A Synthesis of Software Evaluation Methodologies and the Proposal of a New Practical Approach

Armin AZARIAN
Arts et Métiers ParisTech / Laboratoire LCFC, Metz, France
Email: Armin.azarian@metz.ensam.fr

Ali SIADAT
Arts et Métiers ParisTech / Laboratoire LCFC, Metz, France
Email: Ali.Siadat@metz.ensam.fr

Abstract— **A large number of developed, acquired or purchased software tools do not respond to the users' requirements and expectations which had often 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 as they should, secondly because the software and tool evaluation is not robust enough or does not have a minimum required quality [1]. We 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 ($A_{n,m}$) 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 process. Performance metrics are also derived from such indicators to summarize the quality of software evaluation process from the users' point of view. We have applied such an approach on one case study and derived indicators like task effectiveness and efficiency of the software product evaluation. This method helps in two stages of the development cycle of the software: during the design phase and during the verification and validation. The case study helps to identify software imperfections and insufficiencies in a practical manner early in the development lifecycle. Thus improving subsequent releases of the software product.**

Index Terms—**Operational Software Evaluation, Functional Matrix Decomposition, Verification and validation Process, Process Quality, Indicator and Metrics**

I. Introduction

Most Software project statistics point out that the projects are overdue, over spent or budget, lacking functionality, or have never been delivered as they should [1] [2]. The CHAOS study [3] reveals that only 16.2 % of software projects are completed on time and on budget within small companies and around 31.1 % of projects are cancelled before completion. Effective requirement engineering management has allowed much progress to control quality, costs, schedule, functionalities and completeness. Also, if in a software project all requirements are adequately and exhaustively fulfilled, it does not imply that the developed software tool will satisfy the users' needs and expectations in terms of functionality and performance.

In the quality management domain, the term "quality" is sometimes ambiguous and must be considered as a multidimensional concept. From the beginning of quality engineering the term "quality" was recognized as eliminating "bugs" in software. Nowadays, the dimension of quality in software application includes the whole role players: clients, users, developers, etc. A software product that is regarded as high quality by developers may be totally rejected by users, because it would be focused on failure rate, on processes, on internal quality. Often software is developed, but does not meet the users' expectations and needs. [4] identifies three essential elements when handling software products: the **user** who employs a **computer** with specific software to complete specific **tasks**.

In order to achieve the realization of a software product that meets the users' needs and satisfaction, many evaluation methods have been developed. But they are often ignored by developers and a general commonly practice operalization of the methods does not exist and they rarely provide a quantified benchmark of the software's quality. In this paper we have developed an approach which allows quantifying the quality of the software evaluation process. Our approach is based on the completion of some essential tasks by a user. The intermediate steps and use of software functionality during the scenario are saved in the form of a matrix. The analysis of this matrix allows the determination of the degree of coverage for the evaluation. It also allows determination of non-essential functionality, missing functionality and consideration towards optimal GUI design.

The paper begins with a description of the existing evaluation methods. Section II presents the author's approach. Section III presents a use case based on a

Supported by the scientific research department of Arts et Métiers ParisTech.

Communication author: Ali Siadat, Associate Professor, Arts et Métiers ParisTech, Laboratoire de Conception Fabrication Commande, Arts et Métiers ParisTech, 4 Rue Augustin Fresnel, 57078 Metz, France
Email: ali.siadat@ensam.fr

simplified drawing application with a case description, an empirical study and finally a synthesis and confrontation to related evaluation methods. Section IV provides a conclusion of our approach.

A. Overview Software development

Requirements engineering management has helped a lot to increase users' satisfaction. Indeed, it helps to establish and maintain an agreement between customer and developer on both technical and non-technical requirements. This agreement forms the basis for estimating, planning, performing and tracking project activities throughout the project and for maintaining and enhancing developed software. But requirements management still has drawbacks leading to unsatisfying software products due to the following encountered facts [5]:

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

These aspects inhibit strongly users' requirements engineering for software [5].

The assessment of the human-computer interface which is not configured in an ergonomic manner often results in the software not being accepted by users. The functionalities of the software are not fully recognized by the users or functions and operations are utilized incorrectly and faulty operation might lead afterwards to possibly uneconomical situation or other undesirable effects.

The software ergonomics are originated from the multiple possibilities offered by technology to present information on the screen. Beside the questions of performance and effectiveness, the adaptation of computer systems to users is the bottom line. The realization of the objective usability for software systems is on one hand assisted by international standards [6] and on the other hand by evaluation measures. Software analysis and evaluation is an essential and well established activity during the entire lifecycle from the development to application for the verification and validation of the software's adherence to the quality demands. Wottawa [7] defines the evaluation as "A collection and combination of data with a weighted proposition of scale which allow achieving a comparative or a numeric assessment" The evaluation means also a systematic collection, appraisal and interpretation of data in order to allow the achievement of a reliable and valid assessment of the user interface [8]. The execution of software evaluation is essential for the design of comprehensive and ergonomic software, but nowadays the development effort, the time and costs of complex systems are considerably high, and there is an increasing need for a practical formalized and comprehensive evaluation method that encompasses all factors affecting the software's functionality and usability to achieve a high quality of the system. The following section presents a brief summary of the existing evaluation methods.

B. Existing evaluation Methods

Software products can be evaluated on various levels: the code (classes, blocks), the representation of information, etc… and the evaluation has different objectives [9]. According to Holz auf der Heide [10] the evaluations can be classified in the following groups depending on their objectives:

- Which one is better?

