Application of QFD to the software development process

William D. Barnett and M.K. Raja
Department of Information Systems and Management Science, University of Texas at Arlington, Arlington, Texas, USA

Introduction
Quality of the software product and process affects the competitive position of the business organization and the role of the IS function within the organization[1]. Information systems (IS) departments are under increased pressure to deliver operational systems on-time[2]. Poor quality systems consume additional resources and contribute to the two- to three-year backlog of development projects that exists in most companies[1,3]. The lack of information technology (IT) support in turn degrades the competitive position of enterprise[1,4].

Software engineering and information engineering (IE) have long advocated the application of engineering-like discipline to the software development activity[1,5,6]. Integrated computer aided software engineering (I-CASE) represents an attempt to use automation to increase developer productivity. These advances provide improved methods for carrying out the software development process; however, there is no accompanying improvement in the understanding of this process. The adaptation of the philosophy of total quality management (TQM) from the manufacturing quality literature has been proposed by some as a possible guide for introducing quality into the software development activity[7-10].

This article will examine the methods that have been proposed in the literature for moving the quality focus from the software product to the development process. The difficulties inherent in current software quality methods and the relevance of TQM tools for addressing these problems will be examined. Finally, a requirements specification model based on quality function deployment (QFD) will be proposed.

Process improvement in software engineering
Software quality has traditionally been defined in terms of fitness for use[3,5,11]. A software product is deemed fit for use if it performs to some level of user satisfaction, in terms of functionality and continuous operation[1,11,12].

Quality is built into the software product through a series of progressively more integrated testing:
- unit testing;
- string testing;
• integration testing;
• system testing;
• acceptance testing.

Folkes[1] raises the point that user satisfaction is highly dependent on the individual using the software application. Testing has also been criticized as impractical. The increasing complexity of modern software severely limits the ability of developers to inspect and test every possible input stream and logic path[12,13].

Cho[14] has described a statistical method for making inferences about the quality of a software product based on concepts of acceptance sampling. Software outputs are equated with the outputs of a manufacturing process. Testing is performed on a random sample of program outputs, given that the failure rate is at or below an acceptable quality level (AQL), the product is deemed to be acceptable. The use of acceptance sampling to make inferences about product quality has been criticized as theoretically unsound[15].

Dunn[11] also highlights the reduction of management risk as an element of software quality. Late delivery, cost overruns, inadequate product performance, and a short product life are management risks that must be controlled along with fitness for use to achieve quality software. These management risks relate to the process of developing a software application and not the software product itself.

The most serious criticism of current development methods is that these methods are product focused[8,11-13,16,17]. There are currently no formal methods for software requirements specification and quality control. Discovery of quality problems by waiting for system failures is no longer acceptable given the critical role software plays in embedded systems and mission critical applications[4,9,18]. Testing, verification, and inspection have been shown to be an ineffective strategy for software development. The higher order specification (HOS) method and the iterative life-cycle have been developed to combat such problems[5,6]. These methods emphasize the development of complex software from less complex components that can be completely tested or mathematically proven (HOS).

However, these risks cannot be reduced by addressing the software product after it has been produced, risk reduction must occur as part of the software development process[8,11,13,16,19]. Proponents of the application of TQM techniques to software engineering hold that there is a need to shift from an product focus that introduces quality through inspection to a process focus that introduces quality into the software development activity. Unfortunately, there has been little progress on the application of tools of TQM to the software development function. Many firms attempt to apply directly the seven tools, and seven new tools of TQM[20] such as statistical process control (SPC) or quality function deployment (QFD), without allowing for the differences between manufacturing and software development.
TQM techniques in software development

There have been calls within the software development and quality literature for improvements in the application of TQM tools to the software engineering process[7,9,13,21,22]. The Software Engineering Institute (SEI) framework for development quality closely follows the company-wide quality control (CWQC) framework that is the foundation of the quality strategies used by the Japanese[23]. The top layers of both these frameworks emphasize process quality and improvement.

Card[22], and Mills and Poore[24] describe the use of SPC techniques to evaluate the capability and stability of development efforts. The methods proposed are based on observed defects and mean time between failure rates. This represents the use of “attribute” data for control purposes, which is the most rudimentary form of SPC[15,25]. Further, defect detection has been criticized as inappropriate given the critical role of software to the operation of many businesses and to embedded systems[18]. “Variables” data based on the identification of critical product characteristics is a more advanced form of SPC[4,15,18,25].

