CHAPTER 6

Q-RPDI: MODEL FOR A QUALITY SOFTWARE

6.1 Introduction

Computer systems are used in many critical applications where a failure can have serious consequences. Developing systematic ways to relate the software quality attributes of a system to the system’s architecture provides a sound basis for making objective decisions about design trade-offs and enable engineers to make reasonably accurate assumptions. The ultimate goal is the ability to quantitatively evaluate and trade-off multiple software quality attributes to arrive at a better overall system. The definition of software quality given by few technical authors is in terms of “Fitness for purpose”, but more recent commercial thinking would not support this description. Ince describes the modern view of quality: “A high quality product is one which has associated with a number of quality factors” [1]. These could be described in the requirement specification; they could be cultured, in that they are normally associated with the artifact through familiarity of use and through the shared experience of users; or they could be quality factors which the developers regard as important but are not considered by the customer
and hence not included in the requirement specification. Software quality is the extent to which an industry defined set of desirable features are incorporated into a product, so as it enhances its lifetime performance.

6.1.1 Software Quality Models

Quality consists of those quality features which meet the need of customers and thereby provide product satisfaction. Software quality models are a well-accepted means to support quality management of software systems. Over the last 30 years, a magnitude of quality models have been proposed and applied with varying needs of success. Research on software quality is as old as software research itself. As in other engineering and science disciplines, one approach to understand and control an issue is the use of models. Therefore, quality models have become a well accepted means to describe and manage software quality. “Quality model” is a schema to better explain of our view of quality. Process of building a quality model decomposes in two main tasks generally:

i) choosing a super characteristic/ factors

ii) choosing and organizing characteristics related to super-characteristics/factors.

A quality model is used to document or analyze the required or actual quality of an application. One can also use a quality model to document or analyze the planned or actual quality of a business unit. Quality models must evaluate high-level quality characteristics with great accuracy in terms well-known to software engineers to help
maintainers in assessing programs and thus in predicting maintenance efforts. Such quality model can also help developers in building better quality programs by exposing the relationships between internal attributes and external quality characteristics clearly.

The last three decades in quality modeling generated a magnitude of very diverse models commonly termed as “quality models”. Although definition, assessment and prediction of quality are different purposes, they are obviously not independent of each other. It is hard to assess quality without knowing what it actually constitutes and equally hard to predict quality without knowing how to assess it. The DAP classification views prediction models as the most advanced form of quality models as they can also be used for the definition of quality and for its assessment. Thus a quality model can be defined as: “A framework to define, evaluate and improve quality” [2]. This usually includes a quality meta-model as well as a methodology that describes how to instantiate the meta-model and use the model instances for defining, assessing, predicting and improving quality. A meta-model is required to precisely define the model elements and their interactions. Quality models should act as a central repository of information regarding quality and therefore the different tasks of quality engineering should rely on the same quality model. The variety in software systems is extremely large, ranging from huge business information systems to tiny embedded controllers. These differences must be accounted for in quality models by defined means of customization. In current quality models, these differences are not considered [3][4].
6.1.2 Goal Directed Approach

A goal-directed approach to build a quality model for software is effective for accommodating and balancing the needs of the different interest groups. The set of desirable properties or quality attributes of software provides an abstract or high level specification for what we will call software product quality. A software characteristic is an abstract property of software that classifies a set of tangible quality-carrying product properties. Modularity is an example of a software or product characteristic. Software characteristics may correspond either to a set of functional entities or a set of non-functional tangible properties. Software characteristics help software to satisfy quality attributes and may be used to support the definition of high level quality attributes.

Following are some additional design principles and guidelines:

i) We choose to associate abstract properties, called quality attributes with software.

ii) The quality of software may be characterized by a set of high-level quality attributes.

iii) The quality attributes of software correspond either to a set of domain-independent behaviors of software or a set of domain-independent users of software.

iv) The quality attributes of a quality model should be sufficient to meet the needs of all interest groups associated with the software.
v) Each high-level quality attribute of software is characterized by subordinate properties which are behavior, users or software characteristics.

vi) Software characteristics can be either generic, product specific, language-specific or domain-specific.

vii) We associate abstract properties, called quality attributes, with software to characterize its desirable behavior and uses.

