CREATIVE MATH. & INF. **Online version at** http://creative-mathematics.ubm.ro/
17 (2008), No. 3, 420 - 426 **Print Edition: ISSN 1584 - 286X Online Edition: ISSN 1843 - 441X**

Dedicated to Professor Iulian Coroian on the occasion of his 70^{th} *anniversary*

Quality evaluation of the software product approach

MARA HAJDU-MĂCELARU

ABSTRACT. We will provide in this paper an approach of evaluating the quality of the software product, using in the planning and developing phase of the product the two approaches that exists in the literature: Quality Function Deployment approach (QFD) and Goal Question Metric approach (GQM).

1. INTRODUCTION

The Goal Question Metric (GQM) approach is based upon the assumption that for an organization to measure in a purposeful way it must first specify the goals for itself and its projects, then it must trace those goals to the data that are intended to define those goals operationally, and finally provide a framework for interpreting the data with respect to the stated goals. Thus it is important to make clear, at least in general terms, what informational needs the organization has, so that these needs for information can be quantified whenever possible, and the quantified information can be analyzed a to whether or not the goals are achieved, [1].

QFD was developed initially by Akao in Japan in 1966. Akao (1990) defined it as *a method for developing a design quality aims at satisfying the customer and then translating the customer's demands into design targets and major quality assurance points to be used throughout the production stage*.

We will present an approach of evaluating the quality of the software product based on this two approaches.

2. QUALITY EVALUATION OF THE SOFTWARE PRODUCT APPROACH

Using the QFD algorithm approach we will plan the specification, the requirements that will be met in the final software product. In order to have a quality product customer requirements have to be considered. Let's suppose we have n customers for our product.

Notation: We will consider $U = \{u_1, u_2, \ldots, u_n\}$ as being **the set of our customers**.

We will apply the algorithm from the QFD approach. Each user defines a set of requirements that he will need to have in the final software product. The same requirement can be asked to be implemented in the final software product by different customers.

Received: 11.09.2008. In revised form: 03.05.2009. Accepted: 21.05.2009.
2000 *Mathematics Subject Classification.* 68N30.
Key words and phrases. *Software quality evaluation, metrics, requirements.*

Notation: We will consider $C = \{c_1, c_2, \ldots, c_m\}$ the total set of requirements defined by the set of customers, where m is the total number of requirements asked to be implemented in the software product.

Definition 2.1. *We define the Matrix of customer importance ratings of the requirements* $U[i,j]$, $i = \overline{1,n}$; $j = \overline{1,m}$; *define as follows:*

- $U[i,j] = 0$, *if the customer* u_i *did not ask for the requirement* c_j *to be implemented in the software product*
- $U[i,j] = 1$, *if the customer* u_i *ask for the requirement* c_j *to be implemented in the software product with the importance rate quantified through the numerical value 1. (the least important)*
- $U[i,j] = 2$, *if the customer* u_i *ask for the requirement* c_j *to be implemented in the software product with the importance rate quantified through the numerical value 2. (quite important)*
- $U[i,j] = 3$, *if the customer* u_i *ask for the requirement* c_j *to be implemented in the software product with the importance rate quantified through the numerical value 3. (important)*
- $U[i,j] = 4$, *if the customer* u_i *ask for the requirement* c_j *to be implemented in the software product with the importance rate quantified through the numerical value 4. (very important)*
- $U[i,j] = 5$, *if the customer* u_i *ask for the requirement* c_j *to be implemented in the software product with the importance rate quantified through the numerical value 5. (the most important requirement)*

Definition 2.2. *We define the* $VRT[j]$ *for* $j = \overline{1,m}$ *the Total Importance Rating of the requirement* c_j, $j = \overline{1,m}$ *as being the importance rate* c_j *for all the customers from the Set of our customers* $U = \{u_1, u_2, ..., u_n\}$. *The computation formula for the Total Importance Rating of the requirement* c_j, $j = \overline{1,m}$ *is:*

$$VRT[j] = \sum_{i=1}^{n} U[i,j], \quad j = \overline{1,m}. \tag{2.1}$$

We know that $VRT[j] > 0$ for all $j = \overline{1,m}$; because at least one user has asked for the c_j, $j = \overline{1,m}$ to be implemented in the software product otherwise it will not appear in the set of requirements.