Establishment of critical product characteristics is based on the analysis of customer requirements[26]. QFD is described in the quality literature as a method for translating high-level customer requirements into process specifications. Critical characteristics and their specifications are derived from the process specifications developed during QFD[27,28]. The “voice of the customer”[28], which is the highest level of the CWQC framework, provides the basis for establishing process control, that is the highest level of the SEI classification scheme.

Software quality function deployment

The basis of QFD is to obtain and translate the needs of the customer, in their own words, into a set of detailed design specifications that can be used to guide all phases of the production process. This same objective lies at the heart of the software development life-cycle. However, traditional development methods are lacking in their ability to translate customer needs into process specifications.

Criticisms of traditional requirements gathering approaches

Traditional approaches and life-cycle methods for the development of software do not provide a rigorous “formal” way to obtain customer needs that must be translated into system requirements[8]. The functional requirements of a software system are typically collected through series of interviews with users and subject-area experts. Requirements and information about the logical operation of the proposed system are then converted into models that document the developer’s understanding of the business[3,6,29]. There are a number of criticisms of this process. There are no disciplined methods to move from high-level customer needs or functional requirements to more detailed technical requirements[5,6,30]. Structured methods described by Yourdon[3], McMenemy and Palmer[29], and Martin[6] explicitly separate considerations
of logical and technical requirements across the development life-cycle. System planning does not address the issue of technical requirements and the trade-offs imposed by implementation strategies. Separating the consideration of technical requirements temporally in the development life-cycle does not permit the development of a coherent set of software specifications[8].

Customer needs versus user specifications. Many development methods blur the distinction between function and form in gathering the specifications for a software system. There is not a definite distinction made between high-level customer requirements and end-user specification. Prototyping as a specification method is an example of gathering this mixture of specification types. A number of prominent authors in the systems development discipline describe prototyping as a method to elicit end-user specifications[3,5]. The issue ignored is that satisfaction of the fundamental business needs of a customer (not necessarily the end-user) may require multiple business systems.

Lack of requirements interaction. The relationships between customer requirements are also largely ignored in current development methodologies. Determination of the degree of importance or priority of any particular requirement is not established. Further, the impact of the requirements on each other is not also addressed in any major development methodology. All requirements are treated with equal attention, trade-offs between requirements are established subjectively based on statements from the user.

Applying QFD to software development
The application of QFD to the software development process has been proposed as a means to formalize the collection and transformation of customer needs into a set of system design specifications. Software QFD (SQFD) provides a formal method to link high-level customer requirements to specific system requirements in a coherent fashion[8,16]. Additionally, SQFD enables a development team to evaluate their design based on:

- quantified statements of customer priorities;
- explicit representations of the relationship between requirements.

Haag[10] describes how a number of major corporations have adapted QFD to document user specifications. Large development organizations such as Hewlett Packard, IBM, and DEC have used this quality tool to great advantage in improving the quality of applications delivered to customers[31]. Much of the work in applying QFD to software development though, remains proprietary. Examples of SQFD are very rare in the published literature, compared with the application of QFD in manufacturing. Zultner[8,32], and Eriksson and McFadden[16] have presented non-proprietary methods for conducting SQFD. The focus of this article is on the application of SQFD for the development of application software as an end product, models for building software as part of embedded systems is beyond the scope of this study.

Eriksson and McFadden implementation of SQFD. Eriksson and McFadden[16] apply SQFD to the development of a hypothetical information
system for Metpath Inc., a clinical testing laboratory. The process of performing QFD for the system is described through a narrative of the development team meetings for the project. The SQFD method presented by the authors involves a series of three relationship matrices that relate requirements, “what” must be included, to implementations, “how” the requirements are addressed. Figure 1 shows the basic configuration of relationship matrix.

The first matrix developed by the authors compared customer requirements (“what”) versus software characteristics (“how”). This comparison closely follows the manufacturing QFD model where customer requirements are translated into technical characteristics of the product[23,28]. The vertical axis of the matrix listed the customer requirements that were gathered. A relative importance rating for each requirement was then listed. This importance rating quantitatively demonstrates what the customer perceives to be the most important requirements of the new system. Prioritizing requirements allows the development team to focus time and resources on the critical few requirements that directly contribute to customer satisfaction[8,25]. Software characteristics were then listed across the top of the matrix.