For building a comprehensive model for software product quality, it is useful to have a framework into which software characteristics and their defining tangible properties can be fitted. Such a framework can help us systemize and structure our knowledge about software. It can also point to area where there are gaps, misclassification and other problems in our knowledge and understanding of software characteristics. Over time we can correct, refine and improve the framework to more accurately suit our needs. The different quality models have advocated somewhere between six and twenty quality attributes at the highest level.

6.2 Limitations of Existing Models

Like most of the other quality models, we have concentrated on the quality attributes that pertain to software designs and implementation. The architectures of most models only go as far as providing architecture for quality. They neglect to include model for software or any sort of explanation for how quality attributes can be mapped on to the software. Each phase of the development of the software prescribe the high-level quality attributes
that influence the quality of the software. This lack of structure hampers the
efforts to make software product quality intellectually manageable,
implemental and applicable.

6.3 Q-RPDI Model

Our architectural proposal for software product quality has been designed to
overcome the efficiencies in earlier models. The newly introduced quality
model represents the various phases of software development and their
quality parameters associated. Model is based on the fact that” the improved
quality of each phase of the development process increases or improves the
quality of the final product”. In any development process, the improved
quality helps in some phases like requirement engineering phase, and then
the design phase that includes modeling of both process and data and then a
quality user interface can provide a quality interactive system to the user.
Thus in our quality model, we improve the quality of the outputs of the
defined phases of the development process of a software which will finally
help to improve the quality of the final product. The newly introduced quality
model is defined as below:
6.3.1 Quality Requirement

In the quality requirement phase, the most important task is to give right and correctly formatted answers in the form of requirements. It is necessary to decide on the system requirements of different stakeholders. Different people associated with the software like system analyst, customers, end-users etc. can have some kind of communication gap due to some technical terms used by the analyst which may not be understood by the common end-user who might prefer to communicate through natural language. This communication
gap may lead to unfulfilled expectations of the stakeholders. Therefore, careful analysis and validation of requirements are considered as important ingredients for the success of software system development.

An SRS framework is introduced which gives the final product as quality requirement specification. The framework inhibits an iterative behavior where a repeated execution of all the steps of requirement engineering can improve the quality requirements by subsequent evaluation of stakeholders. In this process, requirements gathered from various sources are collected at one place and the process of fusion will collect the related and similar requirements and store them in groups. This process of fusion will help the elicitation process to select the intended requirements from the predefined group of requirements. The useful and necessary requirements are then elicited using scenarios. Requirement elicitation is followed by analysis. Analysis of requirements is done to detect and resolve conflicts between requirements and to elaborate system requirements to software requirements. Analysis is basically done to elaborate and refine requirements. An informal specification is created that documents all the necessary requirements. This document is evaluated by the stakeholders to identify any lack of requirements or unambiguous requirements being included in the specification. This framework will provide a formal document of requirements that will include quality requirements.

A great deal of formal evidence exists that the poor quality of individual requirements or requirement specification that document
them is a primary reason why so many projects continue to fail. Requirement elicitation involves sifting through large amounts of information and deciding what exactly is relevant. Scenarios are one of the most successful ways of eliciting requirements. A scenario is a description in natural language of a particular sequence of interactions between the user and the system. During requirement elicitation, scenarios can help users to explain what they do to carry out particular tasks, what information they need and what the output will be. To elicit quality requirements, a scenario functionality model (Fig 2.5) is developed. This functionality model represents the structures of all the scenarios describing interaction between user and the system.

