The leading edge in QFD for software and electronic business

Georg Herzwurm
Dresden University of Technology, Dresden, Germany, and
Sixten Schockert
University of Cologne, Cologne, Germany

Keywords Computer software, Quality function deployment, Electronic commerce, Quality, Planning

Abstract In this article an overview of the state-of-the-art of Quality Function Deployment (QFD) in software development or also called software QFD is given. The differences between classic QFD in manufacturing industries and software QFD are described. Following that certain software-specific QFD models (Zaltner, Shindo, Ohmori, Herzwurm and Schockert), which can be considered the most appreciated ones in theory as well as in practice, are introduced. Experiences in practice with these software QFD models are presented as well. Finally, through explaining the main principles of a special QFD variant for e-commerce, called continuous QFD (CQFD), the article will show that QFD is suitable for planning electronic business applications as well.

Introduction
More than 30 years after the legendary NATO conference in Garmisch-Partenkirchen which is celebrated as the moment software engineering was born, many users are still unsatisfied with the software delivered to them. Existing software requirements engineering methods pursue the primary goal of transforming user requirements that exist in a natural language into formal specifications and models (e.g. object models) to serve as a basis for design and implementation (e.g. Davis, 1993). Customers who are not familiar with these technical models created for specialists and who do not want to invest the necessary high learning efforts, however, only comprehend such models to an unsatisfactory degree. This leads to constant communication problems with software developers. “New” requirements engineering techniques such as rapid prototyping (e.g. McConnell, 1996) or joint application development (e.g. Wood and Silver, 1995) point the right way, employing the concepts of teamwork and user involvement. However, they have not been able to prevent many software development projects from still being rather geared towards what is technically feasible than what is needed by the customers. Therefore, modern methods are required which are focused less on formalizing and specification of customer requirements than on recognizing and satisfying customer needs.

Quality Function Deployment (QFD) is a method developed in Japan that fulfills these demands (Akao, 1990). QFD provides a systematic but more informal way of communication between customers and developers. Additionally, QFD is aimed at products that present not all technically possible
characteristics but those that customers demand (“fitness for use”). The entire QFD process is carried out by a cross-functional team, including customer representatives, and an experienced moderator. Especially in the automotive industry, Japanese companies have gained a competitive edge not least by systematically employing QFD, but also in software development QFD is constantly gaining more attention – far exceeding Japanese territory. At QFD conferences, the following firms among others reported employing QFD in software development projects: DEC, Hewlett-Packard, Hughes Aircraft, IBM, Motorola, NTT Data Corporation, Roche Diagnostics, SAP, Siemens, Texas Instruments, Toshiba, Vodafone (formerly Mannesmann Mobilfunk), and Unisys.

In this article we will give an overview of the state-of-the-art of QFD in software development or also called software QFD. At first we briefly describe the differences between classic QFD in manufacturing industries and software QFD. Following that we introduce certain software-specific QFD models (Zultner, Shindo, Ohmori, Herzwurm and Schockert), which can be considered the most advanced ones in theory as well as in practice. Practical experiences in practice with these software QFD models are presented as well. Finally, through explaining the main principles of a special QFD variant for e-business, called continuous QFD (CQFD), we will show that QFD is suitable for planning electronic business applications as well.

Differences between software QFD and classic QFD

The basic purpose of product development is universal: customers have certain demands concerning the use of products, to be satisfied by development in a complex process, which has to consider time, costs and quality. In principle, QFD in its classic form can be applied to software products as well (Barnett and Raja, 1995). Two essential differences, however, have to be taken into consideration when transferring QFD to software development.

At first in the software industry the production process in a strict sense is a mere duplication process, just as the definite implementation process can hardly be influenced by special adjustable process parameters (Betts, 1990; Zultner, 1994). Therefore, the problem lies to a still higher degree than in manufacturing in the early phases of development. Applying QFD in software development therefore has to focus on the ability to prioritize the engineering activities and pay less attention to the deployment down to the software’s last line of code.