In this case, at least two systems are compared. It is used to determine the best product or the best system of a series of prototypes. In these types of evaluation criteria like subjective user's satisfaction or objective performance measure can be used.

- How good?

The evaluation examines a desired or required characteristic of the system.

- Why bad?

The objective of this type of evaluation is to collect indications of weakness in order to deliver propositions for the user's interface. Often, these evaluations are performed with interviews, video-recording of a user who performs a series of tasks.

In this section, we describe only evaluation methods that are concerned with the usability of software. Evaluation techniques are activities of evaluators which can be precisely defined in behavioral and organizational terms [11]. Many authors attempt to group evaluation methods in different classification schemes like: subjective, objective, analytic, empiric, heuristic, etc. One type of classification of evaluation techniques divides the methods into: the group of descriptive evaluation techniques and the group of predictive evaluation techniques [12], but actually no widely accepted classification scheme exists, meanwhile the criteria "user-friendliness" and "usability" [13] as basis for the software design has achieved general acceptance.

- **Descriptive evaluation techniques** are used to describe the status and the actual problems of software in an objective, reliable, and valid way. These techniques are user based and can be subdivided into several approaches like: opinion based methods, behavior based method and usability testing [12].
- **Behavior based methods** contain observations, "thinking aloud" and log or video-recording method. The result of these methods is an interview protocol. The questions are mainly focused on critical points, like interactions.

Thinking aloud is often applied in formative evaluation [14]. Three or five users are asked to describe their considerations, problems, alternatives when they realize tasks. This technique can also be applied without predefined task; in that case the user explores the system on his initiative, which can be helpful to get a general impression of the system (cf. [14], [15]). The procedure to transform the lingual information into criteria is only weakly standardized.

In log-recording, all observations and in particular all interactions with the user interface of the systems are saved in a file. Sometimes a video recording is prepared at the same time. All steps of the user: keyboard and mouse events are saved in a log file, but the analysis of this file does not allow deducing the considerations or impressions of the user which leads to behavior. Table 1 gives an example of a protocol [16].

- **Opinion based methods**: Interview methods, questionnaires like: QUIS (Questionnaire for User Interaction Satisfaction), SUMI (Software Usability Measurement Inventory), IsoMetrics [11] 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 on the software [13].

The evaluation with questionnaires is a good technique if an important number of participants is consulted. The acquisition of data from questionnaires offers the facility to obtain opinion on a certain domain of the system. Quantified results like frequencies can be easily extracted from the questionnaires. Specific questionnaires help to investigate restricted areas of the software systems. The usual questionnaires are:

- **QUIS**: "Questionnaire for User Interaction Satisfaction" was developed by Norman und Shneiderman [15] and constitutes a reliable, consistent instrument for the measure of satisfaction of software systems. The questionnaire provides a measure of overall satisfaction. It evaluates some aspects of the user interface based on user opinion.
- **SUMI**: "Software Usability Measurement Inventory" was developed initially to measure the user's perceptions of the usability. SUMI consist of 50 items which are assigned to scale. SUMI was supplemented with the "Item consensual analysis" (ICA) which recognizes patterns of response. A comparison of excepted and observed frequencies allows showing the items which demand a change.
- **IsoMetrics**: The Isometrics usability inventory provides a user-oriented approach for software evaluation on the basis of ISO 9241-10 [6].

The important advantages and drawbacks of questionnaires are reported in [16]. In interviews the investigator needs to appraise the software in the front end, in order to recognize possible weaknesses and to go deeper into interrogation. If the number of testers is not too important, interviews remain a good technique to work out the opinion of the users. An interview can follow guidelines with specific inputs and possible answers (structured interview) or in an unstructured manner. An investigation about the properties of the interview situation is reported in [17], [18]. The descriptive 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 [19].

TABLE I.
EXAMPLE OF A PROTOCOL .

Task Nr.	Function Nr.	Retry	Function time (s)	Break time (s)	Timestamp h.m.s.cs	Canceled	Input means (Keyboard)	Test attendee	Essay Nr.
1	251	1	15	188	16.56.34.81	1	1	8	1
2	15	1	4	16	16.57.05.46	0	1	8	1
3	251	2	17	0	16.57.09.46	0	1	8	1
4	15	2	4	50	16.58.16.25	0	1	8	1
5	251	3	76	0	16.58.20.25	0	1	8	1
6	15	3	5	176	17.02.32.08	0	1	8	1
7	24	1	0	36	17.03.13.71	0	1	8	1
8	…	…	…	…	…	…	…	…	…

- **Predictive evaluation techniques:**
- **Walkthrough methods:** Usually papers with the software's GUI are presented to the evaluators [20]. They write how they think they would use the software and evaluate some standardized metrics for each step. These methods (e.g. cognitive or usability walkthrough) are well indicated to reveal usability problems. Cognitive Walkthrough is a task oriented inspection method without an end-user. The usability expert explores the functionalities on behalf of an imaginary user. The expert assumes that the user explores the software in a way that minimizes the cognitive effort (cf. [21], [22]). For each atomic action within the task sequence a success story is constructed which contains the reasons. The advantage of CW is that it can be applied at an early stage of developments but the simulated steps of a scenario and the estimation of the user's behavior by the expert has to be corrected.
- **Expert inspections and Heuristic reviews:** The software is examined by a usability specialist independent from the software development team. The experts notes and evaluates some items of the software. The heuristic reviews [23] are a variant of this method. They generate a problem list as basis for the software optimization. In a free expert inspection the evaluators check the software using in general guidelines. The inspection is carried out by a user who is not directly involved with the development team. The evaluation can be realized in an early stage of development with paper prototypes. Heuristic evaluation is a special case of an expert inspection except that the cost is lower, so that they are sometimes affected with the label: "discount usability engineering method". The evaluation is focused on the following heuristics:
  - Provide a simple and natural dialog;
  - Speak the user's language;
  - Minimize user's memory load;
  - Be consistent;
  - Provide feedback;

- Provide clearly marked exits;
- Provide shortcuts;
- Provide informative error message;
- Prevent error.

