an interview protocol. The questions are mainly focused on critical points, like interactions.

- Opinion based methods:
  Interview methods, questionnaires like: QUIS (Questionnaire for User Interaction Satisfaction), SUMI (Software Usability Measurement Inventory), Isometrics [5] are part of these methods. The difference with the previous methods is that they rely on standard items and aim to reveal the user's opinion of the software [6].

These methods are used to describe the status and the actual problems of the software in an objective, reliable and valid way. All descriptive evaluation techniques require some kind of prototype and at least one user [7].

- **Predictive evaluation techniques:**
- Walkthrough methods:
  Usually papers with the software's GUI are presented to the evaluators [8]. They write how they think they would use the software and evaluate some standardized metrics for each step. These methods are well indicated to reveal usability problems.
- Expert inspections:
  The software is examined by a usability specialist independent of the software development team. The experts notes and evaluates some items of the software. The heuristic reviews [9] are a variant of this method.
- Group discussion:
  Group discussions help to summarize the ideas and comments held by individual members. Each participant acts to stimulate ideas, and that by a process of discussion, a collective view is established which is greater than the individual parts.

These methods aim principally to make recommendations for future software developments and the prevention of usability errors. These techniques are expert – or at least expertise – based. The criteria objectivity and reliability is hard to apply in these techniques.

1.3 Summary of software evaluation methods

These methods insist mainly on the evaluation of a part of the software product e.g.: User-Interface, Tasks to perform… The achievement of objectives results is expensive because they need to handle many data (from questionnaires or camera-records) [10] [11]. A subjective method delivers acceptance results of the software products and exaggerates weak points of the product. This paper proposes a matrix based approach to formalize a measurement and quantify the software quality and usability in order to facilitate the verification and validation steps. This paper proposes a matrix approach to formalize measures and quantify the software quality and usability. The approach consists on one hand, to assist the developer's team and the client for the specifications and on the other hand to evaluate the final product.

2 Proposed Approach

2.1 Description of the methodology

In the cycles of software development the first stage consists in preparing the analysis of needs and the feasibility study and often the second consists in writing the specifications. However, there is no formal methodology aiming at elaborating specification according to the analysis issues originating from the users' needs. Furthermore, the software is always tested aside when compared to the specifications, not often corresponding to the customers' expectations and finally implying insatisfaction (Fig. 1).

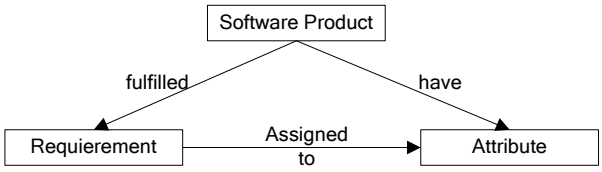

Figure 1: Software requirement engineering

The methodology proposed in this paper consists in working out usability scenarios during the first step of the software's design cycle. The analysis of this scenarios will point out the importance of the future functionnalities and quantify their weight. These usability scenarios represent the tasks which will be performed by the future users of the software. The drafting of these usability scenarios must be carefully realized by the client. They represent the base of the proposed evaluation method.

If there are several classes of users, then the scenarios should be indexed S_J according to the classes C_I . The first stage consists in building a list of the scenarios and to describe the functionalities (user's functionnality) which require an end-user action to be undertaken. Then a matrix is conceived with the scenarios users (dimension N) in rows and the functionnalities in columns (dimension m). This matrix called A is depictured in Fig. 2 (on the assumption of a single class of users):

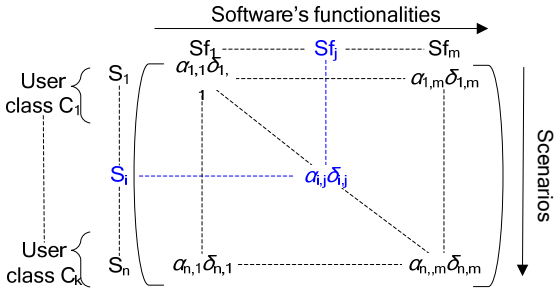

Figure 2: Representation of the scenario-functionality matrix

If dependent scenarios occur (eg. After a gaussian reduction in order to eliminate redundancies) for a same class of users they need to be re-worked or supressed.

