20050627- SERP'05

QFD applied to software development

The 2005 International MultiConference In Computer Science & Computer Engineering

Las Vegas Nevada, USA, June 27-30, 2005

Eng. Lélis Tetsuo Murakami
University of São Paulo
Polytechnic School

Prof. Edison Spina, PhD
University of São Paulo
Polytechnic School

Prof. José Sidnei C. Martini, PhD
University of São Paulo
Polytechnic School

Abstract — *Since the initial development in Japan, in the late 1960's and early 1970's, the industries had used QFD – Quality Function Deployment a tool of TQM – Total Quality Management, to products manufacturing. In 1980's this method was rapidly spread through out the USA and several countries in Europe, Australia and Brazil. The fundamental aim of this technique is to obtain and understand the needs of the customers and transform them into a detailed specification that is used along the process of the product manufacturing. The idea of using QFD technique as it is, to software development is not easy or convenient, since there are differences between product manufacturing and software development. However, it seems like a good procedure, to understand the concepts of QFD and apply them to software development where appropriate, in conjunction with other strategies such as CMM – Capability Maturity Model. This paper is an attempt to describe how to implement quality into software using the available tools. At the end of this paper a real case experience in a medium size automotive industry is presented.*

Keywords: QFD, SQFD, TQM, Software Quality.

1. Introduction

An efficient and maybe the best method to determine the customer requirements of a specific product or service was created in Japan in the late 1960´s and early 1970´s. It was named QFD – Quality Function Deployment and is part of a TQM – Total Quality Management. During the decade of 1970, this technique was improved and became a very strong and reliable tool, spreading across the world, more significantly in the USA in the following decade. This method aims essentially to collect and understand the customer needs or expectations about a specific product, transforming them into a list or requirements used along the product manufacturing process. As a result of this strategy, a reliable product with desired quality is obtained providing the customer satisfaction. QFD goes more deeply trying to find out more requirements not related by customers through interviews, but inferred during the process, in order to provide some characteristics to the product that will cause a positive surprise to customers, a very useful strategy to improve the customers' satisfaction. The same process could be used to establish quality in services. This method could be applied to software development, but some considerations are required. The purpose of this paper is to describe alternatives to improve quality of software, to comment the effectiveness and applicability of several techniques and then, at the end suggest a practical way to guarantee the objectives.

2. The practice of QFD

QFD history was originated in the late 1960 and early 1970 in Japan. At that time, the Japanese industries were performing the "copy" manufacturing that is, use existing products, to create another one. In this *status quo*, QFD concept was created to a new product development. Akao & Manzur in [1] describe in details the history of QFD evolution referring to the past, present and future of this method. Akao after having created this method, observed these two points:

- The importance of quality in the product manufacturing is a fact. However, there was no available bibliography about the theme and how it could be provided;
- The manufacturers had already used worksheets in the quality control process, but after the product manufacture.

and concluded with the question:

Why could not we note the critical points of quality control process prior to production start up?

The term QFD – Quality Function Deployment was originated from the Japanese term 品質基能展開 (ひんしつきのうてんかい) or (HIN-SHITSU KINOU TENKAI). The translation of this term was modified over the time by the author. In 1972, Akao translated it as Quality Function Evolution and in 1983, Masaaki Imai suggested other translation arguing that TENKAI would be better translated as Deployment, and Akao agreed with him.

QFD method was spread significantly through out the USA and other countries beginning at the 80's. A survey sponsored by Tamagawa University and University of Michigan on recent QFD applications shows that 68.5% of American companies use QFD in their activities and only 31.5% of Japanese companies use it.

There are many issues in the literature about QFD and its application in product manufacturing. Lai-Kow Chan and Ming-Lu Wu [2] made an important work compiling many articles about the subject, revising 650 papers about QFD from several sources.

The bibliography of QFD applied to software development is poorly related. There are differences in producing products and producing software, as well as there are differences to specify products and specify software. Moreover, a customer requirement of a product is clear, however, it could be subjective in case of software. A software could attend a process need and also the user. In this case we can say that the user is attended indirectly and so, the QFD method could not be easily applicable. Barnett & Raja in [3] justify the few bibliography of QFD applied to software development arguing that the companies after using the method obtain significant market advantage and so, they maintain the QFD experience proprietary, hiding the knowledge to their own use.

Other authors such as Mercedes Ruiz, Isabel Ramos and Miguel Toro in [4] assume a different point of view referred to software development process, suggesting a strategy of quality assurance, based on modern practices of system modeling.

