CHAPTER 1

LITERATURE REVIEW

1.1 Introduction
During the past 20 years, software has conquered an essential and critical role in our society. We increasingly depend on the features and services offered through computerized systems. Any modern product or service embeds and/or exploits some piece of software. As an example, companies sell (or plan to sell in the near future) systems to automate building operations and to embed Internet-features into home appliances. Unfortunately, software applications are complex products that are difficult to develop and test. Very often, software exhibits unexpected and undesired behaviors that may even cause severe problems and damages.

One of the main directions pursued by researchers and practitioners is centered on the study and improvement of the process through which software is developed. The underlying assumption is that there is a direct correlation between the quality of the process and the quality of the developed software.

Over the past years both software engineers and users have paid increasing attention to evaluating the quality of software products [1]. This fact has lead many organisations to the development of practices enabling them to evaluate and give evidence of the degree of software quality suitable for a specified service [2].
Among others, many Italian companies have been involved in national and international projects [3] [4] concerning software quality evaluation and certification. Most of those companies increased their knowledge in real case studies and, starting from research experiences, developed proprietary measurement methodologies. Due to the state of the art of software quality evaluation, each available measurement methodology is a bit different from the other, and none of them can be considered the most suitable and credible.

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.

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. 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.

### 1.2 Software Engineering

Software has become critical to advancement in almost all areas of human endeavour. The art of programming only is no longer sufficient to construct large programs. There are serious problems in the cost, timeliness, maintenance and quality of many software products. Software engineering has the objective of solving these problems by producing good quality, maintainable software, on time, within budget. To achieve this objective, we have to focus in a disciplined manner on both the
quality of the product and on the process used to develop the product. At the first conference on software engineering in 1968, software engineering was defined as “The establishment and use of sound engineering principles in order to obtain economically developed software that is reliable and works efficiently on real machines”[8]. defined the same as “A discipline whose aim is the production of quality software, software that is delivered on time, within budget, and that satisfies its requirements”[9]. Both the definitions are popular and acceptable to majority. However, due to increase in cost of maintaining software, objective is now shifting to produce quality software that is maintainable, delivered on time, within budget, and also satisfies its requirements.

Software engineering is an engineering discipline that is concerned with all aspects of software production. Software engineers should adopt a systematic and organized approach to their work and use appropriate tools and techniques depending on the problem to be solved, the development constraints and the resources available.

Rarely, in history has a field of endeavor evolved as rapidly as software development. The struggle to stay, abreast of new technology, deal with accumulated development backlogs, and cope with people issues has become a treadmill race, as software groups work as hard as they can, just to stay in place. The initial concept of one “guru”, indispensable to a project and hostage to its continued maintenance has changed. The Software Engineering Institute (SEI) and group of “gurus” advise us to improve our development process. Improvement means “ready to change”. Not every member of an organization feels the need to change. It is too easy to dismiss process improvement efforts as just the latest management fad. Therein lie the seeds of conflict, as some members of a
team embrace new ways of working, while others mutter “over my dead body” [5].

1.2.1 Software and its Characteristics
Software is a computer program and associated program such as requirements, design methods and user manuals. Software products may be developed for a particular customer or may be developed for a general market. New software can be created by developing new programs, configuring general software systems or reusing existing software. The software has seen many changes since its inception. After all, it has evolved over the period of time against all odds and adverse circumstances. Computer industry has also progressed at a break-neck speed through the computer revolution, and recently, the network revolution triggered and/or accelerated by the explosive spread of the internet and most recently the web. Computer industry has been delivering exponential improvement in price performance, but the problems with software have not been decreasing.

Software still come late, exceed budget and are full of residual faults. As per the latest IBM report, “31% of the projects get cancelled before they are completed, 53% over-run their cost estimates and for every 100 projects, there are 94 restarts” [6]. The software is evolving day by day and its impact is also increasing on every facet of human life. We cannot imagine a day without using cell phones, logging on to the internet, sending e-mails and watching television and so on. All these activities are dependent on software and software bugs exist nearly everywhere. The blame for
these bugs goes to software companies that rush products to market without adequately testing them. It belongs to software developers who could not understand the importance of detecting and removing faults before the customer experiences them as failures. It belongs to the legal system that has given a free pass to software developers on bug related damages. It also belongs to universities and other higher educational institutions that stress on programming over software engineering principles and practices.

The software has a very special characteristic e.g., “it does not wear out”. Its behavior and nature is quite different than other products of human life.