This form makes it possible to visualize the coverage of the evaluation. The coefficient $\delta_{i,j}$ is equal to 0 if the scenario S_i does not use the functionality f_j (and 1 otherwise), the multiplying coefficient $\alpha_{i,j}$ represents the number of times the functionality f_j is used in the scenario S_i. It is supposed that the coefficients of the matrix are fixed manners to minimize complexity (cf. below) of the scenarios (it means that the scenarios are realized with the optimal number of functionalities in this case). The properties of the matrix A are deduced from the usual operators on the linear algebra, in particular the rank of the matrix:

This makes it possible to visualize the redundancies of the operations in the scenarios and the degree of coverage of the evaluation.

The complexity of a scenario is noted $\mathcal{C}(S_i)$ and represents an index specifying the number of functionalities which intervene in the elaboration of this scenario. The formal definition is given by the following equation (1):

$$\mathcal{C}(S_i)=\sqrt{\sum_{1\le j\le m}\left(\alpha_{i,j}.\delta_{i,j}\right)^2} \quad (1)$$

The scenarios are also affected by a frequency attribute which can be: "usual", "alternative" or "exceptional" called FU coefficient equal to 1, 1/2, 1/5. (eg: for the realization of a document over a remote session is in usual case an exceptional scenario and writing a document is a usual scenario for a text processing software).

- The weight of the functionalities in the matrix:

This makes it possible to visualize the truly useful functionalities to realize the scenarios. A weight coefficient is assigned to each functionality (2):

$$\mathcal{G}(Sf_j) = \sqrt{\sum_{1\le i\le n} FU_i\left(\alpha_{i,j}.\delta_{i,j}\right)^2} \quad (2)$$

The functionalities with a weight coefficient equal to 0 are not necessary to realize the scenarios. Consequently, these do not need to be developed.

The scenarios having a too high complexity compared to the average value require the designing of new high level-functionalities which are regroupings of smaller functionalities in order to simplify the realization of the scenarios by the users (see 3.2).

The functionalities which have a high weight must be carried out with fast response times because they are often used by the users. They also need to be easily located on the GUI (Graphical User Interface).

The columns which do not have any null vector must be considered with caution, because corresponding to a functionality requiring to be activated in each scenario, it is consequently necessary to wonder about the replacement of such functionality by an adjustment by default (treated in 3.1).

This approach allows determining exactly what the software system shall do. The matrix shows which function are necessary. When the developed software product is based on existing solutions, engineers can extract from their existing product which parts can be re-used and which need specific developments. In that case the developers of the existing software product need to elaborate the matrix, with the functionalities of the existing system (a blank column needs to be considered in the case that new functionalities must be added to achieve the usability scenario). The resulting matrix stipulates what must be accomplished, transformed, produced, provided or kept. It is also possible to decompose the functionalities in usability (low level functionality) and technical function (high level functionality), this allows showing which internal parts (methods or Libraries) should be re-used for the new solutions.

In this case, the method helps to establish a common communication base between client and development teams (who complicate the client's view when they refer to an existing solution). The solution could also help the clients to write the functional requirements of the Software product, because they visualize the capabilities and characteristics of the wished software systems.

2.2 Approach of the evaluation

The evaluation process will be performed according to the same basis of pre-defined scenarios. This time, an end-user will perform and execute the scenarios

with the help of a software prototype. The objective is to set up a matrix named R (lines for scenarios and columns for functionalities). This matrix R will be thus compared to the matrix A (already defined in previous section. One notes $\mathcal{C}_R(S_i)$ and $\mathcal{G}_R(Sf_j)$ the complexity of the scenario S_i and the weight of the functionality Sf_j in the matrix R. The coverage level of the evaluation process depends upon the number of listed functionalities knowing that the rank of the matrix A indicates the action redundancy within the scenarios.