3. Quality in Software Development

A software as a product could be considered in the same way as a manufactured product. Its quality could be perceived as it performs its functionalities with efficiency under the customer point of view. Nevertheless, the list of functionalities could be large and so, one difficulty is to guarantee that all of them and its combinations will work. The traditional software development process suggests the use of several strategies of test to verify if all functionalities are working well. However, the application of this procedure becomes very hard when the number of functionalities or its combinations assume very high values. Therefore, the guarantee of software quality is inversely proportional to the number of functionalities or its combinations. If a given software should sum 2 numbers, probably it will do it well presenting a reliability of 100%. However, the Windows XP software, which has millions of functionalities will have several operational difficulties as predictable. Certainly this software was made by groups of many software engineers and programmers and the probability of problems occurrence is high and it depends on the software 'gigantism'. Although the problems exist, it is not correct to imply that the quality was refused. Referring to software development, there is another point to consider, that is, if some software has a wide spread in the organization and has poor quality it will cause undesirable damage as a presumable consequence.

4. Quality Applied to Software Development Process

If there is no possibility to inspect all range of software functionalities there is a strategy to guarantee that part of it will work correctly.

This idea is quite usual and similar to "divide and conquer" that is, let us divide the problem into sub problems and try to inspect them. We can divide the original problem into many levels and call them components or better software component. This later could be considered as part of the software, which has general functionality and could be used as part of other software. The maturity of these components could be provided along the time, in a continuous improvement as suggest by CMM Methodology. Banker e Kauffman [5] argued that the level of reuse of a component is a way to measure the quality of software development process. If we invest in component manufacturing, we are improving the reuse of them and as a consequence we are improving the quality of the software development providing high quality software.

Barnett & Raja in [3] mentioned some aspects that would not be so correct. They say that the current techniques of software development do not have a methodology to specify the user requirements and more, they emphasize that the actual techniques are inefficient to transform the needs of users into software specifications. If this is true had we developed bad software until now? The answer is no, of course. The *use case* techniques from Software Engineering correctly model the required functionalities of software, transforming them into software specifications that are used by developers. There is another complicate point when the authors affirmed that the user requirements were collected through interviews and the software specifications were modeled according to the developer point of view. There is no problem with the way the needs are evaluated through interviews. It is done by QFD method too. The mistaken point would be the affirmation that the software is modeled by the developer point of view. It is not true. The software should be developed according to the point of view of the end user and should be approved by him. They also say that software development does not have repetitive production process. Again, it is not true. Many pieces of code or functions or templates or modules have general use and are replicated as many times as needed in a repetitive production process. This procedure is becoming more and more frequent. As a way to make a comparison let us suppose a car factory. They produce cars assembling several parts of them. In the same way, we produce software joining parts of them and it is common to mention as software factory.

Another paper mentioning quality in development process is written by Mercedes Ruiz, Isabel Ramos and Miguel Toro [4]. The main concept of this paper is to valuate the quality through statistical process using a dynamic integrated framework, simulating the software development and maximizing the capability of the process.

5. Relevant Aspects

There is no doubt that there are enough differences between products and software manufacturing making the use of QFD impracticable in completeness. However many concepts of this technique are useful, mainly to identify the requirements. The matrix HOQ – House of Quality could be used partially to identify and prioritize the software requirements under the user point of view. Although QFD is widely spread and successful when applied to industries, it does not mean that this methodology can be applied to software development with the same success. More than this, it is not true that the software development will be damaged if QFD is not applied. Software engineering and OO methods always had aimed the excellence in software design. These techniques working together had provided quality in software development through the large use of methods, patterns and management process. The strategy of investing in software development process to obtain quality seems to be adequate and the most successful.

6. Focusing Quality

Software quality is a characteristic that could appear differently and depends on several factors. The quality level established would be a strategic parameter. However, some procedures should be adopted in order to maximize the quality level.