Some of the important characteristics are discussed below:

- **Software does not wear out:**
  There is a well-known “bath tub curve” in reliability studies for hardware products. Important point is that software becomes reliable overtime instead of wearing out. It becomes obsolete, if the environment for which it was developed, changes. Hence software may be retired due to environmental changes, new requirements, new expectations, etc.

- **Software is not manufactured:**
  The life of software is from concept exploration to the retirement of the software product. It is one time development effort and continuous maintenance effort in order to keep it operational. However, making 1000 copies is not an issue and it does not involve any cost. In case of hardware product, every product costs us due to raw material
and other processing expenses. We do not have assembly line in software development. Hence it is not manufactured in the classical sense.

- **Reusability of components:**
  If we have to manufacture a TV, we may purchase picture tube from one vendor, cabinet from another, design card from third and other electronic components from fourth vendor. We will assemble every part and test the product thoroughly to produce a good quality TV. We may be required to manufacture only a few components or no component at all. We purchase every unit and component from the market and produce the finished product. We may have standard quality guidelines and effective processes to produce a good quality product.

- **Software is flexible:**
  We all feel that software is flexible. A program can be developed to do almost anything. Sometimes, this characteristic may be the best and may help us to accommodate any kind of change. However, most of the times, this “almost anything” characteristic has made software development difficult to plan, monitor and control. This unpredictability is the basis of what has been referred to for the past 30 years as the “Software Crisis”.
1.2.2 Software Applications

Software applications are grouped into eight areas for convenience:

1.2.2.1 System software
Infrastructure software comes under this category like compilers, operating systems, editors, drivers, etc. Basically system software is a collection of programs to provide service to other programs.

1.2.2.2 Real time software
This type of software is used to monitor, control and analyze real world events as they occur. An example may be software required for weather forecasting. Such software will gather and process the status of temperature, humidity and other environmental parameters to forecast the weather.

1.2.2.3 Embedded software
This type of software is placed in “Read-Only-Memory (ROM)” of the product and controls the various functions of the product. The product could be an aircraft, automobile, security system, signaling system, control unit of power plants, etc. The embedded software handles hardware components and is also termed as intelligent software.

1.2.2.4 Business software
This is the largest application area. The software designed to process business applications is called business software. Business software could be payroll, file monitoring system,
employee management and account management. It may also be a data warehousing tool which helps us to take decisions based on available data. Management information system, enterprise resource planning (ERP) and such other software are popular examples of business software.

1.2.2.5 Personal computer software
The software used in personal computers is covered in this category. Examples are word processors, computer graphics, multimedia and animating tools, database management, computer games etc. This is a very upcoming area and many big organizations are concentrating their effort here due to large customer base.

1.2.2.6 Artificial intelligence software
Artificial Intelligence software makes use of non-numerical algorithms to solve complex problems that are not amenable to computation or straight forward analysis [7]. Examples are expert systems, artificial neural network, signal processing software etc.

1.2.2.7 Web based software
The software related to web applications comes under this category. Examples are CGI, HTML, Java, Perl, DHTML etc.
1.2.2.8 Engineering and scientific software

Scientific and engineering application software is grouped in this category. Huge computing is normally required to process data. Examples are CAD/CAM package, SPSS, MATLAB, Engineering Pro, Circuit analyzers etc.

1.2.3 Software Process

The software process is the way in which we produce software. This differs from organization to organization. Surviving in the increasingly competitive software business requires more than hiring smart, knowledgeable developers and buying the latest development tools. We also need to use effective software development processes, so that developers can systematically use the best technical and managerial practices to successfully complete their projects. Many software organizations are looking at software process improvement as a way to improve the quality, productivity, predictability of their software development, and maintenance efforts [10]. It seems straight forward, and the literature has a number of success stories of companies that substantially improved their software development and project management capabilities. However, many other organizations do not manage to achieve significant and lasting improvements in the way they conduct their projects. Here are few reasons why is it difficult to improve software process [11, 12].
1.2.3.1 Limitations for Software Process Improvement

- **Lack of time**: Unrealistic schedules leave insufficient time to do the essential project work. No software groups are sitting around with plenty of spare time to devote to exploring what is wrong with their current development processes and what they should be doing differently. Customers and senior managers are demanding more software, of higher quality in minimum possible time. Therefore, there is always a shortage of time. One consequence is that software organizations may deliver release 1.0 on time, but then they have to ship release 1.01 almost immediately thereafter to fix the recently discovered bugs.