The vector VRT elements sums represent the maxim quality degree from the user point of view. If all the requirements that the customers asked to be implemented in the software product are retrieve in the final product, and are correctly implemented we will have a 100% customers satisfaction.

Notation: GCm to be the Maximum Quality Degree from customer's point of view.

$$GCm = \sum_{j=1}^{m} VRT[j]. \tag{2.2}$$

Because in the real life, not all the customer's requirements can be implemented, rarely the software product will have a 100% customer's satisfaction. In many cases it may happen that not all the customer's requirements can be implemented mostly because of technical limitation, or because the time needed to implement a specific request is too long and it does not worth from cost of the product point

of view. Using the QFD approach, we will make a separation of the requirements, as follows:

(1) The degree of technical implementation difficulty of the requirement in the development phase of the software
(2) The relationship between the requirements, meaning which requirements support one another and which are in conflict. (Correlation Matrix from the QFD approach)

This is made by the development team that is going to make the implementation of the software product.

Definition 2.3. *We define the Vector of Implementation Degree of the Requirements* $R[j]$, $j = \overline{1,m}$*; define as follows:*

- $R[j] = 1$, *the requirement* c_j *is very easy to be implemented from the technical (development) point of view*
- $R[j] = 2$, *the requirement* c_j *is easy to be implemented from the technical (development) point of view*
- $R[j] = 3$, *the requirement* c_j *is achievable to be implemented from the technical (development) point of view*
- $R[j] = 4$, *the requirement* c_j *is difficult to be implemented from the technical (development) point of view*
- $R[j] = 5$, *the requirement* c_j *is very difficult/almost impossible to be implemented from the technical (development) point of view*

For the requirements for which $R[j] = 5$, in the VRT vector (Total Importance Rating vector), we make the change $VRT[j] := 0$, meaning that the requirement was eliminated due to the difficulty of its implementation.

Definition 2.4. *We define the Correlation Matrix* $MC[j_1, j_2]$, $j_1 = \overline{1,m}$, $j_2 = \overline{1,m}$ *as follows :*

- $MC[j_1, j_2] = 0$, *if* $j_1 = j_2$, *meaning that there is no way to have a correlation between the requirements as the requirements are the same*
- $MC[j_1, j_2] = 1$, *if requirement* c_{j_1} *supports the requirement* c_{j_2}, *meaning that there is no conflict between the two requirements*
- $MC[j_1, j_2] = -1$, *if requirement* c_{j_1} *does not support the requirement* c_{j_2}, *meaning that there is a conflict between the two requirements*

We need to solve the conflict between the requirements. For this, we cross all the matrix values and if the value is -1 (meaning that we have a conflict) we compute for both the requirements that are in conflict, the ratio between the Total Importance Rating of the requirement c_j which is $VRT[j]$ and the corresponding value of the requirement c_j from the Vector of Implementation Degree of the Requirements $R[j]$ and compare the two values that we receive. The requirement for which the ratio that is greater, that is the one that will remain, and in case the ratios are equal we compare the Total Importance Rating values for both requirements and the one for which the value is greater that will be the one that will remain.