These heuristics have to be considered as general principles which guide the evaluator during the inspections. Evaluators inspect the user interface in individual sessions. The degree of standardization is still low. Finally, the heuristic review shares a common disadvantage with CW: the evaluator must be user and task expert.

- **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. The object of the evaluation is the whole user interface of the system. Group discussion is quite flexible in its orientation: it can be focused on users, tasks, etc. The discussion has to be moderated by a human expert. The application of the techniques tends to be more efficient in the early stages of development process.

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

C. Summary of evaluation methods

In the domain of software-ergonomic evaluation, it is usual to be confronted with concepts like: formative, summative, quantitative, qualitative, subjective, objective, predictive, descriptive, analytic, empiric, etc…

An interesting classification had been made by Görner and Ilg [8] in which the evaluation methods are sorted depending upon the data ascertainment and the involvement of user (see Fig. 1) which easily permits to see the objectivity of the results. In subjective methods, flexible data are used to determine if the system is suitable for the realization of tasks. Objective methods try to eliminate subjective influence.

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) [24] [25]. A subjective method delivers acceptance results of the software products and exaggerates weak points of the product. Stowasser [26] lists the following conclusion:

- The evaluation methods are in part defined inadequately or in a conflicting manner. The description of the methods is often based on inconsistent terminology.

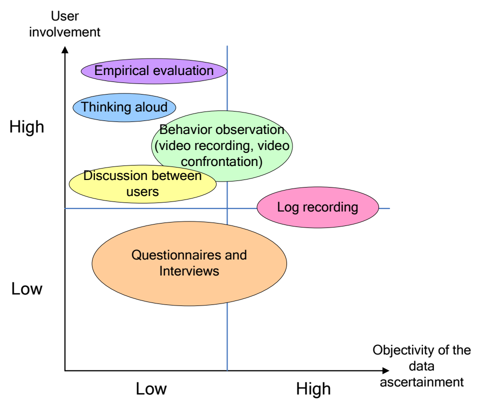

User involvement
High
Low
Empirical evaluation
Thinking aloud
Behavior observation (video recording, video confrontation)
Discussion between users
Log recording
Questionnaires and Interviews
Low
High
Objectivity of the data ascertainment

Figure 1. Data ascertainment proceedings for the evaluation.

- Suggestions for a general operalization of the evaluation criteria with respect to a certain method rarely exist.
- It appears to be difficult to compare the methods with one another. They use varying data collection techniques and differ in their degree of formalism and in their theoretical basis.
- The methods are rarely standardized, meaning that although the same methods are applied in different examinations, the results are not comparable.
- A pragmatic approach is often suggested for the decision regarding the selection of methods for a software evaluation. There is no typical alignment of evaluation goals and evaluation methods.

Therefore, there is a need for a high quality evaluation process. This paper proposes a matrix based approach to formalize a measurement and quantify the software evaluation process in terms of quality and usability in order to facilitate the verification and validation steps. The approach consists on one hand, of assisting the developer's team and the client for the specifications and on the other hand of evaluating the final product.

II. Proposed Approach

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 technical specifications [27]. However, there is no formal methodology aiming at elaborating a specification according to the analysis issues originating from the users' needs. Furthermore, the software is always tested aside when compared to the initial specifications, not often corresponding to the customers' expectations and finally implying dissatisfaction.

The methodology proposed in this paper consists in working out the concept usability of a software product. Under usability the following questions have to be addressed: "Is the software user friendly", "Is the software agreeable to use?", and "Is the software appropriate for the realization of tasks?" (See Fig. 1). According to [6] the definition of usability "the extent to which a product can be used by specified users to achieve specified goals with effectiveness" encompasses three concepts: Effectiveness, Efficiency, and Satisfaction.

Our approach consists in working out usability scenarios during the first step of the software's design cycle. The analysis of these scenarios will point out the importance of the future functionalities 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 basis of the proposed evaluation method.

If there are several classes of users, then the scenarios should be indexed S_j and sorted into the classes C_i. User classes represent categories of users that use the software product in different ways, for example: an expert user (who uses advanced functionality) and a remote expert. The classes of user are thus sets of scenarios (see Fig. 2) which correspond to the specific tasks a given user class will have to realize.

The second stage consists in building a list of the scenarios associated with each user class consisting of the usual tasks that will be performed with the help of the system. The list of all the functionalities of the system has to be established. Here, only the user's functionality are considered, which require an end-user action to be undertaken. Then a matrix is built with the scenarios' users (dimension N) in rows and the functionalities in columns (dimension M). This matrix called A is depicted in Fig. 2.

This form makes it possible to visualize the coverage of the evaluation (Fig. 2). The coefficient $\delta_{i,j}$ is equal to 0 if the scenario Si does not use the functionality Sf_j (and 1 otherwise), the multiplying coefficient $\alpha_{i,j}$ represents the number of times the functionality Sf_j is used in the scenario S_i. It is supposed that the coefficients of the matrix are fixed manners to minimize complexity (cf. section II.A) of the scenarios; it means that the scenarios are realized with the optimal number of functionalities in this case.