The relationships between the software characteristics and customer requirements were then evaluated. Relationships were characterized as strong, some, and negative. Definition of the strength of the relationships permits the development team to see the potential trade-offs between characteristics of the software and customer requirements. The inclusion of a negative relationship condition provided recognizes the concept of suboptimization, demonstrating that not all characteristics are mutually supportive of the final product.

The second matrix presented by the authors translates the software characteristics into a set of production features of the software product. This translation is performed by making the software characteristics the “what” element in the matrix and developing a set of features to achieve these

![Figure 1. TQM relationship matrix](image-url)
Application of QFD to software development

This presentation of SQFD primarily uses the matrix tool of TQM without the extensions that define QFD. The rich information that can be gathered through QFD as it is implemented in the quality literature is not obtained. The authors did not employ the “house of quality” form (see Figure 2) that typifies QFD in manufacturing[20,26,28]. This more traditional form permits the comparison of current function to the competition, and the evaluation of the relationships between the implementation characteristics or “hows”[20,28].

Zultner model of SQFD. Zultner’s[32] model for applying QFD to software development is quite extensive and detailed. There are 14 basic matrices used in the model and an in-depth presentation of each is beyond the scope of the current study.

Haag[10] has summarized Zultner’s model for SQFD into four interrelated sets of matrices that link customer requirements to software system specifications and supports detailed a priori planning of a software development project. The first set of matrices in Zultner’s model is the Z matrices. This group of matrices extends the QFD method described by Hauser and Clausing[28] by developing an understanding of the customer. The “Z-0” and “Z-1” matrices identify customer characteristics, the major groupings of

Figure 2. House of quality matrix
customers based on these characteristics, and prioritizes these groupings. Customer requirements are then identified that satisfy the customer groups. Understanding the customer is a fundamental aspect of the continuous improvement philosophy as it is described in the current literature and is a logical aspect of deploying the “voice of the customer”[33-35].

The next phase of the Zultner model involves the translation of the user requirements into a set of technical requirements. The techniques employed are very similar to those presented in the quality literature by Hauser and Clausing[28] and Sullivan[27]. Zultner[32] does not employ the characteristic “house of quality” form used by other proponents of QFD. Instead, the competitive assessments are broken down into several sub-matrices. Quantitative evaluations are computed that show where the organization stands in relation to competitors on specific requirements, both technical and customer. Issues such as planned improvement and difficulty level are also examined as customer requirements are translated to technical requirements.

Zultner[32] has modified the progression of matrices used in QFD for use with software development. A set of “T” type matrices[20] like the matrix shown in Figure 3 are used to translate technical requirements into the entity types and processes (process is used here in a programming context). The relationship strength of technical requirements to both functional processes and entity types is evaluated forming hardware and software deployments respectively[32]. Again, competitive assessments are broken into separate matrices. The relationship of task requirements to technical requirements is also developed to provide the “service deployment” for the system[32].

![Figure 3. “T” type matrix](image-url)
The impact of new technologies are determined and evaluated during the third phase (E matrix set) of SQFD in Zultner’s model. New technologies are identified along the y-axis of the “E-0” matrix[10]. The new concepts that are necessary to implement these new technologies are then elaborated along the x-axis. These new concepts are then related to the stated customer requirements, technical requirements, and to the entity types and processes required to meet the technical requirements. This set of matrices provides an assessment of the impact of new technologies on the design of the finished system. The fourth phase examines the impact of failure modes on the customer requirements. The matrices used in this phase are similar to those used in the examination of new technology. Failure modes are identified and then related to the stated customer requirements, technical requirements, and to the entity types and processes required to meet the technical requirements.

Discussion

Despite the apparent benefits of applying QFD to software development, this technique has not received widespread discussion in the literature, nor has it been incorporated into modern development methods such as information engineering (IE) or object oriented (OO) methods. SQFD remains largely misunderstood as an improvement in how software is developed. The most likely explanation for the lack of acceptance is the absence of a systemic approach to SQFD that is specific to the needs of the modern software development process. This section will evaluate the capability of the two SQFD models presented to support modern systems development.