- **Lack of knowledge**: A second obstacle to widespread process improvement is that many software developers do not seem to be familiar with industry best practices. Normally, software developers do not spend much time reading the literature to find out about the best known ways of software development. Developers may buy books on Java, Visual Basic or Oracle, but do not look for anything about process, testing or quality on their bookshelves. The industry awareness of process improvement frameworks such as the capability maturity model and ISO 9001 for software have grown in recent years, but effective and sensible
application still is not that common. Many recognized best practices available in literature simply are not in widespread use in the software development world.

- **Wrong motivations:** Some organizations launch process improvement initiatives for the wrong reasons. May be an external entity, such as a contractor, demanded that the development organization should achieve CMM level X by date Y. Or perhaps a senior manager learned just enough about the CMM and directed his organization to climb on the CMM bandwagon. The basic motivation for software process improvement should be to make some of the current difficulties we experience on our projects to go away. Developers are rarely motivated by seemingly arbitrary goals of achieving a higher maturity level or an external certification (ISO 9000) just because someone has decreed it. However, most people should be motivated by the prospect of meeting their commitments, improving customer satisfaction, and delivering excellent products that meet customer expectations. The developers have resisted many process improvement initiatives when they were directed to do “the CMM thing”, without a clear explanation of the reasons why improvement was needed and the benefits the team expected to achieve.
Insufficient commitment: Many times, the software process improvement fails, despite best of intentions, due to lack of true commitment. It starts with a process assessment but fails to follow through with actual changes. Management sets no expectations from the development community around process improvement; they devote insufficient resources, write no improvement plan, develop no roadmap, and pilot no new processes. The investment we make in process improvement will not have an impact on current productivity; because the time we spend developing better ways to work tomorrow is not available for today’s assignment. It can be tempting to abandon the effort when skeptics see the energy they want to be devoted to immediate demands being siphoned off in the hope of a better future. Software organizations should not give up, but should take motivation from the very real, long-term benefits that many companies (including Motorola, Hewlett-Packard, Boeing, Microsoft etc.) have enjoyed from sustained software process improvement initiatives.

1.2.4 Software Quality

Software quality is the extent to which an industry-defined set of desirable features are incorporated into a product so as to enhance its lifetime performance. This definition focuses on the existence of
a product in the first place and that its quality has a time dimension. It also focuses on features which will enhance the product.

To understand the landscape of software quality it is central to answer the so often asked question: what is quality? Many prominent authors and researchers have provided an answer to this question.

### 1.2.4.1 Definitions according to researchers

- **Quality according to Crosby**

  In the book “Quality is free: the art of making quality certain” [13], Philip B. Crosby writes:

  *The first erroneous assumption is that quality means goodness, or luxury or shininess. The word “quality” is often used to signify the relative worth of something in such phrases as “good quality”, “bad quality” and “quality of life” - which means different things to each and every person. As follows quality must be defined as “conformance to requirements” if we are to manage it. Consequently, the nonconformance detected is the absence of quality, quality problems become nonconformance problems, and quality becomes definable.*

  Crosby is a clear “conformance to specification” quality definition adherer. However, he also focuses on trying to understand the full array of expectations that a customer has on quality by expanding the, of today’s measure, somewhat narrow production perspective on
quality with a supplementary external perspective. Crosby also emphasizes that it is important to clearly define quality to be able to measure and manage the concept. Crosby summarizes his perspective on quality in fourteen steps but is built around four fundamental "absolutes" of quality management:

1) Quality is defined as conformance to requirements, not as “goodness” or “elegance”

2) The system for causing quality is prevention, not appraisal. That is, the quality system for suppliers attempting to meet customers' requirements is to do it right the first time. As follows, Crosby is a strong advocate of prevention, not inspection. In a Crosby oriented quality organization everyone has the responsibility for his or her own work. There is no one else to catch errors.

3) The performance standard must be Zero Defects, not "that's close enough". Crosby has advocated the notion that zero errors can and should be a target.