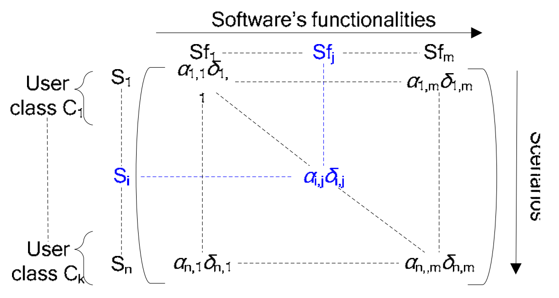


Figure 2. Representation of the scenario-functionality matrix.

The properties of the matrix A are deduced from the usual operators on the linear algebra. For each class of users a block matrix can be extracted and a Gaussian reduction can be performed on the rows of the matrix in order to eliminate redundancies. If redundancies occur (e.g. one scenario is a linear combination of others), then it has to be suppressed. This makes possible to visualize the redundancies of the operations in the scenarios and the **degree of coverage** of the evaluation depending upon the used functionality. This aspect will be detailed practically in the case study (section III).

In parallel, a time matrix T is built on the same basis (scenarios as row and functionalities as column). The coefficient $T_{i,j}$ of the matrix corresponds to the time taken by a user realizing the scenario S_i to find the function Sf_j and execute it.

The matrix A and T will permit to derive performance indicators (section II.A). The theoretic matrix A and T represent the ideal case, which means:

- All scenarios are successfully completed with the minimal call of functions.

- The time to find a function is minimal and the execution time of a function corresponds to the requirements.

The matrix A' and T' are built on the same model as A and T, but this time they are built experimentally when a user attempts to realize the scenarios.

A. Complexity and weight indicators

The complexity of a scenario is noted $\xi(S_i)$ and specifies the number of functionalities which intervene in the elaboration of the given scenario S_i. The formal definition is given by (1):

$$\xi(S_i) = \sqrt{\sum_{j=1}^{m} \left(\alpha_{i,j}\delta_{i,j}\right)^2} \tag{1}$$

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. For example: if a text processing software is considered: the writing of a text document is a normal scenario, but the realization of a document over a remote session is an exceptional scenario.

- 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 $\omega(\mathrm{Sf}_j)$ is assigned to each functionality (2):

$$\omega(\mathrm{Sf}_j) = \sqrt{\sum_{i=1}^{n} FU_i \left(\alpha_{i,j}\delta_{i,j}\right)^2}\ . \tag{2}$$

The functionalities with a weight coefficient equal to 0 are not necessary to realize the scenarios. Consequently, these do not need to be developed. And if they are really required by the client, the performance of the function does not need to be optimized. 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 graphical user interface.

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

The column vectors of matrix A which do not have any null value 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 if possible by an adjustment by default (treated in III.A). The detailed matrix is depicted in Fig. 3. The overall performance of the software can be measured with the help of the following metrics (which complete the methodology proposed in [27]):

-Time factor: Software is developed in order to perform some tasks in an efficient way. The time factor identifies where a task is performed in a short time but gives no output about the quality of the result.

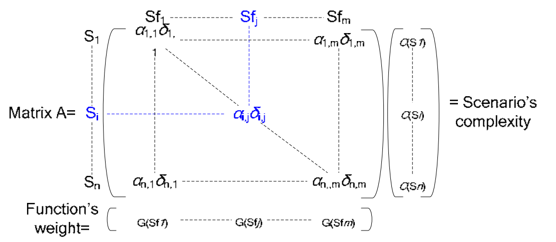


Figure 3. Representation of the scenario-functionality matrix with weight and complexity.

-Quantity: The quantity is the percentage of functionalities used to complete a scenario.

-Quality: The quality is the deviation between a user's scenario realized in an ideal way and the realization of the same scenario by a user (see III.B)

-Effectiveness: The effectiveness is an aggregate index of the quantity and quality. The effectiveness is given by (3). This metric definition is motivated by the need for a unique factor that quantifies how good the software is at the realization of a scenario. Moreover the geometric mean returns a more centered aspect of the data and is less sensitive to variation. This makes it adequate as an "overview indicator".

$$Effectiveness = \sqrt{Quantity * Quality}\,. \qquad (3)$$

-Efficiency: The efficiency is defined as the division of the effectiveness of a scenario by the time the scenario is completed. This metric summarizes how good the software allows to realize tasks in a reasonable time.

-Success rate: The success rate is the percentage of scenarios completed successfully by the users.

This approach allows determining exactly what the software system is capable to do. The matrix shows which functions are indeed 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 this 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 which 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), enabling to show 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 baseline 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 favored software systems.

B. 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 *A'* (lines for scenarios and columns for functionalities). This matrix *A'* will thus be compared with the matrix *A* (already defined in a previous section. One notes $\xi_E(S_i)$ and $\omega_E(Sf_j)$ the complexity of the scenario S_i and the weight of the functionality Sf_j in the matrix *A'* (The *E* in the index stands for experimental meaning that this time the matrix *A*' reflects the use of functionality by a real end-user in the completion of the different scenarios). The coverage level of the evaluation process depends on the number of the listed functionalities knowing that the rank of the matrix *A* indicates the action redundancy within the scenarios.

The comparison of $\xi(S_i)$ and $\xi_E(S_i)$ enables the user to know if the software is well designed. If $\xi(S_i) < \xi_E(S_i)$; 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 graphical user interface 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 initialization requiring a corrective action of the user (see III.A). 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 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.

