"As long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem and now that we have gigantic computers, programming has become an equally gigantic problem. In this sense the electronic industry has not solved a single problem, it has only created them - it has created the problem of using its product."

E.W. Dijkstra

Turing Award Lecture (Dijk72)
1.1 Prologue

We are already in the firm grips of the whirlwind of the Second Industrial Revolution caused by the computers. There is no escape from this 'Future Shock' as envisaged by Alvin Toffler (Toff79).

During this decade the computer technology has changed the fabric of modern society. The introduction of computerised consumer products and personal/home computers for hobbyist and layman for managing accounts, environment and day-to-day activities has been possible due to abundant computing power on a silicon chip and very soon we shall be witnessing Gallium chips and then Bio-chips!

Application of computer implies programming or software development. Because of proliferation of computers there is ever increasing demand for software. Software technology is going to have profound impact on our society.

Developing software is an activity that requires much intellect. Evolving an efficient, reliable and maintainable software product in time is an uphill task. We can write programs consisting of several hundred lines of codes. Maintaining this program is not difficult. However, designing
and implementing systems consisting of tens of thousands of lines of codes is herculean task. The effort required to complete such a system is beyond the comprehension and capacity of one person. And, when we add more people to the project to share the effort, communication and management problems are introduced. Maintaining such a system is difficult since no one person can comprehend its structure. It is a fact that at one time or the other every programmer has frustrating experience while working on large software systems because of strange side-effects of modules or badly defined interfaces.

1.2 The Case of MISI

In order to bring home our point, it is worthwhile to mention the case of a firm MISI of Louisiana in USA. It was started by two brilliant and expert professors of a local university where they taught Computer Science. The two embarked upon developing and marketing software for health care units. In five years they had a very good business by selling seven large systems and employing about twelve highly educated analysts/programmers. The environment under which this firm operated was quite favourable and prospective. However, when the size and number of systems got larger than a critical mass, it was impossible to meet the schedules and maintain the quality of the software developed. This resulted in a litigation from one customer who could not get the software to work satisfactorily in time. The MISI had to be closed under the cover of bankruptcy (Sada85).
1.3 **Software Crisis**

The first public recognition of the existence of this crisis did not appear until the International Conference on Software Engineering (NATO Science Committee) held at Garmisch(Germany) in 1968. Essentially it implies that it is much more difficult to build software systems than our intuition tells us it should be. Following are the symptoms of software crisis which were identified during late sixties but still persist:

1. Software does not meet users requirements (irresponsive)
2. Software costs are excessive and unpredictable (costly)
3. Software is often delivered late (untimely)
4. Software often fails (unreliable)
5. Software maintainence is quite costly, complex and error prone (unmaintainable)
6. Software from one system is seldom used in another (untransportable)
7. Software development efforts do not make optimal use of the resources involved - memory, processing time etc. (inefficient).

Fig. 1.1 is the representation of how the ratio of estimated hardware to software costs has rapidly moved to the point where software costs are dominant. Also, the costs for software maintainence are often greater than the costs of original development. As the Fig. 1.2 indicates in USA DoD software costs are expected to increase phenomenally (Booc83). These rising costs are mostly due to an expected large increase in computer applications.
1.4 Software Exigency

Retrospectively, we can describe the software phenomenon of the last quarter of this century with significant events - computer technology has advanced so rapidly and the requirements for sophisticated software system so demanding that a remarkable lag in software technology, project management and talent growth has been created resulting in software exigency as shown in Fig. 1.3. In fact the Malthus theory is applicable here. The requirements have been increasing in geometric progression while software technology is developing in arithmetic progression.

Software complexity has steadily increased. In the early days, software technology was able to satisfy customer needs because software requirements were simple due to hardware limitations. As the hardware became more powerful and economic, much more imaginative requirements were defined and the degree of software complexity reached a point at which the software technology was inadequate. This gap is continually increasing and the software technology has not been able to keep pace to decrease this exigence (JeTo79).

It should be noted that at one time, problems involving large computer memories, faster CPU, large auxiliary storage were declared impractical and were abandoned but the same are in limelight and active field of interest i.e. pattern recognition, natural language translation, artificial intelligence, KB systems, SDI etc.
1.5 Software Aspects of SDI

In June,'85, Dr. D.L. Parnas, an esteemed computer scientist with twenty years of software engineering research experience including about ten years of real-time software development work for DoD, resigned from the panel on Computing in Support of Battle Management convened by SDIO (Strategic Defence Initiative Organization). He submitted a report explaining why the software required by SDI for 'Star Wars' effort would not be reliable putting a question mark on the feasibility of the project. His conclusions are not based on political or ethical considerations but the software engineering aspects of SDI. The report (Parn85) which is worth reading created a stir in the scientific community and summarily explains the following points:

1. The fundamental technological differences between software engineering and other areas of engineering and why software is unreliable;

2. The properties of SDI software that make it unattainable;

3. Why the techniques commonly used to build defence software are inadequate for this job;

4. The nature of research in software engineering and why the improvements that it can effect will not be useful.

Incidently, this is not the first time, earlier Prof. F.P. Brooks had to resign from IBM 360 OS project when he was not able to produce the bug-free product as per schedule and within budget (Broo75).
1.6 The Problem

Computers are with us for about three decades. During this period hardware technology has taken long strides. The computer applications have also grown in magnitude and complexity. But we are not able to develop software synchronously to meet these requirements as this constitutes projects involving groups of skilled people doing highly creative work in a system context.

We have not yet fully established an engineering type discipline for planning, controlling and execution of software project. Software project management depends upon realistic and quantitative estimation to provide data from which a project may be costed and progress monitored.

Summarily, in this thesis our endeavour has been to investigate some vital aspects of software project management and to provide quantitative answers to resource estimation problems of feasibility, costing, duration, staffing, manloading, cashflow, trade-off and risks by applying what we call 'Integrated Life Cycle Methodology' to software projects.
FIG. 1.1 - HARDWARE – SOFTWARE COST TRENDS IN USA (Boch 79)

FIG. 1.2 - DoD HARDWARE/SOFTWARE COST RATIO (Whel 86)

FIG. 1.3 - SOFTWARE EVICENCY