The comparison of $\mathcal{C}(S_i)$ and $\mathcal{C}_R(S_i)$ enables the user to know if the software is well designed. If $\mathcal{C}(S_i) < \mathcal{C}_R(S_i)$; this means that the user invoked too many functionalities than necessary for realizing the scenario. It is also possible that the user did not find the functions or icons through GUI or he/she used the functionalities by chance and then suppressed the actions. Such a situation could also happen in case of inadequate settings or initialisation requiring a corrective action of the user (see 3.1). This implies a high useless complexity level and dissatisfaction of the end-user.

By viewing the actions of the end-user (or a video-record of his action), it is possible to measure the time that the user needs to find important weighted software functionality. Such functionalities often used, need to be placed in a position which facilitates access by a shortcut icon. The icon needs also to be well designed to facilitate the comprehension of the executed function.

3 Application case

3.1 Use case for the designing of a drawing software

The first example consist of the designing of a software aiming to producing drawings. The 3 defined scenarios are as follows:

1. Draw a black square on a background white colour and print it (S_1)
2. Draw a pyramide with triangular basis (without representing the hidden sides) on a background blue colour and print the drawing (S_2)
3. Draw a blue square on a background black colour and print it (S_3)

Their is just one class of user and all scenarios are considered as usual (FU=1 for all case). In these cases the specific solution is build on the base of a commercial available tool which presents the following functionalities:

- Rectangular selection (Sf_1)
- Clear (Sf_2)
- Selection of a colour (Sf_3)
- Zoom (Sf_4)
- Airbrusch (Sf_5)
- Text (Sf_6)
- Right line (Sf_7)
- Curve (Sf_8)
- Rectangle (Sf_9)
- Ellips (Sf_{10})
- Choice of colour selection (blue, white, black, red, etc....) (Sf_{11})
- File save (Sf_{12})
- Print (Sf_{13})
- Fulfilling (Sf_{14})

The matrix of scenarios versus functionalities is represented by figure 3 when considering that the background colour by default is white and when selecting a given colour, the choice remains valid.

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 5 & 0 & 0 & 0 & 2 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 2 & 0 & 1 & 1 \end{pmatrix}$$

Scenario's complexity:

$$\mathcal{C}(S_1) = \sqrt{3}\ ; \mathcal{C}(S_2) = \sqrt{31}; \mathcal{C}(S_3) = \sqrt{7}$$

Functionalities-Weight:

$$\mathcal{G}(Sf_7) = 5\ ; \mathcal{G}(Sf_9) = \sqrt{2}\ ; \mathcal{G}(Sf_{11}) = 3\ ;$$
$$\mathcal{G}(Sf_{13}) = \sqrt{3}\ ; \mathcal{G}(Sf_{14}) = \sqrt{2}$$

For other functionalities: $\mathcal{G}(Sf_i) = 0$

Figure 3: Matrix of scenarios-functionalities

The rank of the matrix is 3 (Fig. 3): this implies that none of the defined scenarios is redundant. According to the theoretical approach explained above, the complexity of each scenario as well as the weight of each useful functionality is calculated knowing that for other functionalities the weight factor is zero. It is noticeable that Sf_7 is the most important functionality in terms of weight, implicating that it remains indispensable.

The complexity of the scenario N° 2 is higher than the average which is also due to the subsequent use of Sf_7 and drawing of a pyramid. As a conclusion, it is recommended that the software enables the user to draw also a pyramid once and consequently reduces the complexity of the scenario.

Another noticeable aspect of the matrix concerns the defaults settings. As the setting of the colour necessitates user's action, one can envisage the modelling of such functionality. If the default color in the drawing software is blue and if the scenarios consist in drawing objects in black, then the preset

black for the color will allow simplifying considerably the complexity of the scenarios.
The conclusions of the methods for this example are:

- The functionalities Sf_1, Sf_2, Sf_3, Sf_4, Sf_5, Sf_6, Sf_8, Sf_{10} and Sf_{12} did not to be implemented because their weight is null.
- A new high-level functionality which allows to draw pyramids needs to be conceived.
- The black as default color setting is advantageous for the defined scenarios.

The functionalities with a high weight (often used to realized the scenarios) must been realized with fast response time and must be easy to locate on the GUI by the users.