III. USE-CASE

The use case for design and validation of the approach is based on a fictional drawing application. The functions that will be implemented in the application are detailed in table 2. The user interface is depicted in Fig. 4 and is derived from a real painting application. In this section 3 scenarios detailed in Table 3 are considered.

In this use-case only one class of users is considered and all scenarios are usual (FU=1 for S_1, S_2 and S_3).

In the first subsection (3.A) the evaluation approach will be applied during the design phase of the software product. In the second subsection (3.B), the user will be confronted to a prototype.

A. Use case for designing a drawing software

The ideal matrix (*A*) of scenarios versus functionalities and the time matrix (*T*) are represented in Fig. 5 when considering that the background color by default is white and when selecting a given color, the choice remains valid. In the matrix (*T*) the given coefficients are average means of the timings obtained through 3 runs of each scenario. Due to the fact that the execution of functionality may vary depending on the context (for example CPU load of the computer, or slow internet connection), the arithmetic mean allows the obtaining of an expected value for timing.

TABLE II.
DETAIL OF THE FUNCTIONALITIES.

ID	Name	Description	Code	Performance (in ms)
1	Rectangular selection	Allow to select a rectangular area	Sf1	20
2	Clear	Clear an area	Sf2	5
3	Selection of a color	Allow to select a color	Sf3	2
4	Zoom	Zoom in the drawing window	Sf4	40
5	Airbrush	Airbrush function	Sf5	30
6	Text	Allow to write text	Sf6	52
7	Right line	Allow to draw a right line	Sf7	23
8	Curve	Allow to draw a curved line	Sf8	43
9	Rectangle	Allow to draw a rectangle	Sf9	9
10	Ellipse	Allow to draw an ellipse	Sf10	12
11	Choice of color	Allow to choose a color	Sf11	7
12	File save	Allow to save the drawing in a file	Sf12	17
13	Print	Allow to print the drawing	Sf13	457
14	Fulfilling	Allow to fulfill a shape in a certain color	Sf14	345
15	New project	Allow to create a new drawing project	Sf15	12
16	Open project	Allow to open a project	Sf16	26

From the matrix *A*, the indicators of the scenarios and the functionalities described in section II can be computed (Table 4 and 5). The rank of the matrix (as constructed in Fig.3) is 3: this implies that none of the defined scenarios are redundant. According to the theoretical approach explained above, the complexity of each scenario as well as the weight of each useful functional element is calculated knowing that for other functionality the weight factor is zero. It is noticeable that Sf7 is the most important function in terms of weight, implying that it is indispensable. The complexity of the scenario S_2 is higher than the average which is also due to the subsequent use of Sf7 and the drawing of a pyramid. As a conclusion; it is recommended that the application also allows the user to draw pyramids, thereby reducing the complexity of the scenario.

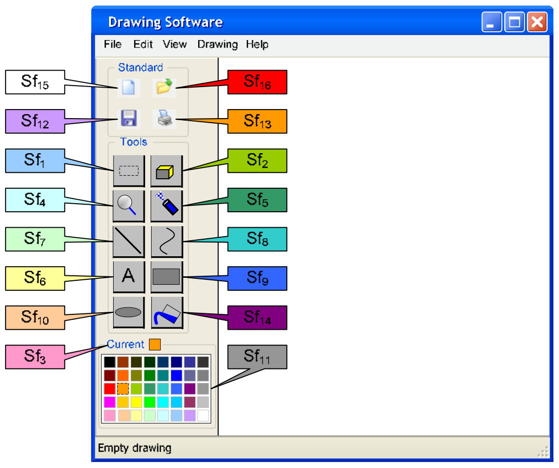


Figure 4. Example of the GUI of the case study.

TABLE III.
DESCRIPTION OF THE SCENARIOS.

ID	Description	Codification
1	Draw a black square on a background white color and print it	S1
2	Draw a pyramid with triangular basis (without representing the hidden sides) on a background blue color and print the drawing	S2
3	Draw a blue square on a background black color and print it	S3

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

$$T = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1012 & 0 & 843 & 0 & 2140 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 5115 & 0 & 0 & 0 & 1121 & 0 & 2140 & 758 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1012 & 0 & 1121 & 0 & 2140 & 758 & 0 & 0 \end{pmatrix}$$

Figure 5. Scenario versus functionality and time matrix.

Another noticeable aspect of the matrix concerns the **defaults settings**. As the setting of the color necessitates user action, one can envisage the modeling of such functionality. If the default color in the drawing software is blue and if the scenarios consist of drawing objects in black, then presetting black for the color will allow considerable simplification of the scenario.

TABLE IV.
SCENARIO'S INDICATORS.

Scenarios	Quantities	Time (ms)	Complexity	Literal expression of the scenario's complexity
S_1	18.75%	3995	1.732050808	√3
S_2	25.00%	9134	5.567764363	√31
S_3	25.00%	5031	2.645751311	√7

FUNCTIONALITIES INDICATORS.

Functionalities	Execution time (ms)	Weight	Literal expression
Sf7	23	5	5
Sf9	9	1.414213562	√2
Sf11	7	3	3
Sf13	457	1.732050808	√3
Sf14	345	1.414213562	√2
other		0	0

The conclusions of the methods applied during the design phase for this example are:

The functionalities Sf1, Sf2, Sf3, Sf4, Sf5, Sf6, Sf8, Sf10, Sf12, Sf15 and Sf16 were not to be implemented because their weight is null or neglectable.

