Chapter 3

Background

For software decisions, the most critical and difficult task is to estimate the cost of a proposed project. Software cost estimation is required to define the resources needed to produce, verify, and validate the software product in demand.

This chapter will give an overview of different software cost estimation methods and techniques that exist. It will mention strengths and weaknesses of each approach. Then two main cost estimation techniques that are analyzed in more detail are COCOMO II and expert estimation and these are the main focus areas of my research. The chapter will also talk about the different tools associated with these techniques.

This will be followed by another important segment of this research i.e. software quality. In this, we will talk about the different quality models existing in the market and their attributes.

The chapter will close with analysis of different types of costs related to quality.

3.1 Estimation Techniques – At A Glance

Many cost estimation techniques have evolved in the last 30 years, with most of them forming effort as the base for estimation the cost of developing software. Although effort and cost are closely related, they may not necessarily correlate with each other by a simple transformation function. Effort measured in person-months of the programmers, analysts and project managers is converted into a dollar cost figure by calculating an average salary per unit time of the staff involved, and then multiplying this by the estimated effort required [17].

Following are some popular techniques:
Majorly software cost estimation techniques are categorized into Algorithmic and Non-algorithmic also known as parametric and non-parametric.

1. **Algorithmic techniques** are those where some mathematical calculations are done by using formulae like

   \[ \text{Effort} = f(x_1, x_2, \ldots, x_n) \]

   …where \( x_1, x_2, \ldots, x_n \) denote the cost factors i.e. software metrics.

   The existing algorithmic methods differ in two aspects: the selection of cost factors, and the form of the function \( f \).

Some of the existing algorithmic models are COCOMO 1[18], COCOMO II [18] and Putnam popularly known as SLIM [18] etc. These techniques include more structured or better formulated estimations as their equations have been constructed by application of various numerical analysis techniques on the data from present and past software projects.

a) **Cost Constructive Model (COCOMO):** This family model was proposed by Barry Boehm. This method uses some equations and parameters that have been derived from previous experiences about software projects for estimation. In the COCOMOs, the code-size \( S \) is given in thousand LOC (KLOC) and Effort is in person-months. There are three modules of COCOMO as given by Barry Boehm

   a. Basic COCOMO, which is very simple and easy to use,
   b. Intermediate COCOMO, that uses an Effort Adjustment factor (EAF) along with size of the software in order to estimate the cost, and
   c. Detailed COCOMO, which works on each sub-system separately and has an obvious advantage for large systems.

   Effort adjustment factors include database size, product complexity, programmer capability, application experience, language and tools experience etc.

**COCOMO II** is the revised version of COCOMO I. This model is specially designed for the 21st century software projects. In this, the numbers of effort adjustment factors are also increased by 5.
b) **Putnam’s Model and SLIM:** Putnam’s model was proposed in accordance with manpower distribution and examination of many software projects. SLIM is based on Putnam’s analysis of the life-cycle in terms of a so-called Rayleigh distribution of project personnel level versus time. In SLIM, software development characteristics of size and technology factors are linked to the basic Rayleigh manpower distribution model using productivity. Recently, Quantitative Software Management has developed a set of three tools based on Putnam’s SLIM

a. SLIM-Estimate, which is a project planning tool

b. SLIM-Control, which is used for project tracking and oversight, and

c. SLIM Metrics, which is a software metrics repository and benchmarking tool.

2. **Non-Algorithmic Estimation:** On the other hand **non-algorithmic** is one where cost is estimated based on previous experience, historical data etc. Here no mathematical calculation or formulae are used in order to derive cost. These methods require one or more completed projects and derive estimation through reasoning by analogy using the actual costs of previous projects. Some of these methods are as follows:

a) **Expertise Based Techniques:** This is a very popular method in the industry. It is used in cases where gathering information is a challenge. This method captures knowledge and experience of experts, providing estimates based on analysis of known outcomes of all the past projects in which the experts have participated. The only disadvantage of this method is that the estimate derived does not have any numerical / established data in the background. It is purely based on what the experts felt and mentioned. Examples of expert based techniques include: Delphi technique and rule based systems [18].