Criticisms of current SQFD models

The SQFD model presented by Erikkson and McFadden[16] attempts to apply the matrix tool from the TQM seven new tools directly in a fashion similar to QFD. A major aspect of QFD is the development of competitive assessments, sales points, and planning goals for the design of the product[23,28]. Also, by not employing the “house of quality” form, developers are not able explicitly to examine the relationships between implementation deployments or the “hows” listed along the x-axis of the matrix. The effects of product or process characteristics remain unknown, and therefore cannot be improved.

The most serious criticism of Erikkson and McFadden’s[16] SQFD model is that the end product consists of a design for performing the development and not for a software product. Customer requirements are translated into process metrics that meet the needs of the development function for maintaining the software. There is no statement of how the customer needs will be met. This highlights a major point of confusion in the application of QFD to the software development process. Although QFD is a method of process improvement, the result of using QFD is a set of design specifications for production. The actual improvement to the software design process is the formal deployment of customer requirements into design specifications.
Like Eriksson and McFadden[16], Zultner’s SQFD model does not use the conventional “house of quality” form (see Figure 2) for his QFD matrices. The Zultner[8,32] SQFD model though, is more extensive than that proposed by Eriksson and McFadden[16]. Manufacturing QFD is modified to some extent to support software development through the addition of hardware and software deployment activities. Zultner also introduces an extension to the QFD methods found elsewhere in the literature by defining and prioritizing customers. This is an important addition to the concept of QFD in that just as not all customer requirements contribute equally to customer satisfaction, not all customer segments contribute equally to the value of a process supported by a software application. However, Zultner[8,32] does not draw the connection between customer segments and the organizational processes (process in this context implies high level, cross-functional activities that provide value to the customer).

The most significant shortcoming with the two SQFD models presented in this study is that they both attempt to apply manufacturing QFD (MQFD) directly to software development. Production of manufactured goods and the production of an information system are quite different in several respects. These differences make the direct application of MQFD methods and examples to software development difficult for development teams. The cognitive difficulties in applying QFD techniques may account for the lack of widespread acceptance of this lauded technique.

Unique needs of the modern software development process
There are unique characteristics of software development that necessitate the modification of conventional QFD or MQFD for use in deploying software designs. The first of these differences is that software development does not possess a repetitious production process that must be designed for each product. MQFD was originally conceived as a way of deploying customer requirements throughout the production process of a product offering, from design to manufacturing[23,27]. Software elements are developed to a set of design specifications. In MQFD this would equate to part or product deployment[28]. The software is then duplicated (in the case of a commercial software house such as Microsoft) or used (in the case of in-house development). Program duplication for a software house is part of a manufacturing process and is not germane to SQFD.

Process characteristics or variables in software development that have been proposed for process control purposes such as complexity, clarity, and modularity[1,16] are actually manifestations of the development methodology. The development methodology in essence is the production process. However, methodological considerations remain relatively constant across all software products developed using that methodology, variation in process quality is in the form of adherence with policies for development. Deployment of production process specifications is essentially not pertinent to software development at the individual project level.
The second major difference between manufacturing and software development is that customer requirements are not directly met by specific technical specifications. Unlike manufactured goods, software or information systems are typically intended to provide a support infrastructure or to act as an enabling mechanism for an organizational process[36-39]. In MQFD, customer requirements are translated into technical specifications for a particular product (i.e. materials, or weights)[20]. In the case of software or more accurately information system development, customer needs are met by providing certain system functions. These system functions may require the development of one or more software systems. For example; the customer need “accurate inventory information” would require the construction of inventory tracking software, purchasing and receiving software, and cost management software. These individual program elements would require an integration structure finally to meet the stated customer requirement. A single or set of intermediate translations is required before customer requirements can be deployed as a set of technical requirements.

Finally, modern software development methods must cope with issues not relevant to manufacturing. Some of these issues are the use of code libraries and reuse, as well as distributed processing. Banker and Kauffman[40] make the argument that the level of code reuse is a measure of the quality of a development process. The wide array of options for performing client-server processing requires that a design deployment considers dispersion requirements[41,42]. The SQFD models presented by Erikkson and McFadden[16] and Zultner[8,32] are grounded in the traditional waterfall life-cycle, which is not greatly different from the manufacturing design activity.