4) The measurement of quality is the cost of quality. Costs of imperfection, if corrected, have an immediate beneficial effect on bottom-line performance as well as on customer relations. To that extent, investments should be made in training and other supporting activities to eliminate errors and recover the costs of waste.
Quality according to Deming

Walter Edwards Deming’s “Out of the crisis: quality, productivity and competitive position” [14], states:

The problem inherent in attempts to define the quality of a product, almost any product, where stated by the master Walter A. Shewhart. The difficulty in defining quality is to translate future needs of the user into measurable characteristics, so that a product can be designed and turned out to give satisfaction at a price that the user will pay. This is not easy, and as soon as one feels fairly successful in the endeavor, he finds that the needs of the consumer have changed, competitors have moved in etc.

One of Deming’s strongest points is that quality must be defined in terms of customer satisfaction – which is a much wider concept than the “conformance to specification” definition of quality (i.e. “meeting customer needs” perspective). Deming means that quality should be defined only in terms of the agent – the judge of quality. Deming’s philosophy of quality stresses that meeting and exceeding the customers' requirements is the task that everyone within an organization needs to accomplish. Furthermore, the management system has to enable everyone to be responsible for the quality of his output to his internal customers. To implement his perspective on quality Deming introduced his 14 Points for Management in
order to help people understand and implement the necessary transformation:

1) **Create constancy of purpose for improvement of product and service:** A better way to make money is to stay in business and provide jobs through innovation, research, constant improvement and maintenance.

2) **Adopt the new philosophy:** For the new economic age, management needs to take leadership for change into a *learning organization*. Furthermore, we need a new belief in which mistakes and negativism are unacceptable.

3) **Cease dependence on mass inspection:** Eliminate the need for mass inspection by building quality into the product.

4) **End awarding business on price:** Instead, aim at minimum total cost and move towards single suppliers.

5) **Improve constantly and forever the system of production and service:** Improvement is not a one-time effort. Management is obligated to continually look for ways to reduce waste and improve quality.

6) **Institute training:** Too often, workers have learned their job from other workers who have never been trained properly. They are forced to follow unintelligible instructions. They can't do their jobs well because no one tells them how to do so.

7) **Institute leadership:** The job of a supervisor is not to tell people what to do nor to punish them, but to
lead. Leading consists of helping people to do a better job and to learn by objective methods.

8) **Drive out fear:** Many employees are afraid to ask questions or to take a position, even when they do not understand what their job is or what is right or wrong. To assure better quality and productivity, it is necessary that people feel secure. "The only stupid question is the one that is not asked."

9) **Break down barriers between departments:** Often a company's departments or units are competing with each other or have goals that conflict. They do not work as a team; therefore they cannot solve or foresee problems. Even worse, one department's goal may cause trouble for another.

10) **Eliminate slogans, exhortations and numerical targets:** These never help anybody do a good job. Let workers formulate their own slogans. Then they will be committed to the contents.

11) **Eliminate numerical quotas or work standards:** Quotas take into account only numbers, not quality or methods. They are usually a guarantee of inefficiency and high cost. A person, in order to hold a job, will try to meet a quota at any cost, including doing damage to his company.

12) **Remove barriers to taking pride in workmanship:** People are eager to do a good job and distressed when they cannot.
13) **Institute a vigorous programme of education:**
Both management and the work force will have to be educated in the new knowledge and understanding, including teamwork and statistical techniques.

14) **Take action to accomplish the transformation:**
It will require a special top management team with a plan of action to carry out the quality mission. A critical mass of people in the company must understand the 14 points.

- **Quality according to Feigenbaum**
  The name Feigenbaum and the term total quality control are virtually synonymous due to his profound influence on the concept of total quality control (but also due to being the originator of the concept). In “Total quality control” [15] Armand Vallin Feigenbaum explains his perspective on quality through the following text:

  *Quality is a customer determination, not an engineer’s determination, not a marketing determination, nor a general management determination. It is based on upon the customer’s actual experience with the product or service ,measured against his or her requirements – stated or unstated, conscious or merely sensed, technically operational or entirely subjective – and always representing a moving target in a competitive market. Product and service quality can be defined as: The total composite product and service*
characteristics of marketing, engineering, manufacture and maintenance though with the product and service in use will meet the expectations of the customer.

Feigenbaum’s definition of quality is unmistakable a “meeting customer needs” definition of quality. In fact, he goes very wide in his quality definition by emphasizing the importance of satisfying the customer in both actual and expected needs. Feigenbaum essentially points out that quality must be defined in terms of customer satisfaction, that quality is multidimensional (it must be comprehensively defined), and as the needs are changing quality is a dynamic concept in constant change as well. It is clear that Feigenbaum’s definition of quality not only encompasses the management of product and services but also of the customer and the customer’s expectations.