6.1 Patterns and Reusability

The use of patterns in the software development process has several positive aspects. The development process got high performance because no time is lost to think and decide which style will be used or what language will be selected or how the code should be developed. The more patterns are

clear the more high performance is obtained. When we talk about patterns it is convenient to select tools and communication language to promote the link between all actors of the development process. Well done documents of system, follow-up, failure statistical analysis, support, observed failures etc. are necessary to manage the organization process. Coding patterns also drive to improve programming activities and certainly promote the economy of many hours of maintenance. The most important advantage of using patterns is the transparency of the development process, making this activity independent of the programmer. It is strongly recommended to adopt a hard administration to implement a development procedure based in patterns. If there is no adequate control, the results obtained could be inefficient. The IT area has become one of the most important and strategic department of every organization which intends to be competitive in the market. Actually, it is not imaginable a company without a strong IT department manipulating operational data and providing information to the company's senior managers, and feeding the DSS – Decision Support System of the organization. To improve the risk analysis process and the decision making, a structured process is necessary to manage and control the IT activities of the company in order to guarantee the investment payback and improve the organization process. This new IT *wave* is called IT Governance. There is no doubt that the IT technology is needed with all of its paraphernalia such as PC, networks, hubs, servers, internet, firewall etc. The point of discussion is how each company manages his technical tools. That is why a model like ITIL – Information Technology Infrastructure Library has become widely popular in the IT world. However, few companies have adopted it, but the concern exists and the diversity of methods is a fact. ITIL is a collection of best practices and process of IT services and was created in the 80's by CCTA, an agency of British government. It covers many areas but has the focus in quality of IT services management and, if well applied improve the efficiency in the IT management. Another model called CobiT is a guide to the IT management proponed by ISACF – Information Systems Audit and Control Foundation, (www.isaca.org). The CobiT practices are recommended by IT specialist because it permits the optimization of investment and provide a way to measure the results. CobiT is an independent platform and is oriented to business providing detailed information for business oriented objectives. We must point to another methodology for IT management called Rational Unified Process – RUP. It is a software engineering process and has a main characteristic of delegating tasks and responsibilities between the development group or development company. The main purpose of this method is to guarantee the production of software in time with planned cost and with the quality required by the customer. The RUP development life cycle is divided into four phases named: Inception, Elaboration, Construction and Transition.
CMMI – Capability Maturity Model Integration is a reference table used to evaluate the software development maturity. The SEI – Software Engineering Institute from Carnegie Mellon, USA, since 1986 has been developing and improving this evaluation table, which first version was called simply CMM – Capability Maturity Model for Software. The objective of this model is to provide for the organizations a method of measuring maturity in the software development process as well as to establish continuous improvement programs. We can define maturity as the capacity of replicating the success now obtained in future projects.

All these methodologies strongly focus the IT activities management and as a result we obtain the quality assurance of software. As much the companies structure themselves and adopt methods and patterns, the probability of component reuse will increase. Development strategies that focus continuous improvement of software development will quickly reach the high level of CMM.

6.2 SOFTWARE QUALITY IMPLEMENTATION STRATEGY

The first point that arises when software quality is being considered is the strategy to be adopted. This later could be unique in the organization or not but one fact is real and doubtless: assuming maximum quality and reliability for projects are impracticable. Instead, good quality and good reliability is better. To illustrate this idea we may think of an aircraft which risk of accident we want to establish as zero. In this case this plane should never fly and obviously, this is undesirable. The reliability value usually stated in commercial aircraft is less than 10^(-9) failures per hour. What can be considered are levels of failures, which should be managed like in railway and aviation areas where failures are projected to assume small values. One more point is important and shall be considered: assuring a certain quality level requires time and resources and referring to software, we must consider the available time to develop it. In many cases, the available time for software development commands the quality level and in the same way, available resources would establish the quality. In any case, despite all these difficulties it seems to be important to maintain the user requirements with no changes. The strategy to avoid these difficulties could be managed through taking good planning, methods, improvement of performance, components reuse and all kinds of efforts to maximize productivity and minimize the consequences assuring the quality level projected. Actually, companies should provide good productivity with adequate quality level as a way to become competitive. One strategy to implement quality in software is to invest in efficient IT activities management such as CMMI, PMI, COBIT, RUP, etc. as cited before which provide the necessary organization to execute software development activities with patterns and methods. The use of UML – Unified Modeling Language is efficient and promotes good and reliable communication between the actors involved in the development process which makes the quality level up. As pointed out before, quality requires resources and what should be done if a company does not have enough resources? Working with no quality strategy is out of question, however, a good strategy could be made as follows:

a) Use existing IT management practices defining actors, roles, disciplines, etc.

b) Describe advantages and disadvantages of the application of each roles and cost associated.