The current discussion does not intend to imply that there does not exist a model of SQFD that is uniquely tailored to the development of software or infrastructure services in general. Haag[10,31] describes the great success derived from the use of SQFD by a number of companies. Unfortunately, the methods that companies such as Hewlett-Packard and others have employed with such success remain proprietary due to the competitive advantage they afford the company. As the current study has shown, there has been only a limited amount of work published in the literature dealing with how to modify or apply QFD to the software development activity. In the next section a QFD model is proposed that is specifically tailored to the deployment of customer requirements through all stages of the software development life-cycle.

Proposed software-specific model for QFD
QFD is fundamentally concerned with deploying the “voice of the customer” throughout the production process, from design to manufacturing operations, of a particular product offering[27,28]. The final product satisfies some set of customer needs that are identified to be important using QFD. However, software development produces a product, or system of products, that supports the performance of some activity that occurs as part of an organizational process[6,39].
The proposed SQFD model shown in Figure 4 adopts a more holistic approach than used in previous works. Specifically, the SQFD model presented here supports the development of software for the information technology infrastructure of an organization. This orientation is based on two assumptions about the development of software in a modern enterprise.

(1) The role of software in an organization is that of an enabler of core enterprise processes \[37-39\].

(2) Development of individual software solutions does not optimize the value added to the enterprise by information technology \[2\].

Software in this context is not an end product, but is an enabler for the improvement of enterprise operations. As an enabler, software must provide support for those process stakeholders who provide value to the organizational process. This proposed model of SQFD provides the much needed link between the process and the enabling technology project.

Analysis begins with the identification of the customers and their relationships to the organizational processes supported by the software project under consideration. This process proceeds to develop the needs of value-adding customers from high-level requirements into a set of software implementation requirements. Through this focus on deploying the voice of the value-added customer, the organizational impact of the software development process is improved. The benefits of improved customer requirements specification are amplified by focusing on the most vital customers.

Figure 4.
Proposed SQFD model
The tool used to perform this analysis is the “house of quality” matrix described by Hauser and Clausing[28] shown in Figure 2. This form permits the development of quantitative measures of competitive standing, sales points, and customer preferences. Further, the addition of the interaction matrix along the horizontal axis, the “roof”, highlights the relationships between implementation deployments or “hows”. The quantitative measurements and assessments described in other works on QFD can be applied in each of the matrices developed with this model. A detailed description of how each matrix is constructed is beyond the scope of the current study. The discussion presented in the following sections will focus on the intent of each particular phase of the SQFD model and highlight areas that have been significantly modified to create a software specific QFD model.

Customer deployment phase
The implementation of any QFD approach must focus on customer for product under consideration and how the needs of that customer can be met. Conventional approaches to QFD start from the assumption that the customer is known and is a relatively homogeneous group. However, software is implemented to support some activity involved in performing an organizational process. The customers in the context of software development are the process stakeholders. The number and diversity of these groups of stakeholders or customers in an organizational process can potentially be quite high.

The SQFD model proposed in this article begins by identifying customers or stakeholders as a first step in developing software to meet their needs. The first matrix that is constructed establishes the organizational processes as the “whats” along the vertical axis of the matrix. Techniques for identifying organizational processes are described elsewhere in the literature and will not be described here. The processes that are identified can be prioritized based on management guidance or their relationship to the core competence of the enterprise. Emphasizing processes supportive of the core competence of the enterprise further leverages the development effort by focusing on areas that contribute most to the profitability of the firm[43,44].

Along the horizontal axis of the matrix those persons or stakeholders who contribute value to the organizational processes are listed. Essentially, these stakeholders are the “how” for implementing the processes that have been identified. These groups of stakeholders are the customers of the software that is intended to support one or more organizational processes. This breakout of the process stakeholders or customers should be at a level sufficiently detailed enough to permit a clear evaluation of the value added by customers to the identified processes.

The customers that are identified to implement the organizational processes each contribute some level of value to that process[39]. The amount of value added can be positive, zero, or negative. The concept of non-value added or negative value added is very important to the improvement of organizational processes[45]. The software requirements of these customers do not have to be
met to provide value added support for a business process. In the case of negative value added, the needs of these customers should not be met and are targets of other process improvement efforts[34,45]. The value added by each customer to the organizational processes is recorded in the relationship matrix (the intersection of the rows and columns). Symbols or numbers may be used to reflect the value added by each customer to the identified processes[20,27,28].

In the roof matrix area the interrelationships of the customer groups can be explicitly identified. This activity supports the identification of similar customers across functional boundaries. The potential to achieve synergy between customer groups in developing software support can be explicitly identified.

These evaluations and the relative importance rating assigned to each process can be used to derive a quantitative measure of the importance of each customer segment to performing the processes that comprise the organization. This measure, referred to in this model as the value-added factor, is computed by multiplying the relative importance rating by the value-added evaluation, and then summing the products for each customer. Some texts on QFD also provide methods for adjusting the measures derived from the main body of the matrix with measures of interrelationship computed for the roof matrix. A detailed treatment of these methods is beyond the scope of this article.

By focusing on the organizational processes and their various stakeholders, the sources of value activity are explicitly identified. This provides a quantitative statement of who's needs must be met to provide the most effective support for organizational processes. Customer requirements should be elicited from those customers that provide positive value to the process targeted for software support.

Business process deployment phase
With the selection of a particular organizational process as the target of software support, the value-adding customer segments identified for that process become the “whats” in this phase. The process customer segments that contribute positively to an organizational process are listed on the vertical axis of this “house of quality” matrix. Meeting the software needs of these customers provides the maximum opportunity to improve the technology infrastructure supporting the selected organizational process. The relative importance factor assigned to each of these process customer segments in this proposed model is the value-added factor computed in the previous phase. By using this measure, management priorities and actual value contribution can be deployed into the requirements specification for the software project. This addresses the issue of aligning strategy with software requirements[36,46].

The business process requirements of each of the customer segments are then developed along the horizontal axis of the “house of quality” matrix. This activity is described in detail in most published works on QFD. Customer requirements or attributes are expanded into greater levels of specificity through primary, secondary, and tertiary requirements[27,28,32]. The
significant difference between this model and earlier works is that there is a direct link to the business process. Customer requirements in this model are the implementation deployments or "hows" that enable customer segments to add value to the organizational process under consideration. Hence, the requirements for performing the business process are deployed into the design of the supporting software.

The relationships between process customer segments and customer requirements in this phase reflect the importance of each requirement to a particular segment. Indicators of importance are placed at the intersections of the rows and columns in the relationship matrix. These indicators can then be used to derive an objective measure of customer priorities. This customer priority measure quantitatively measures the role of customer requirements in adding value to an organizational process (since the value-added factor is used to compute this measure).

A benefit of using the "house of quality" matrix format is that the interactions of implementation deployments can be explicitly defined. This type of analysis is particularly important in the current and subsequent phases of the proposed model. An issue that is ignored in the software development literature and previous SQFD models is the interaction of requirements specifications. Software requirements are not required to be mutually supportive or neutral in their interactions with each other. The roof matrix is used in this phase to define the interactions (supportive, neutral, or negative) between interaction deployments. In this phase the interactions between customer requirements are evaluated just as the relationships between attributes ("whats") and implementations ("hows") are defined.

The output of the analysis performed in this phase is a set of prioritized customer requirements that are explicitly linked to the business process that the software is intended to support. This set of requirements has also been evaluated with regard to their impact on each other. These impact evaluations highlight potential conflicts and synergies in the development process.

Functional requirements deployment phase
One of the problems with directly applying the manufacturing model of QFD to software development is that customer requirements or needs are not directly met by the technical characteristics of the software. Customer or process needs in software development are met by the functional requirements or functionality of the software[3,29]. This phase of the proposed SQFD model identifies and deploys the required functionality of the software support that is needed to meet the prioritized set of customer requirements. Introducing this intermediate step between customer needs and technical or engineering requirements provides a closer conceptual link between the requirements specification process (QFD) and development methodologies.

This phase of SQFD supports the development of what the QFD literature refers to as a "planning matrix"[8,27,28]. Assessments of the organization's ability to meet customer needs and to provide the desired functionality are
The results of these assessments provide decision-making information. In the context of an organizational information systems development project, a competitive assessment may be used to support decisions regarding in-house development versus outsourcing to meet customer requirements. Commercial software developers would use the competitive assessment the same way that it is described in the QFD literature. The development of sales points would be particularly useful to commercial software vendors. Technical assessments provide a quantitative evaluation of the ability of organization to provide the required levels of functionality. Hauser and Clausing[28] describe methods for evaluating the feasibility of meeting implementation requirements.