- Quality according to Ishikawa

Kaoru Ishikawa writes the following in his book “What is quality control? The Japanese Way” [16]:

>We engage in quality control in order to manufacture products with the quality which can satisfy the requirements of consumers. The mere fact of meeting national standards or specifications is not the answer, it is simply insufficient. International standards established by the International Organization for Standardization (ISO) or the International
Electrotechnical Commission (IEC) are not perfect. They contain many shortcomings. Consumers may not be satisfied with a product which meets these standards. We must also keep in mind that consumer requirements change from year to year and even frequently updated standards cannot keep the pace with consumer requirements. How one interprets the term “quality” is important. Narrowly interpreted, quality means quality of products. Broadly interpreted, quality means quality of product, service, information, processes, people, systems etc. etc.

Ishikawa’s perspective on quality is a “meeting customer needs” definition as he strongly couples the level of quality to every changing customer expectations. He further means that quality is a dynamic concept as the needs, the requirements and the expectations of a customer continuously change. As follows, quality must be defined comprehensively and dynamically. Ishikawa also includes that price as an attribute on quality – that is, an overprized product can neither gain customer satisfaction and as follows not high quality.

- **Quality according to Juran**

  In “Jurans’s Quality Control Handbook” [17] Joseph M. Juran provides two meanings to quality:

  The word quality has multiple meanings. Two of those meanings dominate the use of the word: 1) Quality
consists of those product features which meet the need of customers and thereby provide product satisfaction.

2) Quality consists of freedom from deficiencies. Nevertheless, in a handbook such as this it is most convenient to standardize on a short definition of the word quality as “fitness for use” Juran takes a somewhat different road to defining quality than the other gurus previously mentioned. His point is that we cannot use the word quality in terms of satisfying customer expectations or specifications as it is very hard to achieve this. Instead he defines quality as “fitness for use” – which indicates references to requirements and products characteristics. As follows Juran’s definition could be interpreted as a “conformance to specification” definition more than a “meeting customer needs” definition. Juran proposes three fundamental managerial processes for the task of managing quality. The three elements of the Juran Trilogy are:

- Quality planning: A process that identifies the customers, their requirements, the product and service features that customers expect, and the processes that will deliver those products and services with the correct attributes and then facilitates the transfer of this knowledge to the producing arm of the organization.

- Quality control: A process in which the product is examined and evaluated against the original
requirements expressed by the customer. Problems detected are then corrected.

- Quality improvement: A process in which the sustaining mechanisms are put in place so that quality can be achieved on a continuous basis. This includes allocating resources, assigning people to pursue quality projects, training those involved in pursuing projects, and in general establishing a permanent structure to pursue quality and maintain the gains secured.

**Quality according to Shewhart**

As referred to by W.E. Deming, “the master”, Walter A. Shewhart defines quality in “Economic control of quality of manufactured product” [18] as follows:

*There are two common aspects of quality: One of them has to do with the consideration of the quality of a thing as an objective reality independent of the existence of man. The other has to do with what we think, feel or sense as a result of the objective reality. In other word, there is a subjective side of quality.*

Although Shewhart’s definition of quality is from 1920s, it is still considered by many to be the best and most superior. Shewhart talks about both an objective and subjective side of quality which nicely fits into both “conformance to specification” and “meeting customer needs” definitions.
Some technical authors like to describe software quality in terms of "Fitness for purpose" but more recent commercial thinking would not fully support this description.

1.2.4.2 Definitions by International organizations

Quality is defined by International organizations as follows:

- **Definition by German Industry Standard DIN 55350 Part 11**
  "Quality comprises all characteristics and significant features of a product or an activity which relate to the satisfying of given requirements".

  "Quality is the totality of features and characteristics of a product or a service that bears on its ability to satisfy the given needs".

- **Definition by IEEE Standard (IEEE Std 729-1983)**
  "a) The totality of features and characteristics of a software product that bear on its ability to satisfy given needs: for example, conform to specifications.
  b) The degree to which software possesses a desired combination of attributes.
  c) The degree to which a customer or user perceives that software meets his or her composite expectations."
d) The composite characteristics of software that determine the degree to which the software in use will meet the expectations of the customer".

These standards are a long time in existence and their relevance to the late '90s might be a little too broad. The IEEE standard specifically relates to software, so, it's a good candidate for closer analysis. This standard defines quality in terms of features and characteristics.

