

Quality Function Deployment (QFD): An Effective Technique For Requirements Acquisition and Reuse

Tuyet-Lan Tran and Joseph S. Sherif
Jet Propulsion Laboratory, Software Assurance
California Institute of Technology, Pasadena, CA 91109 USA

Abstract

A general and accepted understanding of how to capture requirements, allocator flow-down top-level requirements, verify and validate lower-level requirements, is at best sought in theory but not rigorously sought in practice. More often than not, the customers (or users) are blamed for not properly articulating their requirements or even understanding their own needs. However, the problem is deeper than that, and it involves not only the customers but also the system analysts or engineers and designers as well.

This paper discusses problems in requirements acquisition and reuse, describes some tools to alleviate these problems, and puts forward Quality Function Deployment (QFD) as an effective tool for the capture and refinement of requirements. QFD when applied to a project will: (1) improve quality by emphasizing defect prevention rather than defect inspection, (2) improve communications between customers, system engineers, programmers and testers and thus contribute to project success, (3) enable alignment between customer requirements, product (or design) requirements, and cost requirements (or constraints), by explicitly correlating key product requirements to customer needs and expectations, (4) reduce costs of projects due to reduced costs of late-in-the life-cycle rework, of unnecessary post-facto documentation and inspection, (5) improve the management of change through rigorous prioritization and explicit tradeoff analysis, and (6) enable refinement and reuse of requirements across projects through cataloging of key performance requirements.

Introduction

Requirements engineering is one of the most crucial parts of the development process of any project, yet it is the least supported or least understood part due to the following reasons: (1) requirements are particularly difficult to specify and analyze since they are derived from the needs of many different customers or people; (2) difficulty to achieve a complete understanding of the application domain within which the proposed system will function, as discussed by Rubenstein and Waters [9]; and (3) all relevant aspects of a proposed system may be difficult to capture by a single paradigm. This is due to the fact that each paradigm is embodied in a single requirement language that may have its own limitations to express some important requirements. Hsia, et. al., [3] also assert that the quality of a product is only as good as the process that creates it; and that requirements engineering is one of the most crucial steps in this creation process. Hsia describes requirements engineering as the disciplined application of proven principles, methods, tools, and notations to map a proposed system's intended behavior and its associated constraints. This mapping includes: (1) identification and documentation of user needs, (2) development of a requirements document that describes how to satisfy user needs, (3) analysis and validation of the requirements document and (4) means to support the evolution of users needs. The primary output of requirements engineering is a requirements specification that shall be consistent; consistent with other existing documents; correct and complete with respect to satisfying users needs; understandable to users, designers and

(esters; and capable of serving as a basis for both design and test [4],

Requirements Acquisition and Reuse

The principal problems in Requirements Acquisition and Reuse include difficulties in (1) agreement about requirements statements, (2) communication, (3) managing change, i.e., maintenance and evolution of initial requirements and identifying inconsistencies between initial and new requirements, and (4) formalism and abstraction in capturing objective reality, since constructed reality is, after all, a result of interactions among participants in the requirements process.

Curtis, et. al [2] identified two significant problems in requirements that may cause major difficulties during the development of projects: acquisition of accurate problem domain knowledge, and volatility of requirements. Any of these problems will contribute to low quality projects, budget overrun and schedule slip.

Lubars et. al., [7] assert that the traditional way of requirements capture by prose-like unstructured, obscure and somewhat ambiguous statements is no longer effective; and they recommend that new techniques and tools for requirements engineering should be advocated. At the other end of the spectrum, requirements can be orthogonal to one another. This high level of rigor is sometimes worthwhile pursuing, sometimes not.