b) **Estimation by Analogy:** Estimation by analogy can be performed at any level/stage in the project. Performing this activity at an overall project level will include all cost components of the system while doing it at a sub-system level will include granularities and be relatively accurate [6].
c) **Top-Down Estimating Method**: This method is also called the Macro Model. This model is used to derive the overall cost estimation of the project based on its global properties. The project is then partitioned into low-level components. Using this method, the cost can be estimated at the very early stages of the project (when there is minimal information available) as the model is only dependent on global properties of the project. This ‘minimum detail requirement’ property of the model makes it faster and easier to implement [19].

d) **Bottom-up Estimating Method**: This method works in the reverse order as compared to the top-down one. The cost of each software component is estimated first and is rolled up to calculate the total cost of the project. The estimation may not be accurate as ground level information may not be available in the early stages of the project. It is more time consuming as compared to Top-Down estimation.[19]

e) **Parkinson**: In this method, software cost is estimated based on resources available so, the project costs whatever resources are available. Even though this “method” might be accurate in some cases, but definitely it is not advisable because it is not economical. The estimates derived from this method are problematic and leads to uneconomic investments. The budget and schedule of the software is not guaranteed and even the system is usually unfinished.

f) **Price-to-win**: In this method, software cost is estimated based on whatever the customer has to spend on it. It is not the honest estimation. Instead, the price is chosen in a way that it allows calls for bids to be won. This method is highly criticized, but very commonly used in industry and one of the major reasons for projects to run out of budget and fail. The probability that the customer gets the system he or she wants is low. Costs do not accurately reflect the work required.
g) **Machine learning algorithms**: It is a new area that demonstrates the promise of producing consistently accurate estimates. The system effectively "learns" how to estimate from training set of completed projects. This technique categorizes into methods like Neural Networks, Fuzzy Logic, Case-Based Reasoning, Analogy, Rule-Based, Regression Trees and Hybrid System[20]

The above said software cost estimation techniques have one common goal that is to determine the accurate estimate for size, effort and schedule. In next section we will understand more about estimate accuracy and the relationship between uncertainty and accuracy.

### 3.2 Understanding an Estimate’s Accuracy

Earlier, software development was less complicated than today. As technology is emerging day by day, new devices and techniques are hitting the market. This is making the software development process more complicated. Software cost estimation was not very accurate earlier owing to too many uncertainties. When we talk of accuracy, it is an indication of how close something is to reality and precision is an indication of how finely something is measured. In software development, we start with estimating the size of the project. If size estimation is in the accuracy range, we can safely assume that all values calculated from this (like cost, effort, schedule, etc.) will also fall in range.

In the lifetime of a project, you make several estimates. The range should narrow down every time and the estimate should approach what will eventually be the actual cost values for the product or system being developed [21]. This process of estimation is represented by the cone of uncertainty as shown below [22]:

---

20
Apart from uncertainties like unstable requirements, new concepts, new technology, changes in product design and using some COTS components, some other environmental uncertainties like competition, technology evolution, changes in organizations and mission priorities also impact the cost estimation. These factors can cause the best of estimates to crumple and may lead to an increase in the estimated cost.

The shape of the uncertainty cone depends on the usage of different cost estimation methods that integrate risks and available knowledge into new project estimates. The uncertainty cone is not always symmetrical. The reason for this is that, according to Parkinson’s Law, “work expands so as to fill the time available for its completion” [23].

Now, in next section I will be discussing two main cost estimation techniques on which my research is based in detail i.e. **Formal Estimation and Expert Estimation** and the supporting tools available for them.
Formal Estimation Method

There are various formal estimation methods available in the industry but the commonly used methods includes the non-commercial ConstructiveCostModel (COCOMO) and the three commercial models Putnam’s SLIM, Price System’s PRICE-S and SEER-SEM by Galborath. In each of these models, the actual estimation step is based on a series of formulae. This is what makes them similar [53].