3.1 Use case for the validation of a drawing software

If the precedent software has been realized without the functionalities Sf_1, Sf_2, Sf_3, Sf_4, Sf_5, Sf_6, Sf_8, Sf_{10}, Sf_{12} , and without a function to draw pyramids and that a future user realizes the 3 scenarios developped in section 3.1, then the functionalities used by the user to conclude the scenarios are represented in a matrix form (the scenarios are the lines of the matrix and the functionnalities are the colunms). We suppose we obtain the matrix R represented in Figure 4 (the indexes of the functionnalities Sf_7 respectively Sf_9 ; Sf_{11} ; Sf_{13} ; Sf_{14} are changed by the indexes: 1 respectively 2,3,4,5). The matrix A represents the matrix issue from section 5.1 with the new indexes of the functionalities.

$$R = \begin{pmatrix} 4 & 0 & 1 & 1 & 0 \\ 5 & 0 & 2 & 1 & 1 \\ 4 & 0 & 3 & 1 & 1 \end{pmatrix} \quad A = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 \\ 5 & 0 & 2 & 1 & 1 \\ 0 & 1 & 2 & 1 & 1 \end{pmatrix}$$

Figure 4: Matrix A and R

If the matrix A and R are being compared (Fig. 4), the complexity of the scenarios 1 and 3 of the matrix R are superior to those of the matrix A. This is the result of a wrong manipulation of the software by the user, because he didn't use the function „rectangle" for the scenarios 1 and 3, he prefered to use 4 times the function „right line" to design a rectangle. The user did not remark the icon of the function „rectangle", and the software did not mention the existence of this function to teach the user. A dialog box could have appeared and mention the function after recognizing that the user traces a rectangle by using 4 times the functions „right line".

The comparaison of these two matrixes highlights that the user didn't use the software as mentioned. The cultural origin, the age, and the knowledge of the final users must be considered to design comprehensive icons, so that they are placed adequately in the GUI. Furthermore we can consider dialog boxes which inform the user of new functionalities.

4 Conclusion

There are many evaluation techniques which can be applied in practice, although many of them need to devote many efforts and special competences (experts). The method developed in this article allow for small companies to apply a low cost evaluation in particular with no experts knowledge, and to use this method to develop a practical verification and validation (V&V) tool in order to achieve an evaluation of the software's quality (with the help of the user scenarios). The method combines cost-benefits, adequate coverage of the functionalities, and a feedback which enables the user to increase the software's usability. This method is similar to the axiomatic design method which aims to "make human designers more creative, reduce the random search process, minimize the iterative trial-and-error process, and determine the best design among those proposed.", by revealing the needs of developing high level functionalities. The methods also propose a common base to avoid articulation problems between clients and engineers.(The engineers can visualize the user's issue, which is a frequent occurred situation).
The complexity level of each scenario as well as the weight factor of each functionality considered as a performance metric are defined. The software evaluation method allows to understand the design imperfections especially regarding the GUI for which the weight factor is playing a major role in order to highlight the useful functionalities offered by GUI. The basic criteria of the matrix approach are matrix rank, scenario complexity and functionality weight coefficient. The quantification is made by calculation of the norms of line and column vectors and comparison with average values. The theoretical approach has been applied to a fictive drawing software study. The results seem satisfactory and meaningful.
A tool has being designed to compute automatically the weight of the function and the scenario's complexity.