Johnson et. al., [5] develop a tool ARIES (Acquisition of Requirements And Incremental Evolution of Specifications) to investigate the support of requirements analysis in evaluating system requirements and codifying them in formal specifications. The key feature of this approach is to have a single highly expressive underlying representation interfaced simultaneously to multiple presentations, each with notations that have different degrees of

expression. In so doing, the analyst can use multiple languages for describing systems and where these descriptions yield a single consistent model of the system. The authors indicate two problems that they encountered in their approach. First their approach focuses in recording the underlying semantics of presentations, instead of details of their syntactic form. For example, if a flow diagram is constructed, ARIES does not record the exact position of each node in the diagram, so if the diagram is recreated later, it may not have quite the same shape. Second, when a diagram cannot be mapped onto the underlying representation in a unique way. This happens when multiple expressions are all expressed in the same way in a given notation. The authors give workaround ideas to rectify these problems.

Lefering [6] discusses a new framework that allows the efficient realization of new integration tools to support software development in different phases of the software development life cycle. An integration tool between requirements engineering and programming in the large has been built as a first prototype of integrations that are instantiation of that framework. The tool integrates requirements specifications and architectures by executing a transformation between the documents and then controls the traceability by examining links between transformed increments.

Mays et. al, [8] describe the Planning and Design Methodology (PDM): a requirements planning process that supports the collection, analysis, documentation, and tracking of software requirements. The process includes requirements collection, definition of the underlying problems, development of an external functional description that addresses the problems, and development of system and product designs from the external functional descriptions. PDM is designed to handle both system-level and product-level

requirements with the following major objectives: (1) the ability to categorize, prioritize, coordinate, and manage a large number of diverse requirements from the time they are received until they have been successfully integrated into a product design, (2) support for the systematic analysis of requirements from the customer viewpoint by first identifying the underlying end-user problem, (3) support for the verification of the problem definition and its proposed solution through customer reviews, (4) support for the assessment of a requirement's relative priority, value, and business justification or business case, (5) support for traceability of requirements, from initial receipt through implementation (6) support for verification of the transformation of requirements at each step of the process, particularly at the critical transition from external specification to internal design, and (7) usability and human factors emphasis during requirements definition. Mays et al., assert that the Planning and Design Methodology (PDM) has proved to be valuable in systematically analyzing, documenting, and managing a large set of requirements from initial input through design. However, much work remains to be done in the area of planning and requirements engineering so as to attain additional gains in quality and productivity.

Successful Projects

Tran et al., [10] describe successful projects as those that meet valid functional requirements as well as users expectations; adhere to the spirit of process standards that promote rigor, discipline and continuous improvement; and are accomplished on time and within budget. They also assert that successful projects always exhibit an overall plan with shared purpose and goals and management's commitment to these goals. Also, and of equal importance, are the presence and support of dedicated and skilled process people. They summarize the criteria for in-depth assessment of projects and give

attributes of projects that are considered successful. These attributes include (1) consistent visibility of requirements, (2) well-enforced configuration control, (3) involvement of sponsors, users, and customers throughout the development life cycle, (4) support of a dedicated and skilled development team, (5) effective team communications, and (6) compliance to sound process policies or guidelines (for a disciplined and cost-effective development process). The authors also give the following recommendations for good requirements engineering and management:

- (1) There must be an automated facility or tool to support and enforce both the requirements acquisition and configuration control process, thereby making it more cost effective.
- (2) The customers (including users), system engineers, programmers, and test engineers should all work as a team in validating the requirements up-front. Of equal importance is the thorough assignment (and reassignment, as appropriate) of requirements priorities. And, the system engineer (instead of a manager) must be the person in charge of making product-requirements decisions and of closing the requirements loop with management and all team members, continuously.
- (3) The requirements management process must be owned by a single individual, and preferably by the system engineer (or his or her delegate, such as a subsystem engineer or a process assurance analyst). This owner must also be responsible for requirements baselining and requirements-database control, to ensure a minimum number of hand-overs.
- (4) Since the user representatives are key players in acceptance testing, they must be responsible for

reviewing and concurring with the acceptance criteria specified for each and every high-level requirement (01 "customer acceptance criteria"). This buy-in concept is a success factor critical to the aligning of planned product capabilities with customer needs.