Second, the product software is identified not by its physical characteristics but by its behavior. Put differently: “Software [. . .] is valued not for what it is, but for what it does” (Zultner, 1994). This means that simply turning customer requirements into measurable quality elements, to be controlled in the development process, is difficult. Such a transfer would generally not be sufficient to appropriately take into account the various customer needs. Therefore, applying QFD to software products absolutely requires the function deployment part of Akao’s model (Akao, 1990), i.e. planning of functionality.
This is not considered in the classic four-phase model (ASI, 1990) that focuses on quality deployment.

Thus, besides the differences between requirements and solutions, in software QFD one has to distinguish between functional characteristics (product functions) and non-functional characteristics (quality elements) of a software product. Table I summarizes the most important definitions in software QFD including the common distinction between customer needs in the sense of original customer statements and the concrete customer requirements.

The effect is such that the quality elements used in the classic house of quality (HoQ) in manufacturing are replaced for application in software development by product functions. According to the general definition of the HoQ, in which particularly customer requirements are being analyzed, this new first matrix can be called software HoQ. The quality elements are first taken into consideration in the second matrix. Figure 1 displays an excerpt of a software HoQ.

SAP’s R/3 diary may serve as an example: integrated into the standard software system R/3, this piece of software enables several persons to view and maintain one another’s appointments from different locations at the same time. The numbers 0, 1, 3 and 9 in the cells represent the degree of correlation between the satisfaction of a customer requirement (lines) by a product function (columns). In the most simple of cases the importance of an individual product function is given by the sum of the multiplications of each requirement’s weightiness and the corresponding degree of correlation. In the case of a further development of a product, however, instead of the weightiness of the customer requirements, the importance of the requirement can be taken as the quotient of weightiness of the customer requirement over customer satisfaction. Moreover, the present data allow for numerous other quantitative and qualitative analyses. For example, a consistency analysis can be carried

<table>
<thead>
<tr>
<th>Table I. Important definitions in software QFD</th>
<th>Customer need = demand</th>
<th>Product characteristic = solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition</td>
<td>Need resulting from using the product; business need</td>
<td>Characteristics or capabilities of the product, independent of implementation, which in case of high fulfillment give the customers the advantages their requirements imply</td>
</tr>
<tr>
<td>Expression Definition</td>
<td>Customer requirement brief, concise statements, put in the customers’ words, about advantages which customers get or could get from using the product</td>
<td>Product function functional characteristic feature of the product, usually not measurable (creates perceptible output)</td>
</tr>
<tr>
<td>Example for a diary</td>
<td>Scheduling of appointments</td>
<td>Triggering actions from schedule</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Short response time</td>
</tr>
</tbody>
</table>
out: blank lines (i.e. a customer requirement without correlation) indicate that product functions are missing or have been overlooked, and blank columns (a product function without correlation) hint at the possibility that an unnecessary product function has been defined. These calculations and analysis are similar to the ones for the classic HoQ in manufacturing industries.

Zultner’s comprehensive approach

Despite the fact that software QFD must focus on the early phases of development, Zultner invented a framework of how to apply Akao’s comprehensive QFD to software development, including quality deployment according to the four-phase model (see Figure 2). The most important additional aspect is the emphasis on customer deployment before quality deployment, developed from the notion that software like most other products rarely is conceived to satisfy the needs of only one homogeneous customer. To identify customer groups and their importance for the engineering ahead, a table showing potentially relevant customer characteristics and a prioritization matrix of selected criteria are used (Zultner, 1991).

Quality deployment as a second step combines classic and software HoQ to form one single HoQ, so that functional as well as non-functional product
characteristics are being considered (including Akao’s function deployment). It is only through information deployment that at least the first two phases of the four-phase model have an equivalent in software development (see Figure 3). In information deployment the product functions that have been prioritized in the HoQ are turned into entities, processes or objects, depending on the development techniques used.

Zultner’s function deployment (not to be confused with Akao’s term used before) proceeds similarly when hardware design is required, and task deployment is concerned with activities of the development process itself.