Various quality parameters like completeness, consistency etc. is elicited here and leads to the development of a high quality requirements specification. The development process that relies heavily on scenarios must be very sensitive to issues of completeness. System and requirements evolve and thus it becomes very difficult to produce a complete requirement specification. Every scenario represents a partial specification that is coherent from user’s perspective. Another important quality parameter is consistency which has to be maintained across scenarios. An explicit integration of scenario description and their implementations can promote consistency in the overall system. The proposed scenario functionality model provides a framework for organizing partial specifications, keeping them consistent. One important concern while specifying the requirements is to define the adequacy of requirements. Sometimes customers are unable to specify clearly what they want because of
which the assessment of adequacy becomes very difficult. The scenario functionality model places the requirements in an environment which is similar to the system that will be used by the user. The model allows continuous checks on the requirements that are elicited against the customer’s intentions which results in adequate specifications.

In most requirement specifications, it is been noticed that the requirements are stated informally in natural language which leads to ambiguity. *Unambiguity* is an important quality parameter. Using scenarios makes processes possible that use short cycles between writing and validation requirements that define test cases derived from the scenarios early. Such processes together with user orientation of scenarios yield powerful capabilities for detecting and resolving unambiguity. Thus the scenario functionality model provides a natural means of detecting and resolving ambiguities in communication between humans. Thus the proposed scenario functionality model helps in improving the quality of requirements that are formally represented through requirement specification.

6.3.2 Quality Process

In the quality process design phase, the most important task is to improve the quality of the process design to achieve effective and efficient deliverables. Software design encompasses the different ways of presenting the system through components from different point of views. It is not just influenced by structure and behavior but also by
use, functionality, performance etc. A design is indeed in order to understand the system by breaking the system into subsystems and defining the various interfaces and their relationships. Software process design should not be considered as an independent activity, but a step further in the development and evolutionary process in software products [7].

Modern applications including distribution, portability, interportability, reusability etc. require an early definition of the system architecture in order to achieve overall functional purpose of the software system under construction. For this, a Quality Process Design Framework for process design is introduces (Fig 3.1) where the activities are divided into sub-activities and arranged in an iterative order so as to improve the quality of the design decisions.

Important architectural quality characteristics to be taken into account while developing a process design are maintainability, reliability, security, functionality and efficiency. Functionality is described as a system’s capacity to do the work for which it is proposed. Since any possible number of structures can be conceived in order to implement any functionality, it is not an architectural characteristic by nature. Systems evolve even if they are under construction. This is why maintainability has to be considered as one of the architectural design quality characteristics to be taken into account. Security is defined as the ability of the product to be defended against unauthorized use. Data and information are one of the most important properties and thus security should be given high profile. Reliability is defined as the system’s ability to remain operational over
time. Reliability is related to fault tolerance and the time it takes to recover, both aspects being attainable through the architecture. Thus reliability must also be considered a quality characteristic to be propitiated by design. Efficiency is related to the time required to respond to a particular event. Since efficiency can be measured on the basis of the amount of information and communication between system components, which is a design characteristic, efficiency must also be taken into account when it comes to guaranteeing product quality through design.

To check the quality of the design, an approach is used which provides a framework in which global and locally defined quality criteria can be considered. Therefore this can be used as a standard of excellence measure for a product quality. In this a relationship chart (Fig 3.2) is created which displays graphically the relationships between quality characteristics. Then a polarity profile (Fig 3.3) is created which produces a consensus view of quality based on the relationships identified in the relationship chart. After this an overall quality score (Fig 3.4) for a product is calculated. The single value of quality produced by this method can be used to indicate the overall quality in terms of required and actual values. This value shows quality in terms of the percentage of quality requirements met. This result shows the impact of the relationships between the various quality factors.

The newly introduced framework ensured the maintainability factor of quality. The deliverables of the process design such as problem domain category, components, set of scenarios for quality
attributes; rules, constraints etc. are required to be of good quality. By executing this framework, any error or fault can be easily rectified or modified without disturbing the other activities. Reliability includes maturity i.e. average time between failures, fault tolerance, recovery etc. Since any change made in the process design will not affect the performance, the final product of the framework will be in accordance to the desired results. Since the definition and revision of quality attributes are evaluated through an iterative process, any change or fault can be easily modified without affecting the whole process, thereby increasing the efficiency of the system. Thus, this approach produces reliable products having high efficiency and maintainability.