Quality Function Deployment (QFD)

QFD is part of Total Quality Management (TQM) which is a set of methods and tools that all employees in all departments can use; to maintain or improve quality, cost, procedures and systems; or to give customers or users a product which is of highest quality, within budget and schedule. In support of this goal, the QFD technique provides a structured process for capturing and bringing the voice of the customer into the production organization. It enables the tailoring of products to customer needs, through customer interaction and brainstorming with product development, testing, maintenance, quality control [cam, and system designers. QFD assigns priorities or weights to product improvements, and reduces development time by focusing on essential design (or change) parameters, and by concentrating on defect prevention rather than defect detection -- especially at the customer requirements level.

Yoji Akao introduced QFD to the United States in October 1983 in a short article in the Journal of Quality Progress. QFD is a technique for improving the transition of a project from requirements to design and from design to implementation to delivery. It helps to introduce the idea of quality in the early phases of the requirements cycle from the customer's perspective (as opposed to engineering perspective) and to reevaluate quality considerations throughout the project's entire life cycle from that only perspective. In most implementations, QFD uses charts or matrices to discover interrelationships between customers and/or users needs,

product performance characteristics and design and implementation methods.

The goal of QFD is to deploy the "voice of the customer" throughout the product's entire technical specifications and resource requirements. Detailed matrices listing the customer's rated "whats" (or expectations) are correlated with the "hows", to show how each customer requirement will be met, and which team(s) will be responsible for each performance component [1], This systematic technique of listening to the voice of the customer, and insuring the traceability of design to the customer's requirements are the most crucial aspects of QFD in delivering high quality products.

The customer's requirements planning matrix is the most important tenet for the QFD concept. It is often referred to as the House of Quality as shown in Figure 1 and consists of six basic steps: (1) identify customer's attributes or requirements, (2) identify technical features (counterpart characteristics) of the requirements, (3) relate the customer's requirements to the technical features, (4) conduct an evaluation of competing products, (5) evaluate technical features and specify a target value for each feature, and (6) determine which technical features to deploy in the remainder of the production process.

Customer's attributes are the product's requirements in customer's terms and language. The technical features are the design attributes expressed in the language of the system engineer, designer, and developer. These features must be measurable, since the output will be controlled and compared to objective targets. Relating the customer's attributes or requirements to the technical features (also referred to as product characteristics or performance requirements throughout this paper) will show the strength of the relationship between them; and show whether the attributes are addressed fully and properly or whether the final product

will have difficulty in meeting customer needs. Evaluation of competing products will enable designers to seek opportunities for improvement and technology deployment, and allows importance ratings to be set on specific, key design parameters. Evaluating technical features and developing targets will introduce quantitative measures for product consistency, and customer-perceived quality. Determining which features to deploy will be based on identifying those characteristics that have a strong relationship to customer needs. Only these characteristics will need to be deployed in the design and production process, to ensure that the voice of the customer is heard and that we are producing the right product right (as opposed to producing the wrong product right).

QFD and Software Development

Since 1980 few companies in Japan, Europe and the United States had been considering using Total Quality Management (TQM), concurrent engineering and QFD techniques for software development or at least for the first phases of software development, i.e., software requirements and specifications. For example IBM Japan has developed a system called SQUALAS for applying QFD in support of quality assurance for software. This system gives a thorough definition of at least 39 specific quality characteristics of software such as performance, reliability, usability, flexibility, cost of change, etc., and also clarifies and specifies the divisions of the software development process; and prepares the QFD charts on matrices of the quality control process, the quality assurance item list, and the quality-cost tradeoff matrices. NEC company has developed a QFD technique for quantifying software measurement methods to test software quality. NEC also developed a new QFD time sequence quality chart that addresses the following problems: (1) broadening customer's requirements during the design process, (2) situations

where technology could not be selected until the design was complete, and (3) situations in which additions or changes to software requirements occurred. In essence, this QFD time sequence quality chart can be used flexibly to deploy design specifications in various degrees of completion as explained by Yoshizawa, et. al., [12]. Digital Equipment Corporation (DEC) adapted QFD techniques to a software project that deals with customers direct access to an automated purchasing system together with its accompanying telecommunications and terminal interfaces. Figure 2 shows (DEC) adaptation of QFD four houses of quality to QFD software development terminology [11].