1.2.5 Software Quality Models

1.2.5.1 McCall Model
One of the more renowned predecessors of today’s quality models is the quality model presented by Jim McCall [19-21](also known as the General Electrics Model of 1977). This model, as well as other contemporary models, originates from the US military and is primarily aimed towards the system developers and the system development process. McCall’s quality model is one of the first model of its kind. Its prior focus lies on developers and the development process. By choosing software quality factors, that reflect the user’s and the developer’s point of view, McCall tries to close the gap between these two stakeholders.

The McCall quality model has three major perspectives for defining and identifying the quality of a software product: product revision (ability to undergo changes), product
transition (adaptability to new environments) and product operations (its operation characteristics).

Product revision includes maintainability (the effort required to locate and fix a fault in the program within its operating environment), flexibility (the ease of making changes required by changes in the operating environment) and testability (the ease of testing the program, to ensure that it is error-free and meets its specification).

Product transition is all about portability (the effort required to transfer a program from one environment to another), reusability (the ease of reusing software in a different context) and interoperability (the effort required to couple the system to another system). Quality of product operations depends on correctness (the extent to which a program fulfils its specification), reliability (the system's ability not to fail), efficiency (further categorized into execution efficiency and storage efficiency and generally meaning the use of resources, e.g. processor time, storage), integrity (the protection of the program from unauthorized access) and usability (the ease of the software).
In more details, McCall’s Quality Model consists of 11 quality factors to describe the external view of the software (from the users’ view), 23 quality criteria to describe the internal view of the software (from the developer’s view) and a set of Metrics which are defined and used to provide a scale and method for measurement. The idea behind McCall’s Quality Model is that the quality factors synthesized should provide a complete software quality picture [21]. The actual
quality metric is achieved by answering yes and no questions that then are put in relation to each other. That is, if answering equally amount of “yes” and “no” on the questions measuring a quality criteria you will achieve 50% on that quality criteria. The metrics can then be synthesized per quality criteria, per quality factor, or if relevant per product or service.

1.2.5.2 Boehm’s Quality Model

The second of the basic and founding predecessors of today’s quality models is the quality model presented by Barry W. Boehm [22;23]. Boehm addresses the contemporary shortcomings of models that automatically and quantitatively evaluate the quality of software. In essence his models attempts to qualitatively define software quality by a given set of attributes and metrics. Boehm's model is similar to the McCall Quality Model in that it also presents a hierarchical quality model structured around high-level characteristics, intermediate level characteristics, primitive characteristics - each of which contributes to the overall quality level.

The high-level characteristics represent basic high-level requirements of actual use to which evaluation of software quality could be put – the general utility of software. The high-level characteristics address three main questions that a buyer of software has:

- As-is utility: How well (easily, reliably, efficiently) can I use it as-is?
• Maintainability: How easy is it to understand, modify and retest?

• Portability: Can I still use it if I change my environment?

The contents of the Boehm’s quality model has three levels, high-level, intermediate-level and lowest-level characteristics. In addition, it is noted that there is a number of the lowest-level characteristics which can be related to more than one intermediate-level characteristics, for example, the ‘Self Contentedness’ primitive characteristic could be related to the ‘reliability’ and ‘portability’ primitive characteristics. In the intermediate level characteristic, there are seven quality characteristics that together represent the qualities expected from a software system [Boehm et al, 1976, Boehm et al, 1978]:

1. Portability: the software can be operated easily and well on computer configurations other than its current one.

2. Reliability: the software can be expected to perform its intended functions satisfactorily.

3. Efficiency: the software fulfills its purpose without waste of resources.

4. Usability: the software is reliable, efficient and human-engineered.

5. Testability: the software facilitates the establishment of verification criteria and supports evaluation of its performance.

6. Understandability: the software purpose is clear to the inspector.
7. Flexibility: the software facilitates the incorporation of changes, once the nature of the desired change has been determined.

The primitive characteristics can be used to provide the foundation for defining quality metrics, this use is one of the most important goals established by Boehm when he constructed his quality model. One or more metrics are supposed to measure a given primitive characteristic. Boehm [1978] defined the ‘metric’ as “a measure of extent or degree to which a product possesses and exhibits a certain (quality) characteristic.”
1.2.5.3 FURPS/FURPS+ Model

A later, and perhaps somewhat less renown, model that is structured in basically the same manner as the previous two quality models (but still worth at least to be mentioned in this context) is the FURPS model originally presented by Robert Grady [24] (and extended by Rational Software [25-27] - now IBM Rational Software – into FURPS+3). FURPS stands for:

- Functionality – which may include feature sets, capabilities and security
- Usability - which may include human factors, aesthetics, consistency in the user interface, online and context sensitive help, wizards and agents, user documentation, and training materials
- Reliability - which may include frequency and severity of failure, recoverability, predictability, accuracy, and mean time between failure (MTBF)
- Performance - imposes conditions on functional requirements such as speed, efficiency, availability, accuracy, throughput, response time, recovery time, and resource usage
- Supportability - which may include testability, extensibility, adaptability, maintainability, compatibility, configurability, serviceability, installability, localizability (internationalization)

The FURPS-categories are of two different types: Functional (F) and Non-functional (URPS). These categories can be used as both product requirements as well as in the assessment of product quality.
1.2.5.4 *Dromey's Quality Model*

An even more recent model similar to the McCall’s, Boehm’s and the FURPS(+) quality model, is the quality model presented by R. Geoff Dromey [28;29]. Dromey proposes a product based quality model that recognizes that quality evaluation differs for each product and that a more dynamic idea for modeling the process is needed to be wide enough to apply for different systems. Dromey is focusing on the relationship between the quality attributes and the sub-attributes, as well as attempting to connect software product properties with software quality attributes.

There are three principal elements to Dromey's generic quality model:

1) Product properties that influence quality
2) High level quality attributes
3) Means of linking the product properties with the quality attributes.
Dromey's Quality Model is further structured around a 5 step process:

1) Chose a set of high-level quality attributes necessary for the evaluation.
2) List components/modules in your system.
3) Identify quality-carrying properties for the components/modules (qualities of the component that have the most impact on the product properties from the list above).
4) Determine how each property effects the quality attributes.
5) Evaluate the model and identify weaknesses.

1.2.5.5 ISO 9126 Quality Model

In 1991, the ISO published its first international consensus on the terminology for the quality characteristics for software product evaluation; this standard was called as Software Product Evaluation - Quality Characteristics and Guidelines for Their Use (ISO 9126) [30]. From 2001 to 2004, the ISO published an expanded version, containing both the ISO quality models and inventories of proposed measures for these models.

The current version of the ISO 9126 series now consists of one International Standard (IS) and three Technical Reports (TRs):


1. Internal and external quality model.
2. Quality in use model.

The first part of the two-parts quality model determines six characteristics in which they are subdivided into twenty-seven sub-characteristics for internal and external quality, as in Figure 6 [ISO, 2001]. These sub-characteristics are a result of internal software attributes and are noticeable externally when the software is used as a part of a computer system.

The second part of the two-part model indicates four quality in use characteristics.
Figure 1.4: ISO 9126 Model

Figure 1.4 shows the ISO view of the expected relationships between internal, external, and quality in use attributes. The internal quality attributes influence on the external quality attributes while the external attributes influence on the quality in use attributes. Furthermore, the quality in use depends on the external quality while the external quality depends on the internal quality [ISO, 2001]. For the internal and external software products, each quality characteristics and its corresponding sub-characteristics are defined in ISO 9126-1 [ISO, 2001] as follows:
1. Functionality: “the capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions”. It contains the following sub-characteristics:

a. Suitability: “the capability of the software product to provide an appropriate set of functions for specified tasks and user objectives”.

b. Accuracy: “the capability of the software product to provide the right or agreed results or effects with the needed degree of precision”.

c. Security: “the capability of the software product to protect information and data so that unauthorised persons or systems cannot read or modify them and authorised persons or systems are not denied access to them”.

d. Interoperability: “the capability of the software product to interact with one or more specified systems”.

e. Functionality Compliance: “the capability of the software product to adhere to standards, conventions or regulations in laws and similar prescriptions relating to functionality”.

2. Reliability: “The capability of the software product to maintain a specified level of performance when used under specified conditions”. It includes the following subcharacteristics:
a. Maturity: “the capability of the software product to avoid failure as a result of faults in the software”.
b. Fault tolerance: “the capability of the software product to maintain a specified level of performance in cases of software faults or of infringement of its specified interface”.
c. Recoverability: “the capability of the software product to re-establish a specified level of performance and recover the data directly affected in the case of a failure”.
d. Reliability Compliance: “the capability of the software product to adhere to standards, conventions or regulations relating to reliability”.