Note: Includes customer and quality deployment (see Zultner, 1990)
Vertical deployments are intended to guarantee that general aspects such as reliability and costs are taken into account in any development activity. In this (possibly) measurable quality elements taken from the HoQ are particularly important.

In contrast with this comprehensive QFD framework, Zultner reduced his QFD approach to absolutely essential activities (Zultner, 1995). This model, called “Blitz QFD”, focuses entirely on the reception, analysis and weighting of customer requirements, after customers have been identified (customer deployment) and before the HoQ has been set up (quality deployment). Thereby, most advantages of applying QFD are expected to be achieved in less time with less effort.

**Shindo’s decomposition approach**
Software being an intangible product is the reason why customers find it very difficult to express requirements concerning quality (Shindo, 1999). Therefore, Shindo’s model starts out with a functional definition of the product by customers including possibly all data relevant for the software to be developed. Functions, data and their relationships are entered in a matrix, focusing design both on functions and on data. To get independent subsystems, yielding well-structured functions and data, the software is decomposed using the quantification method of type 3 (QM3) (Kihara, 1992), a technique to analyze the structure between two sets of non-quantifiable data correlated in a matrix, by calculation of pairs of values (see Figure 4).

Each subsystem’s functions, data and interfaces are then specified and, by means of the function point method (Dreger, 1989), functions whose realization

---

**Source:** Shindo (1999)

---

**Figure 4.** Shindo’s decomposition approach up to step 4 (the whole procedure follows nine steps)
would exceed the budget are sorted out. Only after decomposing the software into subsystems can specific quality requirements for each subsystem be gathered from the customers. By setting up quality tables of these quality requirements and functions, performance values or quality levels are determined for each function, and engineering bottle-necks are identified. Moreover, a database is created based on the QM3-table, algorithms are designed to reach the required degree of performance, i.e. quality, and by using failure mode effects analysis (FMEA) and function tree analysis (FTA) test cases are developed to test system integration effectively.

By putting much emphasis on the definition of relevant data and using QFD based on a modular structure of subsystems Shindo’s approach applies a kind of object orientation to QFD.

**Ohmori’s matrix-of-matrices approach**

Ohmori presents his framework of action for the new development of commercial individual software in a complex matrix-matrix-diagram (see Figure 5), yet this approach with a total of 14 matrices only covers quality deployment as it is described in the first two phases of the four-phase model (Ohmori, 1993). Essentially new in this approach are several activities for analyzing a comprehensive business system which combines all tasks necessary to reach the organization’s goals. Software as part of this higher task system has to support some of these business system tasks with its basic functions. Once these high-level functions are known, customer requirements (here called software quality requirements) are being identified, to be set against the product functions (software additional functions) in the software HoQ and (software) quality elements in the classic HoQ.

In another matrix, design points can be deduced from the importance of the quality elements for each product function. These design points indicate certain quality elements that have to be fulfilled to a higher degree when a particular

![Figure 5. Ohmori's software quality deployment](image)

**Source:** Ohmori (1993)
product function is being implemented. In the next step, the functional product characteristics are being related to individual software components such as software subsystems, data files or, later, program modules. The great amount of matrices (Figure 5) results from rigorously taking into account quality elements (not necessarily measurable), concerning the business system as well as the business software.

Ohmori’s approach consists of three phases. In addition to supporting requirements engineering and the early stages of the analysis phase a planning phase exists in order to embed the software into a higher business system. To be able to handle concrete customer requirements concerning software, developers need to have an idea of how the software will be used by the customers. Therefore, the initial focus is a business process-oriented one on the functions the software has to support.

**Herzwurm and Schockett’s PriFo software QFD model**

The important first purpose of QFD in software engineering and the main focus of product planning are on setting prioritized development goals based on the most important customer requirements. As mentioned earlier, in planning software products the preference setting and focusing aspects of QFD by means of the HoQ are more important than the deployment by a matrix sequence. That is the reason why Herzwurm and Schockett called their approach PriFo (prioritizing and focused) software QFD (Herzwurm et al., 2000). The entire PriFo QFD process is carried out by a QFD team put together from all departments (development, quality management, marketing, sales, service, etc.), to be extended in several team meetings by the selected typical customer representatives. Large-scale changes or add-ons to the results achieved in the meetings always have to be debated by the entire team, including the customer representatives. Among other things, this is one reason why the PriFo software QFD model is also called joint requirements engineering (Herzwurm et al., 2000).