For example, if we have that $MC[j_1, j_2] = -1$, then we compute and compare the following ratios:

$$VRT[j_1]/R[j_1] \text{ with } VRT[j_2]/R[j_2]$$

```
If VRT[j1]/R[j1] > VRT[j2]/R[j2]
   then
   Begin
      VRT[j2] := 0;
      MC[j2, j1] := 0;
   end ;
```

Meaning that the c_{j_2} requirement was eliminated as the c_{j_1} requirement is more advantageous to remain. Also the attribution $MC[j_2, j_1] := 0$; was made, as the matrix is symmetrical to the diagonal.

```
If VRT[j1]/R[j1] < VRT[j2]/R[j2]
  then
  Begin
      VRT[j1] := 0
      MC[j2, j1] := 0;
  end ;
```

Meaning that the c_{j_1} requirement was eliminated as the requirement c_{j_2} is more advantageous to remain. Also the attribution $MC[j_2, j_1] := 0$; was made, as the matrix is symmetrical to the diagonal.

```
If VRT[j1]/R[j1] = VRT[j2]/R[j2]
   then
   Begin
      if VRT[j1] >= VRT[j2]
         then VRT[j2] := 0
         else VRT[j1] := 0;
      MC[j2, j1] := 0;
   end;
```

Meaning that in case the ratios values are equal we compare the values corresponding values of the c_{j_1}, and c_{j_2} requirements from the Vector of Total Importance Rating, and the one with the greater importance remains. Also the attribution $MC[j_2, j_1] := 0$; was made, as the matrix is symmetrical to the diagonal.

As it can be noticed above, we have assign the value 0 in the Total Importance Rating vector - $VRT[j]$, $j = \overline{1, m}$, for the requirements that were eliminated, so because of technical limitation not all the requirements will be implemented.

Definition 2.5. *We define the Quality Implementation Degree - Gci in the software product, based on the customer requirements and technical limitation, define as follows:*

$$Gci = \sum_{j=1}^{m} VRT[j]. \tag{2.3}$$

Definition 2.6. *We define the achievable quality degree from the customer requirements and technical limitation that can be implemented in the software product p_u and we compute it as follows:*

$$p_u = \frac{Gci}{GCm} \cdot 100\%.$$

As a conclusion, using the QFD approach we have create a model that computes the achievable quality degree, from the customer's and development perspective.

But there is no guarantee that the achievable quality degree will be the one that will be realized in the final product, after the implementation of the software product. We know which will be the user satisfaction if all the functionalities will be implemented correctly. But it does not assure that the functionality will be 100% correctly implemented and will work 100% for all the supported input values.

Using the GQM approach we will provide a method of computing the quality percentage of the overall software product based on customer requirements, technical limitation and functional correctness.

Using the GQM approach, we will consider that in our case the targets that will be measured will be the software product functionalities (requirements). So, we will suppose that we have to evaluate the overall functionality percentage of the software product based on the functionalities of the software product. As above we discussed about requirements, we will consider that each requirement is functionality in our product. Purpose: Which is the functional correctness percent of a software product for each functionality?

1. Conceptual level (Goal) First step is to establish the product functionalities / requirements. Let's suppose that we have a software product with p functionalities / requirements. We will need to evaluate the functionality percentage for each of the functionality in part. So we have p goals:

Goal 1 -functional correctness of the functionality_1
Goal 2 - functional correctness of the functionality_2
... ...
Goal p - functional correctness of the functionality_p

2. Operational Level (Question) A set of questions is used to characterize the way the functional correctness is achieved. The questions try to characterize the component based on the specification, input values and other necessary characteristic to obtain the most suitable characterization of the functional correctness of the functionality.

Functionality_1	...	Functionality_p
$\{(car_{11}, p_{11}), (car_{12}, p_{12}), ...\}$	...	$\{(car_{p1}, p_{p1}), (car_{p2}, p_{p2}), ...\}$

Notation: The pair (car_{ij}, p_{ij}) means car_{ij} the characteristic j of the Functionality_i, and has the importance percent p_{ij}, $i = \overline{1, p}$;

In our case, for a better precision in the functional correctness evaluation of the functionality, for each goal in par is defined a set of characteristics, and also a importance percent pij of the characteristic in evaluating the goal. (How much

the achievement of the characteristic counts in the establishment of the functional correctness of the corresponding functionality) .The sum of all the importance percentage for a specific Functionality_i is 100% , $i = \overline{1,p}$.