CSK of Japan has also been using QFD for software since 1985. Figure 3 shows the steps in their QFD activities for developing the company's software [12]. These steps include: (1) collecting customer requirements (from original interview data and brainstorming sessions by a cross-functional team); (2) generating the quality requirements (by identifying the several levels of product characteristics that correlate with the customer requirements (or "the demanded quality")); (3) generating the function-based requirements (by exploding the system functions into several levels of functional requirements); (4) establishing the planned quality. This fourth step consists in: (a) extracting and analyzing selected parameters from the quality requirements, (b) deciding which parameters are most strongly correlated with the demanded quality and become the product's "quality characteristics", (c) establishing a standard value for each quality characteristic (also referred to as technical feature, or product characteristic, or performance requirement, throughout this paper), (d) deploying these quality characteristics into processes, and (e) implementing these processes in software development. CSK's next major activities of the QFD technique are as follows: (5) analyzing

the relationships between the impact of the implemented software on customer demands (or customer requirements) and the quality characteristics; (6) capturing the results of this evaluation (by rating customer satisfaction for each customer demand); (7) analyzing the relationships between the deployed software processes and the selected quality characteristics; (8) refining the planned quality chart, for the next development effort. At present CSK is developing a QFD support system using artificial intelligence for improving the company's software development activities, efforts, quality and productivity.

Discussion of QFD Benefits

QFD has become an effective and important aspect of investment for many companies because it is the cornerstone for implementing concurrent engineering and Total Quality Management (TQM). One of QFD's major benefits is that it enhances the efficient use of tools such as Experimental Design and Statistical Process Control for optimization of design performance. QFD charts achieve the competitive edge for the customer and user by identifying desired product goals and their interaction with process capabilities and thus allow explicit focus on selected technical improvements. An important contribution of QFD to people management, less evident in terms of cost savings, is that it acts as a powerful catalyst for team building and for infusing technical excitement into the consensus building process. Most critically, and with respect to customer fulfillment, the benefits of QFD are that: (1) customer needs are better captured, thoroughly rated, and assessed against competing products; (2) customer satisfaction for the entire product line can be greatly improved via reuse and refining of performance requirements and of the relationships between the product's performance characteristics and its customer needs. With respect to development-process cost effectiveness,

the advantages of QFD are as follows: (1) communication of requirements is more consistent in (degree-of-abstraction and more effective through quantified relationships between the characteristics of the product and customer needs; (2) planning becomes more specific, control points are clarified, and duplication of effort is eliminated; (3) tradeoff analysis is more explicit and concentrated on specific potentially-conflicting design features or bottlenecks, consensus-building becomes easier, and an informed balance between quality and cost is made; (4) errors in requirements capture, requirements analysis and design are fewer, and there are fewer design changes late in development or production -- which in turn reduces overall product cycle time and project development costs.

It has been reported that, although only 6 percent of project cost and 10 percent of project duration are spent in the requirements phase, it costs about 10 times more to repair a defect during implementation than during the requirements phase, and it costs between 100 and 200 times more during maintenance. Formal system performance records also show that 30-to-50% of the cost of building a hardware-software system are spent in finding and correcting defects. For certain application areas, about 60-90% of software failures observed are said not to be caused by code errors, but are attributed to requirements errors. These requirements-related problems, when coupled with the Total Quality Management (TQM) goals of increased product quality and lowered cost, suggest that the area for highest-return on quality investment is in the prevention of defects with greatest-impact (or greatest amplification-rate) potential; i.e., at the requirements level and at the front-end of the development life-cycle. This front-end could also mean preproject phase, prototyping phase, or exploratory or conceptual development phase.