By doing so, it is possible to analyze and search what roles the company can perform, according to its available human and financial resources, assuming the risk of non executed activity and controlling the consequences. This is preferable than quality ignoring strategy and will reduce the non desirable situation. The companies will win the market competition as successful as they invest in generic software components which are improved every time when reused, making the continuous improvement and transforming them in doubtless high quality software pieces. The more components are created, the more improved will be the application development. In the near future, people will assembly software instead of coding it. Many templates will help the developer to quickly create reliable applications using powerful wizards in a very short time. Quality in this scenario is an inherent characteristic and nobody is worried about it. One more question arises when we talk about software quality. Frequently hardware, software and peopleware are not equalized in terms of quality. I introduce here the concept of HSP normalization, that is, the idea of managing the compatibility in these three axes. It is a simple question but very often developers forget this issue and make applications with unbalanced HSP. To normalize means to increment conveniently the effort along these axes. For instance, it is not necessary to use parallel processing hardware if the software does not use parallel skills in the software coding. What to say about a complex algorithm that only the author could understand and use? What is the need to research and implement a very complex algorithm to get the optimal solution if the entry data have a percentual error of two digits? In this aspect, the quality is not in the software but in the good sense of developers who are supposed to be regardful to these considerations.

7. QUALITY IMPLEMENTATION IN THE SOFTWARE DEVELOPMENT – A REAL CASE

Quality issue was the focus of a recent project in a medium size automotive company in Brazil belonging to an international holding Co. The company has up to 770 employees and had a gross income of about US$ 56 million in 2004. The IT infrastructure is composed of 20 servers to control Databases, Intranet, Mail, ERP, Applications, Metaframe and Product Engineering. The network has a set of 280 PCs and 30 notebooks. The upper bound of server processor is Xeon 3.06 and Pentium IV 2.8 to the PCs, both using Windows 2000 operational system.

The IT team has 15 outsourced professionals where 8 of them are directly involved in software development. As part of an international group, the company administration should comply with the group regulations and patterns and follow the recommendations resulting from American laws. The assurance of quality to this company was implemented considering the existing best practices of IT administration, customizing them to the company available human and financial resources including the compliance with internal regulations and Sarbanes-Oxley law (2002). This later was created in the USA to increase the corporate governance, giving more responsibility to directors over financial controls and report delivery procedures.

The necessity of this project was cause by the objective of the company to regulate the development process which main guidelines are:

- To comply with internal regulations of software development;
- To improve the performance of software development activity;
- To make the process of development transparent to everyone;
- To attend the project timetable established;
- To implement software quality.

The IT team of the company has used until now several methods to elaborate the software projects to supply the demand. However, it still means a difficulty related to the software development which did not have the same systematic procedure, causing inefficiency in maintenance process and project control. This variation of procedures generates increases of rework and costs of maintenance and development. As the tests criteria and development parameters were established in different ways, the evaluation of quality pattern and the attendance to the user requirements led them to wrong judgments, mainly due to the difficulty to implement a reasonable metric system. Although the procedures of development adopted by the company were consistent, there was a necessity to reorganize and take similar development procedures, based on current IT practices such as CMM – Capability Maturity Model and UML – Unified Modeling Language.

To elaborate the methodology, all the company's internal regulations and procedures such as Sarbanes-Oxley, IPS-Information Policy System, Internal Control etc. were analyzed focusing IT topics. In terms of practices, RUP – Rational Unified Process, PMBoK from PMI, CMMI, COBIT and ITIL were also considered.

The basic guideline for this project had consisted in the attendance to internal regulations and the compliance with the Sarbanes-Oxley law. The methodology developed was customized to the company real conditions referring to the available human and financial resources. Considering these assumptions, the objective was to structure a methodology which makes the best IT governance possible. The strategy was to extract from the practices mentioned before, the relevant points that will lead to the achievement of targets.

Based on RUP methodology, we used the same concept of software project life cycle, dividing it into four phases called Inception, Elaboration, Construction and Transition, involving several disciplines. Each phase is composed of various disciplines and each of them is a set of activities. For each activity the actors involved, their correspondent roles, the responsible actor and the artifacts to be constructed were defined.

The disciplines considered were:

- Services request;
- Requirement Management;
- Project Management;
- Configuration Management;
- Development;
- Tests;
- Quality.

In order to provide more flexibility to role attributions and activity executions for IT members, each of the activities was listed and described, mentioning its meaning, the actor who was designated to do it, costs involved and the results expected. Using all this information, the company may evaluate each activity and the correspondent cost, to decide if, this activity at this cost, which will provide certain results, is interesting at that moment.

The quality requirement was intensively considered in the activity of system specification, using the concepts of QFD to identify the user (or process) requirements and transforming them into project specifications. The methodology established had used intensively regulations and patterns of software development process, improving the software quality. The reusability was also focused on this methodology, developing generic components when possible and creating the continuous quality improvement process.

8. CONCLUSIONS

The concept of quality in software could not be considered as in the product manufacture. There are several differences between producing a product and coding a program. To find out the best color of a product is a simple task however, to decide the best way to code a function is not so easy and it requires a deeper analysis. After all, it is not true that this is the best function. We can not prove that another better function does not exist. It is necessary to distinguish satisfaction and accordance. A given software could be in accordance with the specifications established but it can make the user unsatisfied for any reason. In this case, what can we say about the software quality? It seems to be a usability question and so, should we say that software with bad usability has bad quality? What is the convenient level of usability? What to say about usability and quality synchronism? All these topics require more detailed researches and may be treated in a future survey. The direction of software quality as made in product manufacture is therefore questionable. This is because a software is requested by a customer to perform some action and it is expected not to cause any surprise if the software work successfully. The guarantee to assure the software functionality as specified, can be obtained through appropriate care and accuracy in the development process, focusing also usability aspects. Nevertheless, it is not enough to consider methods, patterns and resources. More than this, it is necessary that all IT members be involved in the *quality up* process having the same objective of improving quality and performance of software development ever more.

Finally, there is no doubt that we should reach high levels of software quality and reliability and we can get it using practices, methods, patterns, components, efficient and well documented coding, statistical analysis of fault tolerance, user complaint management etc.

9. REFERENCED BIBLIOGRAPHY

[1] Yoji Akao & Glenn H. Mazur "The leading edge in QFD: past, present and future" International Journal of Quality & Reliability Management Vol. 20 No. 1, 2003 pp. 20-35

[2] Lai-Kow Chan *, Ming-Lu "Quality function deployment: A literature review", *European Journal of Operational Research 143 (2002) 463–497.*

[3] William D. Barnett and M.K. Raja "Application of QFD to the software development process" International Journal of Quality & Reliability Management, Vol. 12 No. 6, 1995, pp. 24-42

[4] Mercedes Ruiz, Isabel Ramos And Miguel Toro "A Dynamic Integrated Framework for Software Process Improvement" Software Quality Journal, 10, 181–194, 2002

[5] Banker, R.D. and Kauffman, R.J., "Reuse and productivity in integrated computer-aided software engineering: an empirical study", MIS Quarterly, Vol. 15 No. 3, 1991, pp. 375-401

[6] Yoshizawa, T., Akao, Y., Ono, M. and Shindo, H. (1993), ``Recent aspects of QFD in the Japanese software industry'', Quality Engineering, Vol. 5 No. 3, pp. 495-504.

10. CONSULTED BIBLIOGRAPHY

Step by Step – QFD Customer-Driven Product Design – 2nd Edition – John Terninko, 1997

Abrahamsson, Pekka, "Commitment to Software Process Improvement—Development of Diagnostic Tool to Facilitate Improvement", Software Quality Journal, 8, 63-76, 1999.

Omar A.R., Harding J.A. and Popplewell K., "Design for customer satisfaction: an information modelling approach", Integrated Manufacturing Systems 10/4 [1999] 199±209.

Reiblein S., Symons A., "SPI: 'I can't get no satisfaction' – directing process improvement to meet business needs", Software Quality Journal 6, (1997) 89–98.

Matthew L., Siddiqi A., "Can the case for CASE technology be advanced by Process Improvement?" Software Quality Journal 7,(1998) 3–10.

PROCEEDINGS OF THE 2005 INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING RESEARCH AND PRACTICE

SERP'05

Volume II

Editors

Hamid R. Arabnia
Hassan Reza

Associate Editors

Lawrence Chung, Juan J. Cuadrado-Gallego
Sergiu Dascalu, Emanuel Grant,
Frederick C. Harris Jr., Michael Hinchey
Deborah Kobza, Youngsong Mun, Roy Sterritt
Nary Subramanian

Las Vegas, Nevada, USA
June 27-30, 2005
©CSREA Press

This volume contains papers presented at The 2005 International Conference on Software Engineering Research and Practice (SERP'05). Their inclusion in this publication does not necessarily constitute endorsements by editors or by the publisher.

ISBN: 1-932415-49-1, 1-932415-50-5 (1-932415-51-3)
Printed in the United States of America

CSREA Press
U. S. A.