COCOMO
COCOMO I is a holistic model, developed by Barry Boehm in 1981. This model is based on 63 waterfall models based on projects at the TRW Aerospace. The study is based on projects ranging from size from 2000 to 100000 lines of code and programming languages ranging from assembly to PL/I. COCOMO comprises of a series of three models:

- **Basic**: This model is used for rough and quick estimation by using size as an important factor. The cost estimated using this method lacks in accuracy as it depends on only one factor but in real or current scenario there are many factors that influences the software cost.
- **Intermediate**: In this model, 15 costs drivers were considered along with size in order to estimate software cost.
- **Detailed**: This model is used to calculate cost/effort required for each phase of software development.

Boehm considered the size of the system as the key factor that influences the effort. Other factors that influence the effort are project, product, platform and personnel. The main output of the COCOMO model is the estimated effort in person months. The main formula of the COCOMO models is:

Basic COCOMO Equation--- \[ E = a (\text{size})^b \]

Intermediate COCOMO Equation--- \[ E = a (\text{size})^b \times \text{EAF} \]
In the above formula, ‘a’ and ‘b’ are the constants depends on the development mode. This formula was further used to calculate the development time, productivity and average staff size. There were three development modes available i.e. organic, semi-detached and embedded. The following table gives the brief description of different modes [25]:

<table>
<thead>
<tr>
<th>Development mode</th>
<th>Project size (KLOC)</th>
<th>Nature of project</th>
<th>Development environment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organic</td>
<td>Between 2 and 50</td>
<td>Small size project, experienced Developers. For example, pay roll, inventory projects etc.</td>
<td>Familiar &amp; In house</td>
</tr>
<tr>
<td>Semi-detached</td>
<td>Between 50 and 300</td>
<td>Medium size project, Medium size team, Average previous experience on similar project. For example: Utility systems like compilers, database systems, editors etc.</td>
<td>Medium</td>
</tr>
<tr>
<td>Embedded</td>
<td>More than 300</td>
<td>Large project, Real time systems, Complex interfaces, Very little previous experience. For example: ATMs, Air Traffic Control etc.</td>
<td>Complex Hardware/ Customer Interfaces required</td>
</tr>
</tbody>
</table>

Table 3.1: Modes of COCOMO I
This version of COCOMO is too simple to accommodate the ever changing requirement and technology demand. Therefore, COCOMO II was developed in the 1990’s and was calibrated using a dataset of 161 projects. This model handles additional information in the later stages of a project and considers the cost of reusing software components and the effects of various factors on the diseconomies of scale (such as the Development flexibility, team cohesion, risk resolution and the maturity of the development process)[27].

COCOMO II consists of three sub models i.e. Applications Composition, Early Design and Post-architecture models. Application composition model are used for the prototyping stage of application generators, infrastructure & system integration along with application composition type of projects. Early design models are used when less is known about the project. Post-architecture models are used after the completion of the detailed architecture of the project.

The base equation of this model is:

$$\text{PM (nominal)} = A \times \text{size}^B$$

In the above equation, A is constant i.e. 2.5 and the B is the aggregate of five scale factors that account for the economies and diseconomies of scale encountered in software projects. Each scale factor has a range of rating levels from Very Low to Extra High. The scale factors are precedentness, development flexibility, team cohesion, architecture /risk resolution and process maturity. The value of B is calculated as:

$$B = 0.91 + 0.01 \times (\text{sum of rating on scaling factors})$$

In COCOMO II, there are 17 cost drivers that fall into same four categories as COCOMO I i.e. platform, project, personal and product. These cost drivers are [26]:
<table>
<thead>
<tr>
<th>Cost drivers</th>
<th>Categories</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RELY</td>
<td>Product</td>
<td>Required system reliability</td>
</tr>
<tr>
<td>CPLX</td>
<td>Product</td>
<td>Complexity of system modules</td>
</tr>
<tr>
<td>DOCU</td>
<td>Product</td>
<td>Extent of documentation required</td>
</tr>
<tr>
<td>DATA</td>
<td>Product</td>
<td>Size of database used</td>
</tr>
<tr>
<td>RUSE</td>
<td>Product</td>
<td>Required percentage of reusable components</td>
</tr>
<tr>
<td>TIME</td>
<td>Platform</td>
<td>Execution time constraint</td>
</tr>
<tr>
<td>PVOL</td>
<td>Platform</td>
<td>Volatility of development platform</td>
</tr>
<tr>
<td>STOR</td>
<td>Platform</td>
<td>Memory constraints</td>
</tr>
<tr>
<td>ACAP</td>
<td>Personnel</td>
<td>Capability of project analysts</td>
</tr>
<tr>
<td>PCON</td>
<td>Personnel</td>
<td>Personnel continuity</td>
</tr>
<tr>
<td>PCAP</td>
<td>Personnel</td>
<td>Programmer capability</td>
</tr>
<tr>
<td>PEXP</td>
<td>Personnel</td>
<td>Programmer experience in project domain</td>
</tr>
<tr>
<td>AEXP</td>
<td>Personnel</td>
<td>Analyst experience in project domain</td>
</tr>
<tr>
<td>LTEX</td>
<td>Personnel</td>
<td>Language and tool experience</td>
</tr>
<tr>
<td>TOOL</td>
<td>Project</td>
<td>Use of software tools</td>
</tr>
<tr>
<td>SCED</td>
<td>Project</td>
<td>Development schedule compression</td>
</tr>
<tr>
<td>SITE</td>
<td>Project</td>
<td>Extent of multisite working and quality of inter-site communications</td>
</tr>
</tbody>
</table>

Table 3.2: Cost drivers of COCOMO II

After identifying the nominal person month’s value it is being adjusted by considering the value of cost drivers based on how much information developer has.

In case of early estimation model, only 7 cost drivers are considered as less is known at that point of time. So the formula for early estimation is:

\[
\text{PM adjusted} = \text{PM nominal} \times \text{EAF}
\]
In the above formula, EAF is the product of seven early cost drivers wherein each cost driver has weighting factors for every rating level. Following table shows the early cost drivers with their respective weighting factors.

<table>
<thead>
<tr>
<th>Early design Cost drivers</th>
<th>Extra Low</th>
<th>Very Low</th>
<th>Low</th>
<th>Nominal</th>
<th>High</th>
<th>Very High</th>
<th>Extra High</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCPX</td>
<td>.73</td>
<td>.81</td>
<td>.98</td>
<td>1.0</td>
<td>1.30</td>
<td>1.74</td>
<td>2.38</td>
</tr>
<tr>
<td>RUSE</td>
<td>-</td>
<td>-</td>
<td>.95</td>
<td>1.0</td>
<td>1.07</td>
<td>1.15</td>
<td>1.24</td>
</tr>
<tr>
<td>PDIF</td>
<td>-</td>
<td>-</td>
<td>0.87</td>
<td>1.0</td>
<td>1.29</td>
<td>1.81</td>
<td>2.61</td>
</tr>
<tr>
<td>PERS</td>
<td>2.12</td>
<td>1.62</td>
<td>1.26</td>
<td>1.0</td>
<td>0.83</td>
<td>0.63</td>
<td>0.50</td>
</tr>
<tr>
<td>PREX</td>
<td>1.59</td>
<td>1.33</td>
<td>1.12</td>
<td>1.0</td>
<td>0.87</td>
<td>0.71</td>
<td>0.62</td>
</tr>
<tr>
<td>FCIL</td>
<td>1.43</td>
<td>1.30</td>
<td>1.10</td>
<td>1.0</td>
<td>0.87</td>
<td>0.78</td>
<td>0.62</td>
</tr>
<tr>
<td>SCED</td>
<td>-</td>
<td>1.43</td>
<td>1.14</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3.3: Early Cost Drivers with its weighting factors

In post architecture estimation model, there are 17 cost drivers for equation as it is calculated when more is known to the developer about the project. So the equation is

\[ PM_{\text{adjusted}} = PM_{\text{nominal}} \times EAF \]

Here, EAF is the product of 17 cost drivers instead of seven in case of early cost estimation model.

In addition to the above equation/formula to estimate the effort in person months, there are other equations that exist in order to determine the staffing required per phase, the duration and the cost.

**Function Point - An alternate for COCOMO**

In the above discussed COCOMO equations, size was used in KDSI. There is an alternate way to replace the size metric to function point. Using size in KDSI is not an appropriate technique and even impossible to measure for complicated and large systems.
It is directly dependent on the programming language, so not reliable to use. The alternate size metric available for use is function point [24]. This metric was introduced by Allan Albrecht in 1979 and was published in measuring application development productivity [28]. This technique is totally independent of the language tools and it is said that “A function point is a rough estimate of a unit of delivered functionality of a software project” [5]. Function point metric considers five factors for estimation i.e. user input, user output, enquiries, interface logical files and external interface file. Each factor is rated as simple, average or complex and has specific weighting factors.

Since different programming languages need different amount of lines for same project, it is advisable to use the function point method. One can easily convert function point value to LOC using backfiring tables and the conversion value exists for most of the programming languages. For example one Function Point equals 50 SLOC in C++.

The COCOMO model is not a standalone model, but can be used in combination with other models. There are different models available for different purposes. Following is the list of different models with brief description of each:

- COCOTS [29]: The Constructive Commercial-Of-The-Shelf Cost Model is one of the most significant changes in software development practice over the past twenty years. It addresses the challenges in estimating software systems with a high reusability of COTS components in order to keep overall development and maintenance costs as low as possible.

- COQUALMO [30]: COQUALMO (ConstructiveQualityModel) formerly called CODEFMO--is an estimation model used for predicting number of residual defects/KSLOC or defects/FP (Function Point) in a software product. It enables 'what-if' analyses to demonstrate the impact of various defect removal techniques and the effects of personnel, project, product and platform characteristics on software quality.
- **COSYSMO [31]**: The COSYSMO i.e. Constructive Systems Engineering Cost Model is used to estimate the Systems Engineering effort for large scale software and hardware systems.

- **COSECIMO [32]**: The ConstructiveSecurity Cost Model is used for costing secure software–intensive systems.

**COCOMO TOOLS**

As COCOMO is very popular and easily available, many commercial, non-commercial and academic applications are available on it. These are as follows:

- Costar 7.0
- True Planning – Price Systems
- Cocomo II Application for Software Cost Estimation (C.A.S.E)
- Software Project Cost Estimates Using COCOMO II Model
- The USC online COCOMO application

**Expert based Estimation**

This technique is based on Expert judgment and is done by getting advices from experts who have wide experience in similar projects. In contrast to algorithmic or formal estimation models, where the estimation is made based on a mathematical formula, expert based estimation depends on a human estimator for the estimation. Algorithmic models are simpler to analyze as compared to expert estimation as it totally relies on human behavior. In some cases, this approach proves to be better where historical data is not present and project is of high technical uncertainty. Expert estimation can be made using different approaches, like, like top- down, bottom –up, Delphi technique and Wideband Delphi.
In my thesis, Wideband Delphi technique has been used. In expert based estimation, different estimators produce different estimates and there is a possibility of bias. It could be possible that some experts are optimistic or some are pessimistic for the project. Some might have a different understanding altogether for the project which influences the overall project duration. Estimated costs from different experts are then aggregated using mean, median.

The main disadvantage with expert estimation is that the experts might be influenced or biased. So, as the solution of this problem, Wideband Delphi Technique comes for rescue. This is a group consensus technique wherein each expert discusses the estimation issues.

**Wideband Delphi technique**

This technique was introduced at Rand Corporation and later on refined by Barry Boehm [5]. The experts in the group interact one-to-one and discuss the rationale behind their respective estimates. This technique can help in estimating cost, planning and scheduling projects. In case any conflicts arise those are resolved face to face till a mutually agreed decision point is reached. This however leads to involvement of a lot of overheads like time, team involvement and planning for relatively smaller sets of tasks. However its strength lies in iterative, team based and collaborative meetings. The following table shows the basic step of wideband Delphi [5] [33]:

<table>
<thead>
<tr>
<th><strong>Wideband Delphi Technique</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1</strong> Planning</td>
</tr>
<tr>
<td><strong>2</strong> Kickoff meeting</td>
</tr>
<tr>
<td><strong>3</strong> Individual</td>
</tr>
</tbody>
</table>
4 Estimation meeting  Coordinator prepares and distributes a summary of the estimates on an iteration form.

5 Assembling tasks  Coordinator calls a group meeting, specifically focusing on having the experts discuss points where their estimates varied widely.

6 Reviewing results & iteration  Experts fill out forms, again anonymously, and Steps 4 to 6 are iterated for as many rounds as appropriate.

Table 3.4: The Wideband Delphi technique [5] [33]

Now, as compared with Delphi expert estimation it can be easily understood by the following figure, which depicts how expert’s opinion changes with the growth of each round.

![Figure 3.2: Convergence of estimates using Wideband Delphi technique [49]](image)

In the above figure, the difference between the estimates taken from different experts is getting reduced with every round. The main reason is the technique i.e. Wideband Delphi
that helps the experts to reduce the variations by discussing the areas of concern with each other. During this process, estimators first try to focus on tasks that have large variations between estimated values. The moderator of the meeting asks the experts to address those points first that have more discrepancies and in this way they proceed with tasks and try to reduce the discrepancies as much as possible. The moderator finally compiles all the documents with results, assumptions and feedback and makes a final report.

3.3 Software Quality

Quality is a very important concept. The main objective of software engineering is to produce quality software within time and budget. Different people have different perceptions about quality. For some, when software meets all client requirements; it is quality, for others, when it is fit for the purpose for which it is develop; it is a quality and so on. In this section we will discuss product quality, process quality and the major attributes of software quality along with some of its models.

3.3.1 Product Quality

Product quality means the ability of the product to fulfill the client’s needs and expectations. In this context, Garvin [50] specified five different approaches for defining product quality:

- **The Transcendent Approach**: In this, quality is seen as an unanalyzable property which we learn to recognize only through experience.
- **The Product-based Approach**: In this approach, quality is taken as a precise and quantifiable variable. Here, the quality of a product reflects the quantity of attributes a product contains. Quality is observed as part of the product.
- **The User-based Approach**: According to this approach, “quality lies in the eye of the beholder” [50]. It is not something which can be measured or analyzed. Quality is determined by one user, and is perceived differently by different users.
• **The Manufacturing-based Approach**: From this point of view, “*quality is the conformance to requirements*”.

• **The Value-based Approach**: Here, quality is defined in terms of cost and prices. According to this approach, a high quality product is a product that delivers a certain level of quality at an acceptable price.

Apart from these above mentioned approaches, Garvin[50] also proposed quality framework wherein, he described quality in 8 dimensions. These dimensions are as follows:

• **Performance**: It is a measurable attribute of software. It mainly involves the product and user-based approaches.

• **Features**: It is an attribute that differentiates one product from another.

• **Reliability**: “*Reliability reflects the probability of a product’s failing within a specified period of time*” [50]. It is a measurable attribute of quality and it is measured using MTFF (i.e. Mean time to first failure) and MTBF (i.e. Mean time between failures).

• **Conformance**: Conformance is achieved by comparing the actual product features to the specified features of product. A product (or service) is conformant when the actual product meets the customers’ requirements.

• **Durability**: It is a measure related to the life-span of a product. It should be addressed from a technical and economical point of view. From a technical point of view, durability is “*the amount of use one gets before a product deteriorates*” [50]. From an economical point, “*it is about making the decision on how much to invest for repairing and maintenance of a product before retiring it.*”

• **Serviceability**: It is measured using the mean time to repair (MTTR) metric. The availability software quality attribute can be calculated by taking the reliability and serviceability into consideration. Availability is defined as Mean time between failures (MTBF) divided by the Mean time between failures (MTBF) plus the Mean time to repair (MTTR).
• **Aesthetics**: This attribute is related to the user approach of quality. Here quality is defined by the client by examining the appealing interface of the product and its ease of usage.

• **Perceived Quality**: This concept is based on the fact that most of the time users cannot access or compare all the quality features of the available products. So, products are evaluated on how they are perceived. Factors that influence the quality are the brand name, company image, advertisements, word-to-mouth etc.

### 3.3.2 Process Quality

Process quality focuses on improving the quality of the process. It will not only improve the quality of a product but also reduce the cost of production. There are many concepts related to process quality. Some are as follows:

- Statistical process control
- ISO 9000 family of standards
- Six Sigma
- Quality Function Deployment

The above mentioned qualities concepts are from manufacturing industries but most them are already tailored to software development. The most important model of software in this regard is CMM (i.e. Capability Maturity Model). This model was developed by the Software engineering institute (SEI). This model has five levels. Almost all software companies are rated under this. In order to get a good rating by CMM, they try to adopt all good quality practices. In the year 2003, the CMM was replaced by the Capability Maturity Model Integration (CMMI).

CMMI focuses not only on software processes but also on the processes in the entire organization. CMMI also provides suggestions on how to improve the existing processes. As said earlier there are five levels in CMM. These are as follows [51] [25]:

33
<table>
<thead>
<tr>
<th>Level No.</th>
<th>Level Name</th>
<th>Characteristics</th>
<th>Key Process Areas</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Optimizing</td>
<td>Continuous process improvement and control</td>
<td>• Innovation and Deployment</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Defect Prevention</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Process Change Management</td>
</tr>
<tr>
<td>4</td>
<td>Quantitatively managed</td>
<td>Software Process measured and controlled</td>
<td>• Software Quality management</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Quantitative process management</td>
</tr>
<tr>
<td>3</td>
<td>Defined</td>
<td>Software Process Defined</td>
<td>• Organization process definition</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Organization process focus</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Training program</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Integrated software management</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Software product Engineering</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Inter group coordination</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Peer Reviews</td>
</tr>
<tr>
<td>2</td>
<td>Managed</td>
<td>Basic software project Management</td>
<td>• Requirements management</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Software Project planning</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Software Project monitoring and control</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Subcontract management</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Software quality assurance</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Software Configuration management</td>
</tr>
<tr>
<td>1</td>
<td>Initial</td>
<td>Adhoc Process</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.5: CMM Levels

Now, I will discuss various attribute of software quality and its importance
3.4 Software Quality Attributes

There are various software quality attributes available and also some metrics to measure these attributes. The most important and relevant software quality standard is ISO/IEC 9126 [52]. This standard defines a set of quality dimensions and its characteristics like reliability, maintainability, usability, portability, security and so on. This standard also specifies the relationship between different approaches of quality. It differentiates between process quality, internal quality, external quality and quality in use attributes. This standard is subdivided into four parts in terms of software quality:

1. Quality Models
2. External Metrics
3. Internal Metrics
4. Quality in use metrics

The first model classifies the software quality into six characteristics and then subdivides into 27 sub characteristics. These characteristics are as follows:

**Functionality** - “The capability of the software product to provide functions which meet stated and implied needs when the software is used under specified Conditions” [52]. This characteristic is further subdivided into suitability, interoperability, accuracy, security and functionality compliance. It is a very important characteristic and commonly taken as the functions that the software supports. Any miss in this area can directly impact the image of the developing company.

**Reliability** - “The capability of the software product to maintain a specified level of performance when used under specified conditions” [52]. It contains maturity, fault tolerance, recoverability and reliability compliance as its sub categories. It is basically the capability of the software to maintain its performance during execution.
**Usability** – It is all about how user friendly the software is and is related to the extent of effort required to use or learn the software. It consists of learnability, operability, attractiveness and usability compliance as its sub categories.

**Efficiency**- It is related to amount of resources required for an appropriate or required performance of the software under stated conditions. The sub categories include Time behavior, resource utilization and efficiency compliance.

**Maintainability**- This characteristic measures the effort required to make the modifications like corrections, improvements or some adaptations based on environment. It includes analyzability, changeability, stability, testability and maintainability compliance.

**Portability** – It is related to the concept of transferring the system from one software, hardware or organization environment to another. It includes adaptability, co-existence, installability, replaceability and portability compliance.

The above mentioned ISO 9126 standard is widely applied in the software industry and provides a realistic set of definitions, metrics and evaluation criteria to analyze measure and assess software quality.

The scope of my current thesis is limited to reliability, maintainability and usability as these are key characteristics that are in demand today.