QFD targets both the front-end of the development process and the product life-cycle itself, for improvement (either small or dramatic). By simultaneously capturing Customer requirements, product requirements and the results of rigorous analysis from a knowledgeable team, QFD becomes the repository for product plans and specifications. This repository in turn provides the single source for subsequent retrieval and reuse of requirements, whether for in-project refinements or for across-project (or across-release) revisions.

Conclusion

This paper discussed problems in requirements acquisition and reuse; described some tools to alleviate these problems, and put forward QFD as an effective tool for the capture and refinement of requirements.

QFD is an effective and promising technique in alleviating the problems associated with the early phases of requirements and specifications. From the TQM perspective, QFD is an excellent avenue for specification of the "right product" at the right price. QFD, indeed, is a cross-functional tool that enables organizations to focus on key customer's and user's demands and develop innovative responses to those needs. Most critically, QFD systematizes rigor in requirements activities, while maintaining documentation and requirements-tracing costs to a strict minimum. Lastly, the authors would like to conclude that, although QFD has been accepted as a useful tool for product planning, as well as for parsimonious requirements specification, its most unique potential will be as catalyst for rapid technology deployment and for parameter design of information systems.

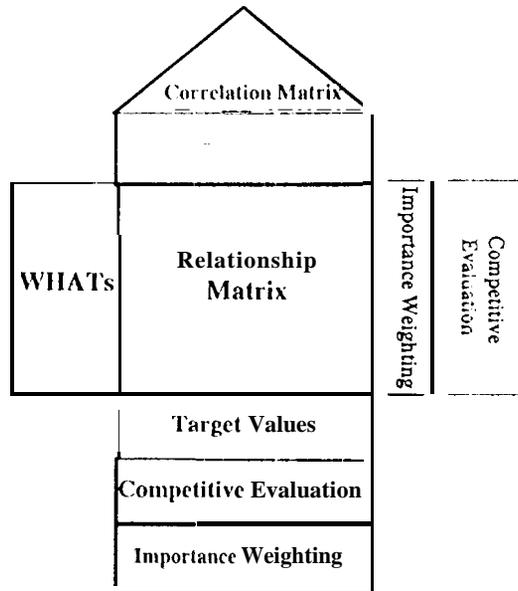
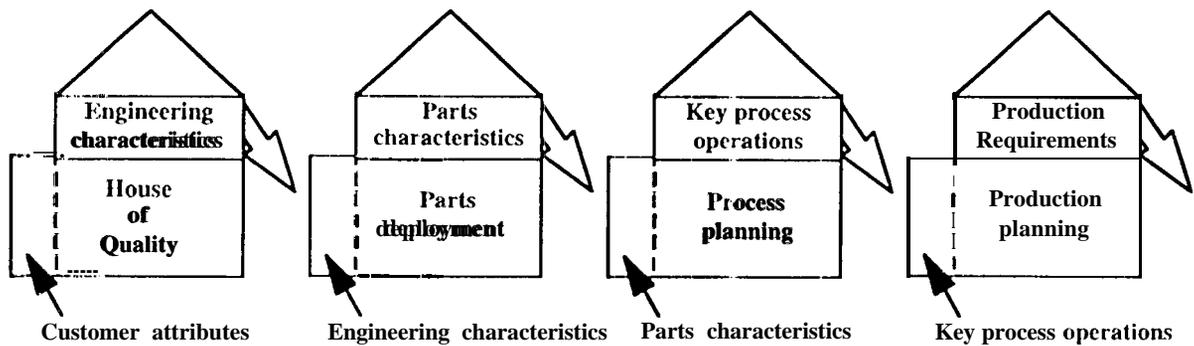