6.3.3 Quality Data
Achieving quality data at early stage of database design is a critical issue for both the database researchers and practitioners. The quality data phase, the most important activity is to achieve quality data that will be stored in the repository for the further development. The consumer focused view of data is “data that is fit for use”. Data is fit for use if they are free of defects and possess desired features. Many different methods are adopted to improve the data quality like moving data from one place to another, using cleansing tools to clean dirty data. However, technical solutions cannot only remove the root causes of poor quality data because poor quality is not only an IT problem but also a business problem.
Dirty data is the data which is incorrect, misleading or unformatted. Different type of data errors exists due to incomplete, outdated and missing information. Different type of dirty data exists in relational databases and files: incorrect data, inaccurate data, inconsistent data, incomplete data, and nonintegrated data. To ensure data quality during the lifecycle of a system, certain guidelines must be followed. Good quality data means that the data should be complete, consistent, accurate etc. To achieve this, a Quality Data Framework (Fig 4.1) is introduced that converts dirty data into quality data. The output of the framework i.e. the quality data having the defined parameters help to improve the performance and outcome of the system. Data from different sources are collected at one place after extracting and cleansing. After conducting subjective and objective analysis of data, a comparative analysis of all the parameters of data quality is done.

The important dimensions considered for the analysis are accuracy, consistency, timeliness, relevance, accessibility and definition. Accuracy of data means that the data is valid and represent correct values. Each entry or record within the database should be correct. While editing, proper care should be taken to store the data. Exceptions or error reports should be generated and corrections should be made. Data accuracy is a major concern while making numeric data entries. Ensuring accuracy involves appropriate education and training and timely and appropriate communication of data definition to those who collect data. Consistency means that the data should be similar across applications. Data to be consistent must also be reliable. Data
should be analyzed using certain standard formulas, scientific equations etc. to achieve consistency. The parameter of data quality i.e. consistency should be mainly checked in while using dates. Therefore edits and conversions of tables must be done while storing data to ensure consistency. Definition means that each data item stored in the database should have clear meaning and acceptable values. Clear definitions help the current and future users to understand the meaning of data correctly. Appropriate comparisons and relationships between the data must be represented clearly. Definition means giving precise and clear definition of data reflecting the aim for collecting the data element. Timeliness means data are available as per information management policy and schedule. It also represents how the data are being used and their context. Timely data analysis allows for the initiation of action to avoid adverse impacts. Timeliness is an important parameter used while data extractions, transactional details etc.

Relevance refers to the meaningful data reflected in the performance of a process or application for which they are collected. The aim for collecting data elements must be clarified to ensure relevant data. Accessibility means data items should be easily accessible or obtainable and easy to collect. For accurate analysis, access to complete data should be good, otherwise results and conclusions may be inappropriate. Security guidelines for who may access the data should be clearly defined.

After analyzing all the parameters of quality of data, the errors and faults in the data can be easily identified and then rectified
optimizing the data. Optimization is done by creating quality schema and then evaluating it. Quality schemas are created by using data quality improvement programs with the help of parameters of quality of data. The data quality modeling approach provides a foundation for the development of a quality perspective in database design. Next step in optimization is to evaluate quality schema to maintain the quality of data. Thus the high quality data obtained is stored in repository for use in further development phases. This quality data stored in the repository minimizes the chances of any failure in the lifetime of system development.

6.3.4 Quality User Interface
The most important aspect of modern interactive computer systems is the level of support they provide for the underlying human activity. This level of support is encompassed in the user interface with which the user interacts with the system. User interface usability is an important factor and is considered only in terms of use of the user interface. It is thus important to emphasize on the fact that usability as an objective is synonymous with the quality of use. The objective of any user interface developer should be to design and implement quality user interface.