The first task of a QFD project concerns setting the project’s goals, discussing the time schedule and cost planning and putting together a QFD team. Apart from these activities, the planning phase of a QFD implementation includes also defining the project’s content (product definition), identification of the customer groups and their importance for the development ahead as well as selecting customer representatives. This phase consists basically of normal meetings and brainstorming sessions of the persons in charge of the project. To define the product one or several QFD matrices from Ohmori’s planning phase can be used, and Zultrna’s customer deployment can serve as a guideline in identifying potential customer types and the weight of different customer groups.

Substituting for or supplementing a customer survey, one of the first QFD team meetings tries to ascertain customer needs and classifies them in the voice of the customer table to identify customer requirements. Then these requirements are structured using affinity- and tree-diagrams and weighted
(e.g. with the analytic hierarchy process and comparison by pairs (Saaty, 1995)) by as many members of the customer groups as possible under the overall control of the customer representatives. In case of a further development of a product the customer representatives will even evaluate the requirements according to the level of satisfaction the fulfillment of the requirements has reached up to that point (measured, for example, on a scale ranging from one indicating total dissatisfaction to five indicating perfect satisfaction). A (subjective) comparison with competitors at the requirements level is costly since one’s customers cannot be assumed to be able to evaluate the competition’s products, as well. Normally, therefore, additional customer representatives would have to be consulted. Combined with weighting and identifying satisfaction levels caused by the fulfillment of customer requirements, this could be done conducting a wide-ranging customer survey. The result of all these efforts is the table of customer requirements.

The table of customer requirements is the input of both quality tables to set up the software-HoQ representing function deployment and the classic HoQ representing the four-phase model. Identifying the product functions that will go into the software-HoQ as additional input is done similarly to the identification of customer requirements, using a voice of the engineer table. The difference is that in this meeting usually only the members of the QFD team are present, among them particularly the developers. Turning requirements into measurable quality elements as an additional input of the classic HoQ takes place at another internal QFD meeting. Identifying the relationships between product characteristics and customer requirements in both prioritization matrices is ideally done together with the customer representatives. A simplified product benchmarking at further internal team meetings, concerning the prioritized characteristics and including the setting of definite development goals, leads to a table of the most important product functions and a table of the most important quality elements.

The development goals derived from the product characteristics are related in a third matrix according to Ohmori and as a final step of consolidation compared among themselves. At the same time, they are examined for potential synergy effects or conflicts when realizing them and then narrowed down to design points as two-dimensional target values. Together with the most important customer requirements, product characteristics and design points these form the basis for setting up a requirements specification as a result of the requirement engineering process. In additional matrices, they can be turned into more specific development goals, at least at the analysis phase. Figure 6 shows the procedure of software product planning with PriPo QFD as described above.

**Experiences with software QFD**

QFD has proved its usefulness in software development practice. Experiences with the application of software QFD show that it is particularly beneficial in the interdisciplinary communication, clear understanding of customer and/or
user requirements, in the consent about the solutions found, a reduced number of post-delivery changes, complete documentation of all steps taken, a profitable product and satisfied customers. The disadvantages of QFD consist of its complexity and the considerable amount of time needed for preparing, carrying out and evaluating the meetings afterwards.

In an analysis of 25 software development projects in six companies (Digital Equipment, AT&T, Hewlett-Packard, Texas Instruments, IBM, CSK) by the University of Texas at Arlington requirements engineering with QFD is rated better regarding all 12 criteria used than requirements engineering with “traditional” methods. One of the reasons for success is indicated to be better communication among the development team as well as between customers and developers and better fulfillment of customer expectations (Haag, 1992).