A new high-level functionality which allows drawing 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 realize the scenarios) must be realized with fast response time and must be easy to locate on the GUI (Graphical User Interface) by the users.

B. Use case for the validation of a drawing software

In this section an end-user is confronted to a prototype of the software for the realization of the three scenarios developed in section 3.1, then the functionalities used by the user to conclude the scenarios are represented in a matrix form called *A'* and the time matrix *T'* depicted in Fig. 6.

These matrixes provide the following results for the indicators of the scenarios (see Table 6) and the functionalities (see Table 7).

Finally these results allow to compute the metrics developed in section II.A:

- Success rate: In that case, the user realizes only the Scenario 2 in a correct manner. In scenario 1 and 3 the user fails because he/she uses four times the function right line instead of rectangle. The success rate is therefore equals to 33.3%.
- Quantities: Here the quantities are the same in the experimental and in the theoretical approach even if different functions are used to achieve the target objective.

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

$$T' = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 9872 & 0 & 0 & 0 & 1120 & 0 & 3150 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 12340 & 0 & 0 & 0 & 2450 & 0 & 3230 & 1576 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 11230 & 0 & 0 & 0 & 4010 & 0 & 3340 & 1753 & 0 & 0 \end{pmatrix}$$

Figure 6. Matrix *A'* and *T'*.

TABLE VI.
SCENARIO'S INDICATORS (EXPERIMENTAL).

Scenarios	Quantities	Time (ms)	Complexity	Literal expression of the scenario's complexity
S_1	18.75%	14142	4.242640687	√18
S_2	25.00%	19596	5.567764363	√31
S_3	25.00%	20333	5.196152423	√27

TABLE VII.
FUNCTIONALITIES INDICATORS.

Functionalities	Execution time (ms)	Weight	Literal expression
Sf7	23	7.549834435	√57
Sf11	7	3.741657387	√14
Sf13	457	1.732050808	√3
Sf14	345	1.414213562	√2
other		0	0

- Quality: The quality is the difference between the ideal user using the software and the practical approach. The formula is given in (4)

$$Quality(S_i) = \frac{\sum_{j=1}^{m} a_{i,j}}{\sum_{j=1}^{m} a'_{i,j}} \times 100\% . \tag{4}$$

In that case *Quality*(S_1) = 50%, *Quality* (S_2) = 100% and *Quality* (S_3) = 56%.

- The effectiveness for each task is equal to 30.6% for S_1, 50% for S_2 and 37.3% for S_3.

- The efficiency (with realization times converted in seconds) is equal to 2.2% for S_1, 2.6% for S_2 and 1.8% for S_3.given by:

If the matrices *A'* and *A* are being compared (Fig. 3 and 4), the complexity of the scenarios 1 and 3 of the matrix *A'* is superior to those of the matrix *A*. This is the result of a wrong manipulation of the software by the user, because he/she did not use the function "rectangle" for the scenarios 1 and 3, he preferred to use four times the function "right line" to design a rectangle. The user did not realize 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 four times the functions “right line”. The comparison of the matrixes T and T' reveals the time a user takes to find the right functions.

The comparison of these two matrixes highlights that the user did not 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 or design a case-based of possible wrong manipulation for the tutoring of users.

The low scores of effectiveness and efficiency in this case can easily be explained by the implementation of functionalities that are not used (Sf1, Sf2, Sf3, Sf4, Sf5, Sf6, Sf8, Sf10, Sf12, Sf15, Sf16). In fact, only five functions are sufficient to achieve the tasks described in the scenarios which lowered the quantity factor implied in the computing of the effectiveness and efficiency. Without the useless function the effectiveness for each scenario will be 54.8%, 89.4% and 66.7% which is significantly higher than the obtained results.

C. Synthesis of the empirical evaluation and confrontation of the related evaluation method

For a robust evaluation it is important to clearly identify the tasks to be performed with the software and derive the scenarios. The scenarios are the basic instruction of the evaluation; therefore if they are too specific (or too nebulous) the risk is that no (or all) users will fail to complete the scenarios. The key to obtaining a good coverage and feedback in the empirical evaluation is to choose the scenarios so that the user needs to interact with different design-elements and controls. Other bias can occur during the empirical evaluation if the tester does not have the same characteristics as the future end users of the software. Depending on the effort allocated to the evaluation, it is important to run the experiments with multiple users. Because one user could find certain functionality necessary to complete a scenario, where another user does not. For example; if in this use case a second user had immediately found and used the “rectangle” function the efficiency for the first scenario would have been significantly higher. For a more robust evaluation the average value of the effectiveness for each scenario performed by different users gives a better indication whether the functionality is recognizable or not. One last bias that can appear is the measurement of the time of a function call. Therefore it is essential in the empirical testing to use adequate equipment in good working condition. If this is not the case, large discrepancies may occur in execution time (e.g. if another program is open with a high CPU load, or if the used function requires the loading and installation of an update package but the test computer had no internet connection).

As mentioned before, the different software evaluation methods uses varying data collections, differ in their degree of formalism and are rarely standardized. Thus it is difficult to compare them, but our approach is used to recognize the status and the actual problems of software in an (task oriented) objective, reliable, and valid way. It belongs to the descriptive evaluation methods [12] due to the fact that it requires a prototype and at least one user. Compared to log recording [16], the results of our approach are easier to interpret and provide a reasonable data objectivity as a basis for the decision of design changes. Nevertheless the bias around the test-user (knowledge, involvement, etc.) mentioned above can have a significant impact on the results of the matrix A' resulting in inappropriate recommendations for future software development. Nevertheless this drawback can be overcome with multiple users testing as in opinion based methods [11, 13, 15]. As our approach is based on a quantified matrix for the evaluation, it does not allow capture of the consideration and impression of the users as in walkthrough or inspections [22]. It derives the countered problems of a user completing a task compared to an ideal user. Thus the results of our approach are not suitable as inspection methods [23] for the recognition of problem lists.