3. Quantitative level (METRIC) A set of data is associated with every characteristic in order to answer in a quantitative way. For each characteristic in part, we choose a set of metrics used to evaluate the characteristic. So, we have:

Functionality_1	...	Functionality_p
$\{(car_{11}, p_{11}), (car_{12}, p_{12}), ...\}$	...	$\{(car_{p1}, p_{p1}), (car_{p2}, p_{p2}), ...\}$
$\{(m11_1, m11_2, ...), (m12_1, m12_2, ...), ...\}$	...	$\{(mp1_1, mp1_2, ...), ...)\}$
$\{f_{11}, f_{12}, ...\}$	...	$\{f_{p1}, f_{p2}, ...\}$

Notation: $(mij_1, mij_2, ...)$ are the metrics used to evaluate the characteristic car_{ij}, and f_{ij} is the quantitative answer to the question, in our case the functional correctness percent of the Functionality_i relative to the characteristic car_{ij}.

As an answer to our goal "Which is the functional correctness percent of a software product for each functionality?, we have the following definition and formula.

Definition 2.7. *The functional correctness percentage of the Functionality_i , $i = \overline{1,p}$ is:*

$$pf_i = \sum_{j=1}^{n_i} (f_{ij} \cdot p_{ij}),$$

where n_i is the number of characteristics that characterize the Functionality_i.

We can use the GQM model in our approach by considering a requirement c_i to be corresponding to a functionality_i, $i = \overline{1,m}$.

The achievable quality degree that could be realized in the software product if all the requirements / functionality were 100% functional correctness was:

$$p_u = \frac{Gci}{GCm} \cdot 100\%.$$

So, if the functional correctness percentage is $pf_i = 100\%$, meaning that all the requirements were implemented without errors, than the overall quality percent of the product based on customer requirements, technical limitation and functional correctness is equal with the achievable quality degree. But unfortunately, in software industry the realization of functionalities to be 100% functional correct is almost impossible. We can compute based on the **Quality Implementation Degree** formula, **achievable quality degree** formula, and **functional correctness percentage** formula the overall quality percent.

We can now compute the **Overall Quality Percent**, for short Pc, of the software product based on the customer requirements, development limitation, and functional correctness, as follows:

$$Pc = \frac{\sum_{i=1}^{m} pf_i \cdot VRT[i]}{GCm} \cdot 100,$$

where pf_i is functional correctness percentage of the Functionality_i; $VRT[i]$, $i = \overline{1,m}$ is the Total Importance Rating vector; and GCm represents the Maximum Quality Degree.

We have present an approach that computes the overall quality of the software product based on the two approaches from the literature GQM and QFD.

3. Conclusions and future developments

Following the steps of this approach, the quality of the software product can be evaluated taking into account the user requirements, and development limitation.

For a software product, it is very important to reach an acceptable level of quality, but another factor to which we need to pay attention is the time/costs. How much will cost to reach an acceptable level of quality or a higher level? Does it make sense to improve the quality of the product if the costs/time is high? This approach can be improve by adding time/cost metrics in the development phase that can provide information about the quality level that was reached at a certain point in the development with the exact costs, and if want to reach higher levels of quality which will be the costs.

References

[1] Basili, V. R., *The Goal Question Metric Approach*, ftp://ftp.cs.umd.edu/pub/sel/papers/gqm.pdf
[2] Shahin, A., *Quality Function Deployment: A Comprehensive Review*, http://www04.homepage.villanova.edu/jessica.byrnes/Quality%20Function%20Deploy/shahin.pdf

North University of Baia Mare
Department of Mathematics
and Computer Sciences
Victoriei 76
430122 Baia Mare, Romania
E-mail address: macelarumara@yahoo.com