An empirical investigation (Herzwurm et al., 1998) of 16 QFD projects – among them seven software projects – in which product developers were asked about their experiences with QFD confirms that QFD fulfills the special expectation which is connected with the employment of the instrument in product development (Figure 7). Concerning the customer-oriented objectives, technical and relative quality, the employment of QFD achieves very high satisfaction values. From the project-related goals point of view QFD particularly improves the co-operation of the persons involved and, due to focusing on the substantial, at the same time leads to a higher economy of the product development.

Source: Herzwurm et al. (2000)
In the SAP’s R/3 diary project the customers were asked before and after the project whether they were content with the diary and its development process (Herzwurm et al., 1996). With the help of the opinions given about the criteria stated above, it was possible to determine an index of the degree of satisfaction with every single criterion and for the diary on the whole (= index of customer satisfaction as mean of the evaluated assessment factors). The results show a fairly high degree of satisfaction with the diary. Comparing the “diary with QFD” and the “diary without QFD”, however, shows that even this rather high level of satisfaction can still be increased by QFD. This is true for each and every criterion as well as for the diary on the whole (without QFD 66.7; with QFD 82.0 (see Figure 8)).

QFD for electronic business: continuous QFD
E-business as understood in this paper is the buying and selling of goods and services on the Internet, especially the World Wide Web (for other and/or detailed definitions see Kalakota and Whinston (1997)). Developing e-business applications leads to new challenges for requirements engineering methods (Yoffie and Cusumano, 1999). The domain is relatively new. The technology is rapidly changing. Time and creativity are important success factors (Arthur, 1996). The tasks are less clear because of the unstable environment and the newness of requirements and technical solutions for both developers and customers. This section presents a special variant of QFD for building e-business applications that considers these conditions. We call this method continuous QFD (CQFD). There are three basic elements of CQFD in order to overcome with fuzzy development tasks:

(1) Incremental planning and implementation cycles make product characteristics feasible and help to make the requirements more clear for the customer.
(2) Employing information technology (IT) is very important for handling the dynamic, particularly the high number of changes regarding requirements and technical solutions.

(3) The use of templates, containing prepared standard requirements, solutions, etc. and representing a tailored process for e-business planning, accelerates the development process and ensures quality.

Incremental planning and implementation cycles
Maintaining a high level of customer satisfaction, when customer requirements change after product delivery or improved technologies become available, implies continuous adaptation of the product. Therefore, planning with CQFD does not end when a specification document is written or a first version of the product is delivered. Instead, planning carries on as long as the product is in the market. Adaptation to changing requirements and/or product characteristics at any time means that all elements in the prioritization matrices may change during the planning process. These changes are an explicit component of the method and represent no exceptions.

To integrate these changes continuously CQFD dismisses the “do it right the first time” approach. Instead, CQFD proceeds in many iterations. This has two implications: on the one hand, decisions taken in an earlier meeting may be subjected to discussion again at a later point, revised and changed. On the other hand, the prioritization matrices are being developed incrementally with each iteration or meeting.

There are a great number of short meetings following the same procedure, consisting of five steps:
The basic difference between meetings is the degree of detail of discussions and (provisional) results. New matrix inputs may occur in any meeting, and evaluation of requirement weights and correlation values becomes more refined with each meeting.

Customer requirements and product characteristics are being collected simultaneously, which is much more appropriate in an unstable environment than the sequential procedure traditional QFD follows. Independently of the degree of detail that the planning has reached at a certain point the focus lies at all times on the most important customer requirements and the product characteristics that are strongly correlated. This is essential when planning is deadline-based and feasibility of product characteristics is uncertain.

In order to carry out the principle of continuous adaptation to changed customer requirements and development, conditions into practice, planning and development take place simultaneously. Thus, at any time a product is available which provides at least the most important functions. Moreover, early customer feedback can be gathered in order to influence and possibly direct the further development.