Our approach allows for the making of a quantified evaluation of the operational way to complete tasks with a software product by the numeric assessment [7] of the scenario’s effectiveness. Thus proposed design changes and performance optimizations after the evaluation’s feedback make for a software product more suited to the end-user and the market.

IV. Conclusion

There are many evaluation techniques which can be applied in practice, although many of them need to entail effort and require special competences (experts). Other evaluations, like IBM’s measure of user’s satisfaction is based on: capability, usability, performance, reliability, instability, maintainability, documentation and availability, exist but the total quality management (TQM) allows much more progress in such area with the focus on customer, process improvement, quality culture, measurement and analysis, knowingly that their cost can only be supported by large companies. There is also a large science handling the internal quality of software with metrics such as: defect density, vocabulary, length, volume, level, difficulty, effort (from Halstead) or the Cyclomatic Complexity from McCabe [28] that is the count of the number of linearly independent paths through the source code ; or the function points, but they are rarely applied and are not concordant with the user’s view of quality. The method developed in this paper allows 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 at “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" [29], by revealing the needs of developing high level functionalities. The methods also propose a common base to avoid articulation problems between clients and developers as reported in [27]. The developers can visualize the user's issue, which is a frequently occurring situation but this methods does not research the field for intelligent software development [30].

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 understanding 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 the help of 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. In the classification scheme proposed by [8], this method can be placed upon the behavior observation because of the demanded user involvement. The data ascertainments (see Fig. 7)are objective if the specification phase of the product is sufficiently detailed (e.g. GUI, main functionalities, performance of functions, etc.) and if the tasks that have to be carried out with the product, are clearly defined.

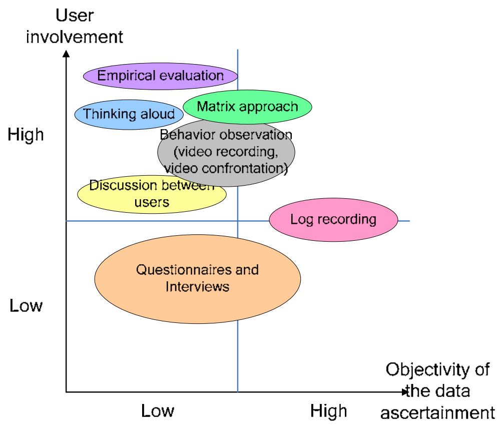

User involvement
High
Low
Empirical evaluation
Thinking aloud
Matrix approach
Behavior observation (video recording, video confrontation)
Discussion between users
Log recording
Questionnaires and Interviews
Objectivity of the data ascertainment
Low
High

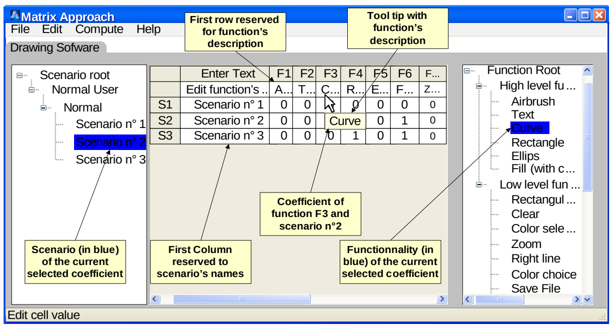
Figure 7. Data ascertainment proceedings for the evaluation.

The theoretical approach has been applied to a drawing software study. The results seem satisfactory and meaningful. Some defaults of the software have been identified during the design phase like useless functionalities, default settings and the implementation of high-level functionalities.

Moreover, a tool has being designed by the authors to compute automatically the weight of the function and the scenario's complexity.

In Fig. 8 the GUI of the tool which implements the proposed evaluation method is depicted.

In the near future, this method will be retained for the case of the development of domain specific software solutions (e.g. production planning, car manufacturer diagnostic systems) 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.



	Enter Text	F1	F2	F3	F4	F5	F6	F...
	Edit function's ..	A...	T...	C...	R...	E...	F...	Z...
S1	Scenario n° 1	0	0		0	0	0	0
S2	Scenario n° 2	0	0	Curve		0	1	0
S3	Scenario n° 3	0	0	0	1	0	1	0

Figure 8. GUI of the Software Evaluation tool.

This method fits particularly well in combination with designing methods of large software project as in [31] due to their complementarities. This case will take into account different user classes and also the role of the frequency attribute of the scenarios. This analysis will also be re-enforced by a practical cost prevision method. The variability of the software [32] will be taken into account because there is an ever-growing demand for variability (configuration) of software which is particularly relevant for branch specific products.

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

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

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

[4] Reiterer H., EVADIS II: A new method to evaluate user interfaces in Proceedings of the Human Computer Interactions, Cambridge University Press,1992, ISBN: 0521445914.

[5] Boehm, B.W., Software Engineering Economics. Englewood Cliffs, NJ: Prentice-Hall, Inc. 1981, ISBN: 0138221227.

[6] ISO DIN 9241-10, 11. Ergonomic requirements for office work with display terminals (VDTs): Guidance on usability. Berlin: Beuth, 1995

[7] Wottawa: Lehrbuch Evaluation. In Weidemann, B., Krapp, A.: Pädagogische Psychologie (4.,vollst. überarb. Aufl). Weinheim : Beltz PVU, 2001, ISBN: 3621275649.