Figure 1. The House of Quality



The Four Houses of Quality

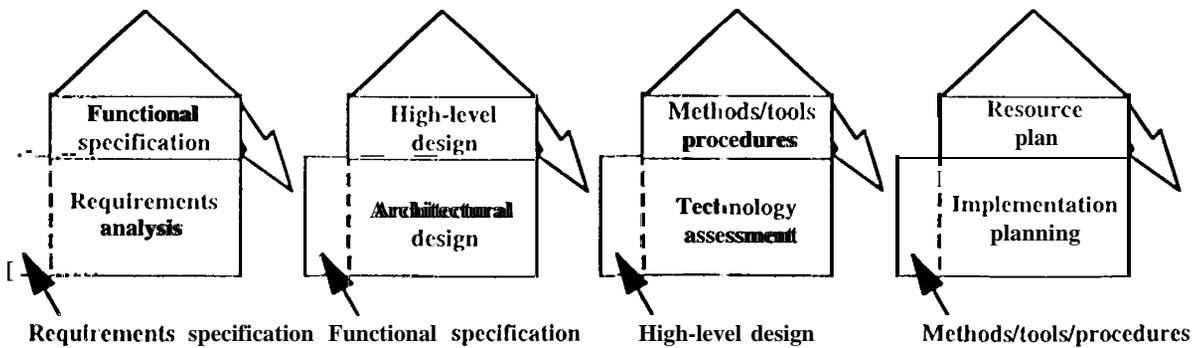


Figure 2. The Four Houses of Quality With Software Terminology

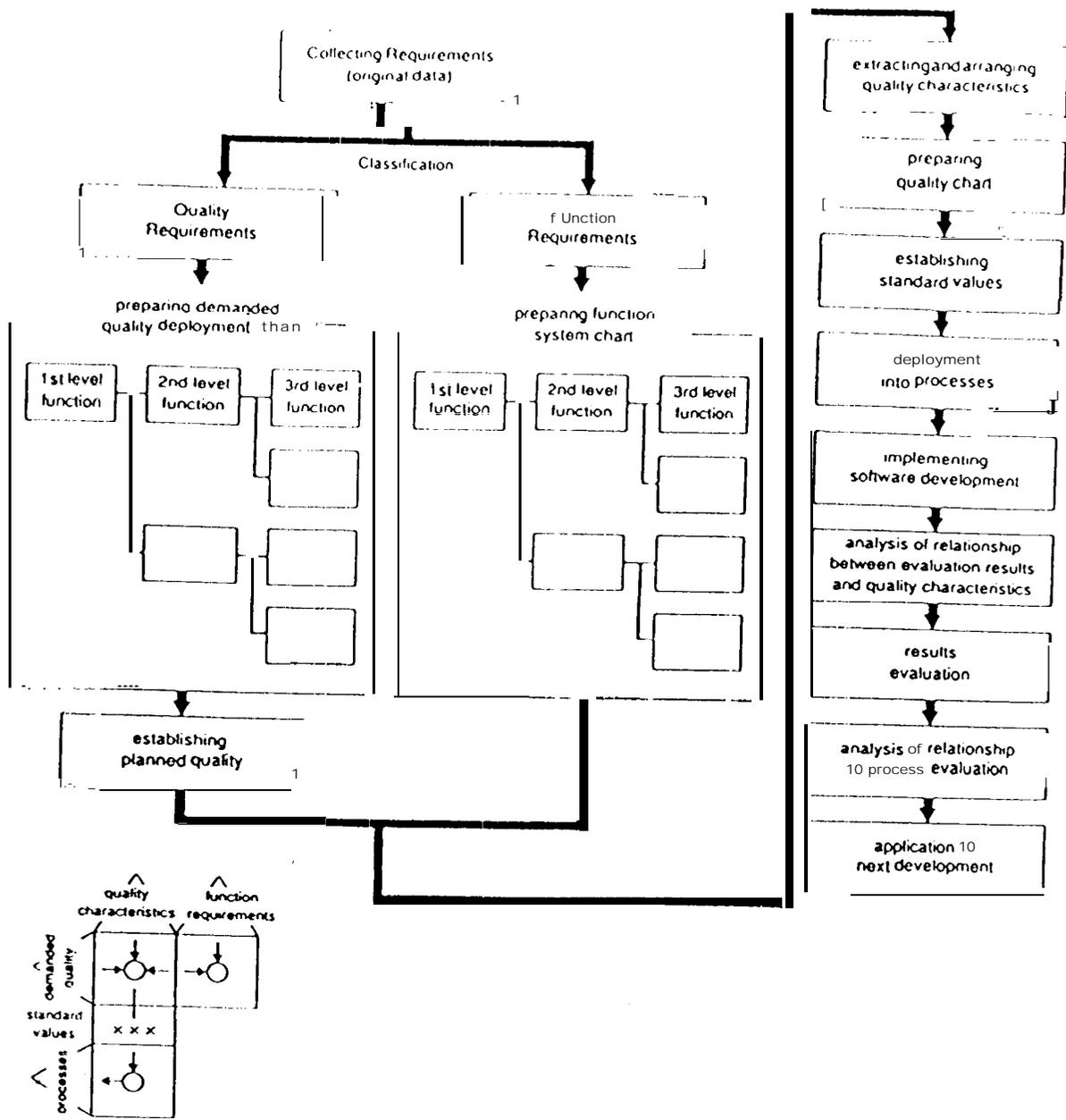


Figure 3. Steps in QFD Activities

REFERENCES

- [1] Clemmer, J. and B. Sheeby, Firing on All Cylinders, Irwin, 11011cWOOD, Ill, 1992
- [2] Curtis, B., Krasner, H., and N. Iscoe, "A Field Study of The Software Design Process for Large Systems," Comm. ACM, 31,11, pp. 1268-1287, 1988.
- [3] Hsia, P., and A. T. Young, "Another Approach to System Decomposition: Requirements Clustering," Pmt. The Twelfth Annual Int'l Computer Software and Applications Conference (COMPSAC), Chicago, ILL, pp. 75-82, 1988.
- [4] Hsia, J., Jayara, Jan, S., Gao, J., King, D., Toyoshima, Y., and C. Chen, "Formal Approaches to Scenario Analysis," IEEE Software 11, 2, pp. 33-41, 1994.
- [5] Johnson, W. L., Feather, M., and D. Harris, "Representing and Presenting Requirements Knowledge," IEEE Trans. on S/W. Eng. 18, 10, pp. 853-869, 1992.
- [6] Lefering, J., "An Incremental Tool Between Requirements Engineering and Programming in the Large," Proc. IEEE Int. Symp. on Req. Eng., San Diego, Calif., pp. 82-89, January 4 - 6, 1993.