3. Usability: “the capability of the software product to be understood, learned, used, and attractive to the user, when used under specified conditions”.

It contains the following sub-characteristics:
a. Understandability: “the capability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use”.
b. Learnability: “the capability of the software product to enable the user to learn its application”.
c. Operability: “the capability of the software product to enable the user to operate and control it”.
d. Attractiveness: “the capability of the software product to be attractive to the user”.
e. Usability Compliance: “the capability of the software product to adhere to standards, conventions, style guides or regulations relating to usability”.

4. Efficiency: “the capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions”. It includes the following subcharacteristics:
   a. Time behavior: “the capability of the software product to provide appropriate response and processing times and throughput rates when performing its function, under stated conditions”.
   b. Resource behavior: “the capability of the software product to use appropriate amounts and types of resources when the software performs its function under stated conditions”.
   c. Efficiency Compliance: “the capability of the software product to adhere to standards or conventions relating to efficiency”.

5. Maintainability: “the capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications”. It contains the following sub-characteristics:
   a. Analyzability: “the capability of the software product to be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified to be identified”.
b. Changeability: “the capability of the software product to enable a specified modification to be implemented”.

c. Stability: “the capability of the software product to avoid unexpected effects from modifications of the software”.

d. Testability: “the capability of the software product to enable modified software to be validated”.

e. Maintainability Compliance: “the capability of the software product to adhere to standards or conventions relating to maintainability”.

6. Portability: “the capability of the software product to be transferred from one environment to another”. It includes the following sub characteristics:

a. Adaptability: “the capability of the software product to be adapted for different specified environments without applying actions or means other than those provided for this purpose for the software considered”.

b. Installability: “the capability of the software product to be installed in a specified environment”.

c. Co-existence: “the capability of the software product to co-exist with other independent software in a common environment sharing common resources”.

d. Replaceability: “the capability of the software product to be used in place of another specified software product for the same purpose in the same environment”.
e. Portability Compliance: “the capability of the software product to adhere to standards or conventions relating to portability”.

1.3 Motivation and Objective of Research
A quality model is a schema to better explain our view of quality. Some existing quality models can predict fault-proneness with reasonable accuracy in certain contexts. Other quality models attempt at evaluating several quality characteristics but fail at providing reasonable accuracy, from lack of data mainly. We believe that 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 effort. Such quality models can also help developers in building better quality programs by exposing the relationships between internal attributes and external quality characteristics clearly.

When specifying quality requirements, a way is needed to organize, clarify, and standardize the relevant meanings of the term “quality” when applied to software-intensive systems. If requirement engineers do this first, they will form a proper foundation for identifying, analyzing, and specifying the large number of quality requirements that are needed on any significant endeavor. This then is one role of a quality model, a concept that comes from the quality and measurement communities: to make the general term “quality” specific and useful when engineering requirements. A quality model first decomposes the general concept of quality to create taxonomy of its component quality factors and sub factors (i.e., aspects, attributes, or characteristics). The quality model then
provides specific quality criteria (i.e., descriptions) and measures (i.e., means of measurement) that can be used to turn these general high-level quality factors into detailed and specific measurable descriptions that can be used to specify the associated aspect of quality or to determine during testing if that aspect of quality actually exists. By mandating minimum levels of quality measures for quality criteria for relevant quality factors and sub factors, requirements engineers can obtain unambiguous and testable quality requirements.

**Objective:**
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 an efficient system. The objective of this thesis will be:

- To identify quality parameters of the various development phases involved.
- To improve the quality of the output of all the involved phases.
- To develop a quality model

### 1.4 Scope of Research

The typical scope of a quality model is one or more related applications. Thus, a quality model is used to document or analyze the required or actual quality of an application. Another common scope for a quality
model is one or more related components, whether these components are being bought or developed, either as part of a larger application or as generally reusable components.

Finally, one can use a quality model to document or analyze the planned or actual quality of a business unit (e.g., specifying the required interoperability, performance, and security of a business organization). Thus, just as quality requirements can be engineered for an application, component, center, or business, quality models can also be developed for each of these four scopes. A quality model can also have another kind of scope. A quality model can be:

• A general industry standard quality model, such as the ISO 9126 quality model.

• An organizational quality model, such as one that is used on all projects within a given development organization (e.g., all applications within a program or related projects or a product line of applications).

• An endeavor-specific quality model, such as one that is used on a single project developing a single application.
1.5 References