[8] Palanque P.A., Bastide R., Design, Specification and Verification of Interactive Systems, 1995. Springer 1995, ISBN: 3211827390.

[9] Becker C.H., Using eXtreme Programming in a Student Environment, Master Thesis, Grin, ISBN: 9783640720040.

[10] Reiterer H., User Interface evaluation in Encyclopedia of Library and Information Science, 1997, ISBN: 0824720601.
[11] Kent A., Williams J.G., Encyclopedia of Computer Science and Technology,Volume 45, 2002, ISBN: 0824722981.
[12] Gediga G., Hamborg K.-C., Düntsch I., Evaluation of Software Systems, in Encyclopedia of Computer Science and technology, CRC Press, 2002, ISBN: 0824722981.
[13] Docherty P., System design for human development and productivity: participation and beyond, North-Holland, 1987, ISBN: 0444702512.
[14] Nielsen, J., Usability Engineering. San Francisco, Academic Press Inc, 1993, p. 195-198. ISBN: 0125184050
[15] Lin, H. X., Choong, Y., Salvendy, G., A proposed index of usability: a method for comparing the relative usability of different software systems. Behavior and Information Technology 16 (4/5), 1997, pp. 267-278. ISSN: 104492910.
[16] Hemmecke J., Stary C., The tacit dimension of user tasks: elicitation and contextual representation, in Proceedings of the 5th international conference on Task models and diagrams for users interface design, Springer, October 2006.
[17] Ammon U., An International Handbook of the Science of Language and Society, Walter de Gruyter, 2005, ISBN: 3110171481.
[18] Leimeister J., Huber M., Bretschneider U., Krcmar H., Leveraging Crowdsourcing: Activation-Supporting Components for IT-Based Ideas Competition, Journal of Management Information Systems (vol. 26 issue 1), 2009, doi:10.2753/MIS0742-1222260108.
[19] Nielsen, J., Usability engineering at a discount, In: Salvendy, G., and Smith, M.J. (Eds.), Designing and Using Human-Computer Interfaces and Knowledge Based Systems, Elsevier Science Publishers, Amsterdam, 1989, p.394-401. ISBN: 9780444880789.
[20] Carroll, J. M., Human-Computer Interaction: Psychology as a science of design. In: Annual review of psychology, Vol. 48, pp.61-83, Palo alto, CA: Annual Reviews, 1997. ISSN: 00664308.
[21] Jeffries, R., Miller, J.R., Wharton, C. & Uyeba, K.M., User interface evaluation in the real world: a comparison of four techniques, Proceedings of ACM CHI'91 conference on Human Factors in Computing Systems, 119- 124 (Association for Computing Machinery, New York), 1991, ISSN:0736-6906.
[22] Lewis, C. & Wharton, C., Cognitive Walkthroughs. In: Helander, M. G., Landauer, T.K., Prabhu, P. (Hrsg.), Handbook of Human-Computer Interaction. Amsterdam: Elsevier Science B.V., 1997, ISBN: 0444818626
[23] Nielsen, J., Usability Inspection Methods, New York. Wiley, 1994. ISBN: 0471018775.
[24] Tan D., Wandke H., Process-oriented user support for workflow applications, in proceedings of the 12th international conference on Human-computer interaction: applications and services, Springer, July 2007.
[25] Salvendy, G., Smith, M., Designing and Using Human-Computer Interfaces and Knowledge Based Systems, Amsterdam. Elsevier, 1989, pp. 394–401. ISBN: 04448807810
[26] S. Stowasser "Methods of Software Evaluation" in The International Encyclopedia of Ergonomics and Human Factors, CRC Press, 2006, p. 3249, ISBN: 04153043010.
[27] A. Azarian, A. Siadat "A Proposal of a practical approach for quantified quality software evaluation during the development cycle" in the 7th WSEAS Int. Conf. on Applied informatics and communication (AIC '07), Athens, Greece, August 24-26, 2007, p. 331-336, ISSN: 1790-51117.
[28] McCabe, A Complexity Measure, IEEE Transactions on Software Engineering: 315, December 1976.
[29] Suh N.P., Axiomatic design: Advances and applications Oxford University Press, 2001
[30] Youtian Q., Chaonan W., Lili Z., Huilai Z., Hua L., Research for an Intelligent Component-Oriented Software Development Approaches, Journal of software, vol. 4 n°10, December, 2009, doi:10.4304/jsw.4.10.1136-1144
[31] Lin Y., Wei S., Chi-long Z., Honglei T., On Practice of Big Software Designing, Journal of Software, Vol 5, No 1, p.81-88, January, 2010, doi:10.4304/jsw.5.1.81-88.
[32] Asikainen T., Männistö T., Soininen T. Kumbang: "A domain ontology for modelling variability in software product families". Advanced Engineering Informatics 21, 2007. ISSN: 1474-0346.

Armin Azarian is associate professor of engineering sciences. He received his PhD in 2009 from the Arts et Métiers ParisTech and the university of Karlsruhe. His major field of study consists of the development of diagnosis models and algorithms applied to the automotive industry.

Ali Siadat is an associate professor of computer and industrial engineering in the Laboratory of Industrial Engineering and Mechanical Production at Arts et Métier ParisTech, France. He received his PhD in robotics from Institute National Polytechnique de Lorraine, France in 1999. His research interests include artificial intelligence, information system and knowledge formalization applied to design and manufacturing system.