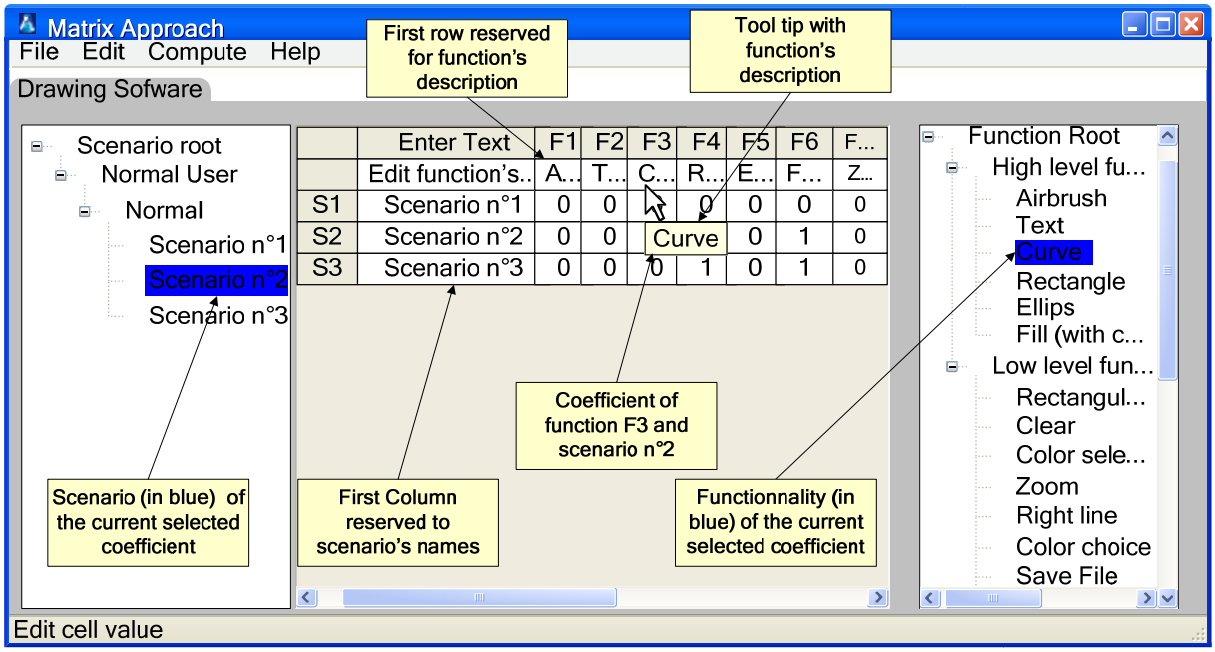

Figure 5: GUI of the Software Evaluation tool

In Fig. 5 the GUI of the tool which implements the proposed evaluation method is depicted.

In a near future, this method will be retained for the case of the development of a branch specific solution based on an existing commercially available tool in order to provide the parts that need client-specific development and the parts that need to be transformed or kept by achieving the user's issue in mind. This case will take into account different user classes and also the role of the frequency attribute of the scenarios.

5 Acknowledgment

This work has been partly performed within Siemens AG Company (Germany). The authors wish to thank this company for the support and opportunities offered for performing this research work.

References:

[1] X1. Azarian, A., Brindejonc, V., Bruère, J.M, *Investigation about elearning systems* – SINTES'12, 2005

[2] X2. Zelesnik, G., *Introduction to Software Requirements. (Requirements Engineering course material*), Software Engineering Institute, Carnage Melon University, Pittsburgh, PA. eds. 1992

[3] X3. Yannouth, M., CHAOS (*Application Project and Failure*), Standish Group Study, 1995, pp.1-4

[4] X4. Boehm, B.W., *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, Inc. 1981

[5] X5. Gediga, G., Hamborg, K. C., *Heuristische Evaluation und IsoMetrics*: Ein Vergleich, 1997

[6] X6. Dzida, W., *Qualitätssicherung durch software-ergonomische Normen*, 1994

[7] X7. Nielsen, J., *Usability engineering at a discount*. ACM New-York, 1989

[8] X8. Carroll, J. M., *Human-Computer Interaction: Psychology as a science of design*, International, Journal of Human-Computer Studies, p. 46, pp. 501–522, 1997

[9] X9. Nielsen, J., *Heuristic evaluation* New York. Wiley, 1994

[10] X10. Eberleh, E., Oberquelle H., Oppermann R., *Einführung in die Software-Ergonomie*, Berlin. de Gruyter, 1994, pp. 373–406

[11] X11. Salvendy, G., Smith, M., *Designing and Using Human-Computer Interfaces and Knowledge Based Systems*, Amsterdam. Elsevier, 1987, pp. 394–401.