Quality user interface is defined as any user interface that is easy to use and allows the users to maximize their efficiency and effectiveness while using it. If different groups of users having different requirements use the same system, then they may require
different characteristics for a product to have quality, so that the
assessment of quality becomes dependent on the perception of the
user. Usability embraces user-perceived quality by relating quality to
the needs of the user of an interactive product. Quality of use is the
extent to which a product satisfies stated and implied needs when used
under stated conditions. Quality of use is determined not only by the
product but also by the context in which it is used.

Usability is the quality of use in context. Usability is generally
referred as the capacity of the software product to be understood,
learned, used and attractive to the user, when used under specified
conditions. Quality of use provides a means of measuring the usability
of the product. The effectiveness, efficiency, safety, satisfaction and
flexibility are the usability parameters with which the user can achieve
specified goals in specified environments. However, the attributes that
a product requires for usability depend on the nature of the user, task
and environment. In software engineering community, the term
usability has been more narrowly associated with the user interface
design. ISO/IEC9126 defines usability as relatively independent
contribution to software quality associated with the design and
evaluation of the user interface and interaction. The ideal way to
specify and measure usability is to specify the features and attributes
required to make a product usable and measure their effect on the
implemented product. The context in which the product is used is very
important since any relevant change in the context of use may change
the usability of the product. A Quality User Interface Design
Framework (Fig 5.2) defines a process for user interface design
defining various attributes required for quality user interface which are related to both the user and the product. In this framework, after planning the process, the context of use is specified and the requirement are determined. The design solution which is generated is evaluated for quality which is the most important activity.

The framework evaluates the quality using various parameters like efficiency, effectiveness, satisfaction, safety and flexibility under a specific environment in a context. It also represents the dependence of user and the product on these parameters resulting in quality of use. Efficiency refers to the measures of resources expended to achieve the intended goals. The resources may be time, which can be used to give measure of temporal efficiency, money, mental or physical effort which can be used to give measures of human efficiency. Effectiveness refers to the extent to which the intended goals of the overall system are achieved. Measures of effectiveness relate the goals or sub goals of using the system to the accuracy and completeness with which these goals can be achieved. Satisfaction is the extent to which the user finds the system acceptable. It is the extent to which the expectations are met. Measures of satisfaction describe the perceived usability of the overall system by its users and the acceptability of the system to the people who use it. Measures of satisfaction reflect the specific aspects of the system. The measures often used are: linkability, pleasure, comfort and trust. These can be measured through questionnaires. Safety refers to acceptable levels of risk of harm to people, business, data, software, property or the environment in the intended contexts of use. There are no simple measures of safety. Flexibility is defined as
the ability of a software application to deal with exceptions to the process model at the runtime and to cope with periodic changes to the process model. The parameters like efficiency, effectiveness and satisfaction can be collectively defined as functional feedback and flexibility can also be defined as application flexibility and dialog flexibility. The evaluation of user interface quality can be done quantitively by measuring the quality parameters like functional feedback, application flexibility, dialog flexibility etc.

The above defined Q-RPDI Model defines the quality parameters of each phase of the development process of an interactive system. Quality of the outputs of each phase is improved which helps in improving the quality of the final product.

6.4 Evaluation of Quality Model
Software quality engineering needs a quality model that is usable throughout the software lifecycle and that embraces all the perspectives of quality. As the current industry focus is shifting from functionality to improving quality, a new category of requirements focused on quality is emerging. A quality model provides a framework towards a definition of quality. Engineers have recognized that in order for something to find its way in a product, it should be properly defined and specified. This can be achieved by a quality model that can be used not only to evaluate software quality, but also to specify it.

The implementation of quality in a software product is an effort that should be formally managed throughout the software engineering lifecycle. Such an approach to the implementation of quality leads to software quality
engineering. Suryn has suggested that this discipline be defined as “the application of a continuous, systematic, disciplined, quantifiable approach to the development and maintenance of quality of software products and systems: i.e. the application of quality engineering to software” [5].

Software quality engineering calls for a formal management of quality throughout the lifecycle. In order to support this requirement, a quality model should have the ability to support both the definition of quality requirements and their subsequent evaluation.