- [7] Lubars, M., Potts, C., and C. Richter, "A Review of The State of The Practice in Requirements Modeling," Proc. IEEE Int. Symp. on Req. Eng., pp. 2-14, San Diego, CA, January 4 - 6, 1993.
- [8] Mays, R. G., Orzech, L. S., Ciarfella, W. A., and R. W. Phillips, "PDM: A Requirements Methodology For Software System Enhancements," IBM Systems Journal, 24.2, pp. 134-149, 1985,
- [9] Reubenstein, H. B., and R. C. Waters, "The Requirements Apprentice: Automated Assistance For Requirements Engineering," IEEE Trans on SE, 17, 3, pp. 226-240, 1991.
- [10] Tran, T. L., Lee, S., and J. S. Sherif, "The Network Operations Control Center (NOCC) Upgrade Task: Lessons Learned," TDA Progress Report Number 42-118, NASA, JPL, pp. 160-168, 1994.
- [11] Treeck, Van, G., and R. Thackeray, "QFD at Digital Equipment Corporation," Concurrent Engineering, 1, 1, pp. 14-22, Jan./Feb. 1991,
- [12] Yoshizawa, T., Togari, H., and T. Koribayashi, "QFD, Integrating Customer Requirements into Product Design, (Akao, Y. editor), Productivity Press, Cambridge, MA, 1990.