The assessments and requirements weightings that are obtained during the development of this matrix have implications beyond improving the quality of software requirements specifications. These measures provide the software development organization with a quantitative assessment of their ability to support organizational processes. The ability of current systems functionality to meet the needs of value-adding customers can be assessed through the analysis performed in the relationship matrix. Such an analysis provides the development organization with:

- quantitative indicators of changes in organizational priorities;
- indicators of the effectiveness of current systems.

The latter point supports continuous improvement efforts on the part of the organizational software development function.

Procedural requirements deployment phase

The functional specifications identified in the previous phase of the SQFD model are high-level statements of the functionality required to satisfy customer needs. These requirements must be translated into a set of procedural specifications that can be implemented as an automated information system. Manufacturing oriented QFD models emphasize the translation of high-level requirements into technology characteristics of the product, i.e. material types or weights. Hauser and Clausing[28] refer to this as “parts deployment”. However, in software development highly technical requirements such as multi-user database or networks constitute the means of product delivery or use. Functionality is delivered through the implementation of procedural logic, information stores, and distribution mechanisms (that provide access to information and processing). The translation of functional and customer requirements into implementation deployment is equivalent to the transition from analysis to design in a software development methodology.

In this phase, a series of three “house of quality” matrices is constructed to deploy functional requirements into components (parts) that can be implemented. This method is similar to the model proposed by Zultner[32]. Prioritized functional requirements identified in the previous phase are deployed into implementable procedures. This forms the specification of the
major system component that will be required to satisfy the functional requirements of value-added customers.

Procedural requirements are then deployed through the identification of information needs (data requirements), and distribution requirements (see Figure 5).

Deployment of functional requirements into procedural, data, and distribution components is based on the work of Zachman[47]. Zachman's framework identifies these three components as the primary areas of concern in the construction of an information system. A blueprint of the software system is now formed that establishes how functional requirements are to be met along with the information and distribution patterns needed to implement these procedures. Essentially, this deployment provides a formal transition from analysis to design that ties the resulting software to supported business processes.

**Conclusion**
The consequences of poor software quality will become more pronounced as the importance of systems increases in the operations of modern organizations. Past methods for ensuring software quality have focused on testing and inspection to build quality into the software product. However, the increased complexity of modern software makes the likelihood of testing every interaction quite remote.
The solution proposed to this crisis in software development is to build quality into the process of constructing software\cite{7,9}. TQM tools such as QFD have been proposed as mechanisms to implement the improvement of the software development process. However, these TQM tools have not become widely used for software development.

One difficulty with incorporating these techniques into the software development literature is that organizations which employ them attain significant advantage in their development efforts\cite{10}. Given the advantage derived, application experience with these TQM tools remains proprietary in many organizations. The most significant problem with the current SQFD found in the literature is that they attempt to apply QFD methods without modification to the software development activity.

This article has proposed a four-stage model for performing SQFD that is grounded in both the quality and software development literature. Software requirements are developed based on the requirements of customer segments that positively contribute to the value of organizational processes. The series of “house of quality” matrices that is developed links the software product design to the supported business process. This serves as a formal method of obtaining system requirements that is supportive of enterprise priorities and that is quantitatively based.

Areas for future research
The model proposed here is not intended to reflect the definitive SQFD methodology. This model is still very much grounded in the conceptual work found in the quality literature on QFD. The progression from matrix to matrix and the weighting methods are taken directly from the published works on QFD. Further study is required to determine if this progression does indeed support the development of software to support business processes. Specific issues include:

- refinements to the deployment scheme used for SQFD;
- the development of meaningful quantitative measures to evaluate the priority of requirements;
- the development of quantitative measures to support trade-offs between implementation deployments;
- formal feedback mechanisms to evaluate the level of improvement attained in meeting the support requirements of business processes.

References
43. Automation & Robotics Research Institute, Perform Continuous Enterprise Improvement Model, working paper, Automation & Robotics Research Institute, University of Texas at Arlington, Arlington, TX, 1991.