Kitchenham and Pfleeger, by reporting the teachings of David Garvin, report on the 5 different perspectives of quality:

- The transcendental perspective deals with the metaphysical aspect of quality. In this view of quality, it is “something toward which we strive as an ideal, but may never implement completely” [6].

- The user perspective is concerned with the appropriateness of the product for a given context of use. Kitchenham and Pfleeger further note that “whereas the transcendental view is ethereal, the user view is more concrete, grounded in the product characteristics that meet user's needs”;

- The manufacturing perspective represents quality as conformance to requirements. This aspect of quality is stressed by standards such as ISO 9001, which defines quality as “[the] degree to which a set of inherent characteristics fulfills requirements” [7]. Other models, like the Capability Maturity Model (CMM) state that the quality of a
product is directly related to the quality of the engineering process, thus emphasising the need for a manufacturing-like process.

- The product perspective implies that quality can be appreciated by measuring the inherent characteristics of the product. Such an approach often leads to a bottom-up approach to software quality: by measuring some attributes of the different components composing a software product, a conclusion can be drawn as to the quality of the end product;

- The final perspective of quality is value-based. This perspective recognises that the different perspectives of quality may have a different importance, or value, to various stakeholders.

One could argue that in a world where conformance to ISO and IEEE standards is increasingly present in contractual agreements and used as a marketing tool [8], all the perspectives of quality are subordinate to the manufacturing view. This importance of the manufacturing perspective has increased throughout the years through works like Quality is Free and the popularity of movements like Six-Sigma [9,10]. The predominance of the manufacturing view in Software Engineering can be traced back to the 1960s, when the US Department of Defense and IBM gave birth to Software Quality Assurance [11]. This has led to the belief that adherence to a development process, as in manufacturing, will lead to a quality product. The corollary to this belief is that process improvement will lead to improved product quality. According to many renowned researchers, this belief is false, or at least
flawed. Geoff Dromey states: “The flaw in this approach [that you need a quality process to produce a quality product] is that the emphasis on process usually comes at the expense of constructing, refining, and using adequate product quality models.” [12]

Kitchenham and Pfleeger reinforce this opinion by stating: “There is little evidence that conformance to process standards guarantees good products. In fact, the critics of this view suggest that process standards guarantee only uniformity of output [...]” [6].

However, recent studies conducted at Motorola and Raytheon show that there is indeed a correlation between the maturity level of an organization as measured by the Capability Maturity Model and the quality of the resulting product [13, 14].

6.4.1 Criteria for Evaluation of Quality Model

The evaluation of a quality model is achieved by satisfying three requirements [15]:

1. A quality model should support the five different perspectives of quality as defined by Kitchenham and Pfleeger.
2. A quality model should be usable from top to the bottom of the lifecycle as defined by IEEE std 1061-1998(IEEE, 1998).
3. A quality model should be usable from the bottom to top of the lifecycle as defined by IEEE std 1061-1998(IEEE, 1998).
6.5.2 Evaluation of Existing Models

The existing models have been evaluated with respect to these requirements.

- McCall introduced his quality model in 1977 [16]. According to Pfleeger, it was one of the first published quality models [17]. Regarding this model, Pressman notes that “unfortunately, many of the metrics defined by McCall can be measured only subjectively” [18]. It is therefore difficult to use this framework to set precise and specific quality requirements. McCall model is not applicable with respect to the criteria outlined by IEEE std. for a top to bottom approach to quality engineering. Furthermore it emphasizes the product perspectives of quality to the detriment of the other perspectives. It is therefore not suited as a foundation for software quality engineering.

- Boehm's quality model improves upon the work of McCall [19]. Boehm’s model loosely retains the factor measurable property arrangement. Boehm considers that the prime characteristics of quality is what they define as “general utility”. Boehm’s model is decomposed in a hierarchy that at the top addresses the concerns of end-users while the bottom is of interest of technically inclined personnel. It is in effect the emergence of user perspective of quality. However, Boehm’s definition of the characteristics of software quality makes this interest in wane. This model is mostly useful for bottom to top approach to software quality as it can effectively be used to define measure
of software quality but is more difficult to use to specify quality requirements.