**Employment of information technologies**

CQFD makes extensive use of IT. The Internet makes asynchronous, distributed, economical and structured generation and processing of information possible, independent of the actual meeting’s time and its location. To prepare the meetings, the moderator will structure and visualize the gathered information so that discussion and evaluation by the entire team can take an efficient course. Special QFD software tools are being employed before, during and after the meetings for documentation, for weighting and entering correlation values by the team members, and for calculation. The benefits are that the process is continually being documented, and that via the Internet this documentation can be read and worked on by all CQFD team members at any time (Herzwurm and Schockert, 1999). With EASY-QFD, a non-commercial free Web-based software tool supporting QFD (www.qfid.de/werkzeuge/easyqfd/easyqfd.htm), QFD teams can collaborate independently of location and time (see Figure 9).

**Use of templates**

In CQFD planning is based on a template corresponding to the individual development situation, here e-commerce applications. Templates include standard content items like customer requirements, product characteristics,
Web components and technology packages. The templates accelerate the process and make the fuzzy planning information more concrete.

Development of an e-business application requires three planning steps (Figure 10), each represented by a prioritization matrix:

1. concept planning;
2. component planning; and
3. technology planning.

Of the three planning steps only concept planning needs to be carried out by developers and customers, the remaining two parts being the developers’ responsibility.

The lists the template provides (represented by light gray boxes in Figure 10) contain standard values for customer requirements, product characteristics, Web components and technology packages. For each CQFD project these lists have to be adapted by entering new items or by changing or completing existing items. For example, in consumer-oriented e-applications usability characteristics such as color, video, sound, placement, feel, etc. are critical quality elements. The list entries are then completed by related data such as weights, satisfaction values, satisfaction comparison with competing products, etc.

The lists of customer requirements and product characteristics then form the input rows and columns, respectively, for the HoQ called concept planning (Figure 11). The correlations filling this house are to be determined by the team.
as a whole, a task that is done in various iterations both via the Internet and within the meetings.

Component planning, then, sets particularly prioritized product functions such as row input against Web components which form the columns. By Web component we understand reusable parts of a Web application (for example, guest book, search engine, links, pop-up help, pull-down menu).

In the third prioritization matrix, called technology planning, especially quality elements are correlated to technology packages in order to determine the set of technologies which best enables the developers to realize the sort and level of quality required by the customers (determined in concept planning). These packages each include a combination of server operating system (e.g. Microsoft NT Server, UNIX servers such as Solaris or Linux), a database application (such as MySQL, Microsoft SQL Server, etc.), a kind of server software (e.g. Apache Server, Internet Information Server, etc.), and a script/programming language (such as PHP, Perl, C++, Java-Script, ASP). These sets do not consist of completely exclusive technologies in each category, but they are all different to some extent, and they present sensible alternatives.

Table II summarises the main differences between CQFD and traditional QFD.

Experiences with continuous QFD
The CQFD approach was tested in an e-business application case study. The project’s target from the initiator’s point of view was to plan and implement a
Web site for a computer dealer whose primary customer group is administrative bodies such as city councils or universities and/or individual departments of these. After the project CQFD team members were asked whether their expectations on the project and the method used for planning had been fulfilled. All participants agreed that the results, including the respective demanded degree of fulfillment, proved planning with CQFD successful in this case study (see Figure 12 for details).

Regarding two issues the expectations of the CQFD team members had not been fulfilled: the Internet-based communication and testing platform caused technical troubles for some customer representatives so that an additional plug-in was required to overcome Java compatibility problems. The time effort between the CQFD meetings was underestimated by nearly all team members.

**Conclusion**

Software QFD bridges a gap in the software development process which has been left behind by development methods commonly used today: it bridges the gap to the customer. Instead of hoping for concrete targets formulated by customers, development can start directly with customer needs and transfer
Table II.
Main differences between traditional QFD and CQFD

<table>
<thead>
<tr>
<th></th>
<th>Traditional QFD</th>
<th>Continuous QFD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process</td>
<td>Sequential</td>
<td>Repetitive</td>
</tr>
<tr>
<td>Implementation</td>
<td>After finishing planning activities (planning determines implementation)</td>
<td>Parallel planning and implementation (implementation also supports planning)</td>
</tr>
<tr>
<td>Results</td>
<td>Completed milestones</td>
<td>Incremental provisional results</td>
</tr>
<tr>
<td>Planning</td>
<td>Activity-oriented</td>
<td>Time-oriented</td>
</tr>
<tr>
<td>Time horizon</td>
<td>Defined end</td>
<td>Continuous</td>
</tr>
<tr>
<td>Changes</td>
<td>Exceptions (to be avoided)</td>
<td>Standard (adaptation intended)</td>
</tr>
<tr>
<td>Extent</td>
<td>Completeness desirable</td>
<td>Focus on essential</td>
</tr>
<tr>
<td>Meetings</td>
<td>Few, long meetings</td>
<td>Many, short meetings</td>
</tr>
<tr>
<td>Templates</td>
<td>For matrix chains only</td>
<td>For matrix chains and content of matrices</td>
</tr>
<tr>
<td>IT-support</td>
<td>Documentation</td>
<td>Documentation and communication (virtual teams)</td>
</tr>
</tbody>
</table>

them to concrete product requirements. This is basically done using the simple means of a systematic procedure for teamwork and the ability to prioritize all information concerning product development in a justified way. In practice, this means that the activities carried out to date, frequently little structured and hard to define, to elicit product requirements are made explicit in a fixed methodical framework.

Several enterprises have recognized the great improvement potential in this area for their use and developed methods similar to QFD, even without (probably) having ever heard of QFD. As an example, in the course of the years Microsoft (Cusumano and Selby, 1995) established a systematic product-planning process called “activity-based planning”. It emerged from people internally involved realizing that the chaotic and not objective evaluation

![Graph showing fulfillment of team members' expectations](image-url)
system in use before (“feature war”) had become outdated due to increasing software complexity, time pressure and a great number of customer complaints. In Microsoft’s activity-based planning employees with different business functions (product manager, developer, tester, etc.) work together in teams so that customer opinion is understood uniformly for the impending release development. Starting-points are customer activities in the sense of business processes. Based on these, a consistent set of product features is derived for fulfillment of these activities. Quantitative evaluation of the activities is done by means of user studies, separated by market segments and/or customer profiles. By distributing weighting points on those features which present (partial) solutions for one or more of the activities, a ranking of the features is established. Parallels with the way a software HoQ is set up, especially to the determination of the correlation values representing the satisfaction of the customer requirements by the product functions, are obvious. More and more firms will follow such positive examples.

One of the greatest advantages of the method is the flexibility inherent in QFD because it allows integration of special, company- or project-specific peculiarities into the procedure. This is also shown by continuous QFD for employment in the special case of fuzzy development tasks. Continuous QFD also indicates the direction for future further development of QFD. So far, only one template exists for development of e-business applications including standard items of Web components and corresponding technology packages. Particularly for development under immense time pressure, growing continually more complex, yet still shorter-lived in the dynamic age of the Internet, further, more specific templates will be added. QFD will be embedded in incremental and iterative development to a growing extent. The future belongs to shorter meetings that continually give more precise results and are run in parallel with all the development. In these meetings, prioritized information will act even more as a guide and controller of development activities in the sense of project management. For frequent feedback from design and implementation to activities of requirements engineering, employment of prototypes and early product versions will be mandatory for stimulation. The tendency towards globalization in the world economy will add further to the pressure for distributed and virtual development. The Internet will be used to connect QFD team members in different locations. Asynchronous communication and collection of information by e-mail or some similar tool will not be the only method used. Direct contact using video-conferencing systems or some similar instrument will be required, especially to support the innovative and idea-creating element of QFD.

Yoji Akao, the founder of QFD, takes the same direction and for the future demands information systems:

... where demanded items such as customer needs and claims are translated into engineering information and then transmitted to each department on a real time basis, so that the production department can begin its work almost simultaneously with design (Akao et al., 1999).
In the coming years many new kinds and cases of employment of (software) QFD will emerge:

In the twenty-first century information era, it is expected that QFD will be positioned as the central method for building a system (Akao et al., 1999).

References


Kihara, T. (1992), “Decomposing software requirements by using QFD and QMIII”, a thesis for the Degree of Master of Science, Thayer School of Engineering, Dartmouth College, Hanover, NH.