- Dromey’s model defines that a quality model should clearly be based upon the product perspective of quality [20]. Dromey has built a quality evaluation framework that analyses the quality of software components through the measurement of tangible quality properties. While dromey’s work is interesting from a technically inclined stakeholder’s perspective, it is difficult to see how it could be used at the beginning of the lifecycle to determine user quality needs. Dromey fails to build a model that is meaningful for stakeholders typically involved at the beginning of the lifecycle.

- ISO 9126- In 1991, the international organization for standardization introduced a standard named ISO/IEC 9126(1991): software product evaluation- quality characteristics and guidelines for their use. This standard aimed to define a quality model for software and a set of guidelines for measuring the characteristics associated with it. In 2001, Pleeger reported some important problems associated with the first release of 1926. Then a revised version of ISO/IEC 9126 is now a four part standard:

  ISO/IEC 9126-1 defines an updated quality model
  ISO/IEC 9126-2 defines a set of external measures
  ISO/IEC 9126-3 defines a set of internal measures
  ISO/IEC 9126-4 defines a set of quality in use measures
This model recognizes all the perspectives of quality and takes an incremental approach to software quality.

### 6.5.3 Evaluation of Q-RPDI Quality Model

Now, the Q-RPDI model needs to adhere to all the requirements for the proper and exact evaluation of a quality model and to support quality engineering. According to Pfleeger, this is an assertion that first and foremost, a software system must be useful to be considered a quality system[17]. The proposed model emphasizes on the fact that each artifact produced in the software lifecycle can be associated with a quality evaluation model. The user perspective is concerned with the appropriateness of the product for a given context of use. The model depicts the user view to a large extent by associating the product characteristics to the user’s needs. The manufacturing perspective represents quality as conformance to requirements. Here model defines quality as a degree to which a set of inherent attributes fulfills the quality requirements. The quality of a product is directly related to the quality of the output of the development phases, thus emphasizing the need for a manufacturing like process. The product perspective implies that quality can be appreciated by measuring the inherent characteristics of the product. This is achieved by measuring some attributes of the different components composing a software product, a conclusion can be drawn as to the quality of the end product. The final perspective of quality is value based. This perspective recognizes that
the different perspectives of quality may have a different importance or value to various stakeholders.

In today’s world where conformance to ISO & IEEE standards is increasingly present, all the perspectives of quality are subordinate to the manufacturing view. Thus adherence to a development process, as in manufacturing, will lead to a quality product. The proposed model encompasses these perspectives of quality by involving the improvement strategies on all the development phases of the lifecycle of a product. Once the requirements for quality are established and software construction is under way, the quality model can be used to predict the overall quality. Quality in use depicted in a development phase is directly related to the user and value based perspectives. User perspective of quality states that it is concerned with the appropriateness of a product for a given context of use. The satisfaction characteristic inherently recognizes that quality can have a different meaning and value for different stakeholders. Thus satisfaction levels can be set according to those levels of perception. The quality characteristics such as functionality and reliability of a product can be linked to manufacturing perspective of quality. Reliability, efficiency, maintainability etc. are all inherent characteristics of the product and a manifestation of the product perspective of quality. Thus the proposed model defines the quality related to both manufacturing and product perspectives. The transcendental perspective of quality relates to quality as something that is recognized but not defined. Thus it cannot be explicitly implemented in a software product.
6.5 Calculation of Quality Using Q-RPDI model

The business worth of a computer system is a function of its quality in use – the extent to which it is fitted for its purpose. ISO/IEC 14598-1 (Evaluation of Software Products) places quality in use as the overall goal for software development. The term quality in use recognizes that software does not exist in isolation, but must fit with a socio-technological work environment if it is to work in practice.

A questionnaire is prepared and is being filled by Quality Managers, Developers and Users to calculate the actual quality of the software based on the Q-RPDI model. Consolidated data and the final quality of the software based on Q-RPDI model is as follows: