Creativity combined with our understanding of nature, material, medium, other humans, and artifacts has always helped us in devising new processes for performing old tasks, and also devising new tasks in our personal, social, professional, and organizational lives. New processes and tasks require the use of existing artifacts in new ways, and also the creation of new artifacts. Often new processes bring advantages in terms of increased speed, reliability, scale, safety, comfort, or flexibility and/or savings in effort, energy, material as well as costs. In addition, humans have also used themselves both as the source of raw energy through physical labor, and as controllers through psychomotor skills to perform these tasks. Taming of animals, tapping of natural energy, steam engine, electricity, etc., helped to reduce our role as energy suppliers. Mankind could focus more on the other two tasks of being the controller and process designers. With the availability of control systems in the last century, our role as controllers has also reduced significantly, and more human energy is now available for the creative work of devising new processes and new tasks. Artisans, engineers, designers, and technologists play a key role in identifying the opportunities and developing new processes and tasks in diverse domains of human activities. Strength, malleability, expected life, and various other affordances of the material and medium influence and constrain our design activities. The digital computer is the most malleable artifact created so far, and it can be further used as a material and a medium to rapidly create a large variety of new artifacts in a very flexible way. This power has given an unprecedented boost to the development of new processes, as well as new tasks in all domains of human activities.

Engineers and technologists plan, design, develop, test, integrate, deploy, maintain, improve, reverse engineer, re-engineer, as well as evaluate components, products, applications, systems, services, standards, processes, and methodologies encompassing various artifacts. Their disciplines are differentiated with each other on the basis of the artifacts they build and focus on. In order to identify and create opportunities of devising new processes and ways in various domains, they need to understand the needs and nuances of those domains as well as humans’ individual as well as social behavior. This is often the most critical and creative task, especially when the subsequent engineering processes are very rapid and fairly stabilized. US Accreditation
Board for Engineering and Technology (ABET) defines engineering as follows: “Engineering is the profession in which knowledge of the mathematical and natural sciences gained by study, experience, and practice is applied with judgment to develop ways to utilize, economically, the materials and forces of nature for the benefit of mankind.”

Unlike science, engineering and technology are oriented towards conception, design, invention, development, application, improvement, and production with an emphasis on current and future needs of society. They require holistic thinking involving integration of many competing demands, theories, data, and ideas as well as decision making based on incomplete data and approximate models. The theorizing attempts go beyond the search of causes, and are focused on new processes and applications. Engineering is not just applied science; it is as much about process as it is about technical knowledge. An engineer’s task involves conceiving and designing products, processes, and systems, and to predict their behavior using science. Scientists create models to understand natural phenomenon with known outcomes, whereas engineers create models to predict outcomes for systems. The use of heuristics distinguishes engineering methods from scientific methods. Engineering is further distinguished from Technology by its focus on more complex problems that involve use of more diverse resources, more diverse groups of stakeholders with varying needs, wider range of conflicting technical, engineering and other issues, more abstract thinking, originality, infrequently encountered issues, and work progress in spite of insufficiency of standards and codes of practice. Technological work needs mastery of discipline and context specific current knowledge, techniques, skills, and tools. A higher focus on quality and timeliness are its distinctions. Broadly, the educational programs of engineering and technology recognize many of these distinguishing aspects of the discipline and respond in various ways through their curriculum and educational methods.

As per the ACM-IEEE joint report [1], Computing means any goal-oriented activity requiring, benefiting from, or creating computers. It includes: designing and building hardware and software systems for any of a wide range of purposes, processing, structuring and managing various kinds of information, doing scientific studies using computers, making computer systems behave intelligently, creating and using communications and entertainment media, finding and gathering information relevant to any particular purpose, etc. Computing engineers
are concerned with four kinds of artifacts: (i) software, (ii) digital ICs and other hardware, (iii) embedded systems, and (iv) digital content. For the last four decades the demand of software developers has been increasing at an accelerated rate. Jalote [2] summarizes the growth of Indian software industry as follows: “It started primarily as a subcontractor for technical manpower. ... shifted to doing complete parts or phases of projects, usually the later phases of coding and testing. ... matured to providing complete solutions offshore. ...most leading companies are operating in the high-end software services business. ... a large number of software companies matured to CMM level 4 or level 5...”

In the last few years, there has been an exponential growth in engineering education, especially in India and China. This growth has led to an era where fresh graduates of computer science related disciplines are easily absorbed in the industry. Indeed, to satisfy the growing demand for software, very large volumes of engineers from other engineering disciplines are also absorbed as well. All engineering graduates are considered to be ready for a direct fit with the requirements of the IT industry [3]. The core competencies developed in all engineering disciplines are considered to be sufficient, and the companies rely more on their own finishing schools for specialized computer science and IT knowledge.

With the advent of the Internet, it has become possible to outsource software development tasks to remote sites, making India an attractive destination, both technically as well as financially. This has resulted in an exponential increase in the demand of software developers in India, especially in the last decade. It has become a challenging opportunity for Indian academic institutions to provide an adequate pool of software professionals of desired quality to the rapidly growing Indian software industry.

To meet this challenge, the Indian academic institutions have been able to expand fast and satisfy the industry’s need of software professionals quantitatively. However, the quality of ‘most of’ the professionals they generate is below the desired industry expectation. The software industry associations, as well as the academic regulatory bodies, have repeatedly shown their concern emphatically about the sub-standard quality of ‘the majority of’ fresh software professionals [4]. Most of the software houses spend around six months to one year in their post-induction, in-
house software development education and training of fresh engineers. It clearly indicates that there is a significant gap in the technical education that academic institutions impart to their software graduates, and what technical expertise the industry expects in them. A *competence mismatch* exists between academic technical offering and software industry employability. We elaborate upon this in the second and third chapters.

*It is not proper to consider software industry as monolithic group.* Even in India, there are different kinds of companies, those involved in software services, and those involved in new product development in large or small companies. There are huge differences in the requirements of these categories. Often, India’s highly dominant software service industry’s immediate requirements dwarf other requirements, which are more futuristic and even more compatible with the goals of excellence in higher education. In sections 2.9 and 2.11, we especially examine the needs different kind of software industry. A NASSCOM-KPMG study [5] argues that key skills required by the industry are not met by the current educational system. It quotes the following observations from a World Bank study on science and technology manpower in India published in 2001: (i) faculty lacks industry rigor, R&D background, and exposure to tools, (ii) students lack opportunity and encouragement for creative thinking, (iii) inflexible and rigid curriculum is not exposed to innovation/industry, (iv) teaching is examination oriented without focus on communication and problem solving skills, (v) continuous evaluation is often not systematized, and (vi) examinations are often memory based, and encourage partial studying through ample choice.

Organizations and their clients have limited tolerance for inept performance. Often engineers engage directly with clients in complex interactions. Educators are expected to teach competencies that are relevant and enhance an organization's performance [6]. Stephen says, “Anyone not aware that this is a time of change in higher education is asleep at the helm” [7]. Universities around the world have become increasingly aware of the need to be able to demonstrate, in a quantifiable manner, the skills and attributes that their graduates are imbued with during their learning experience [8].
State of Indian Contribution in Computing Research

There are over a million software engineers working in India. Further, there are over two thousand colleges offering degree level educational programs in computing. The IT industry’s share in India’s GDP is more than 7%. Seven Indian IT companies have been listed in the top 15 technology outsourcing companies of the world. However, Indian organizations’ contribution to computing research literature remains very meager. The ACM digital library gives access to almost 0.3 million papers. Less than 0.7% papers have been contributed by authors having Indian affiliations. Before 2005, this fraction was only 0.3%. During 2005 to Feb 2010, it increased to 1.3%, which still is a very small number, given the huge number of software engineers and colleges offering computing degrees in India.

A focused search (using affiliation option under advanced search) in March 2010 showed that some of the largest India-based IT companies, i.e., TCS, Infosys, Wipro, HCL, Satyam, Oracle India, have together so far collectively published less than 100 papers that are indexed on this digital library. This library does not include a single paper from other very large Indian IT companies like Tech Mahindra, Patni Computers, and Birlasoft. On the other hand, Microsoft India and IBM India have published approximately 300 papers, and Microsoft and Google have contributed 3,885 and 582 papers respectively. This highlights that the mismatch is not just in terms of immediate specific needs of industry, but also long term goals of professional excellence. This numbers highlight the gross mismatch between published contributions and the size of India’s IT industry, and the number of computing professionals in the industry or academia. In addition to meeting industry’s short term needs, software development education can also stimulate the overall growth of India-based computing research contributions by arousing interest among future software developers.

Section 1.1: Basis for the Need for Reforms in Computing Education

This thesis attempts to contribute towards bridging this competence mismatch by providing ideas for instructional reforms in computing education with special reference to software development. Unlike other disciplines of engineering, computer scientists have always remained interested in understanding the phenomenon of ‘learning.’ Artificial intelligence and computer based teaching were the earlier sub-disciplines within computing that required and encouraged computer
scientists to understand various issues associated with ‘learning’. The International Federation for Information Processing (IFIP) established a technical committee on education in 1963. In its very early years, The ACM also founded special interest groups SIGCSE (Special Interest Group on Computer Science Education) and SIGCUE (Special Interest Group on Computer Uses in Education). More recently, the ACM has started SIGITE (Special Interest Group on Information Technology Education).

Reforms in engineering education have a long but slow history. Felder [9] remarked, “We teach primarily mechanics, and not reasoning methods; memorization and routine application, and not analysis, synthesis and evaluation. We don’t encourage creativity and independence of thought, and in fact often do our best to discourage them.” Sadly nothing much has changed on the ground. The community that is responsible for transforming the lifestyle of the world has not yet transformed its own educational process.

Many engineering faculty have never practiced engineering [10]. The curriculum’s focus on content is disconnected from engineering practices [11-12]. Felder and Brent [13] reported on some recent studies that measured the intellectual growth of engineering students during their studies using Perry’s model of epistemological development [14]. It was observed that the engineering education failed to elevate a significant number of students to level 5 as per Perry’s nine-level model, and the average growth after four years of college was only one level, with most of the change occurring in the last year.

Our exploratory study has shown that the kind of activities that a typical engineering student is generally engaged in, does not help in enhancing creativity, critical thinking, and innovative problem solving [11-12] However, the last decade has seen an increasing recognition of the need for transformation. A certain section of policy makers, universities, accreditation agencies, and faculty members have made tremendous contributions to bring the much needed transformation. Many accreditation agencies have even transformed their accreditation criteria in the last few years. This is expected to drive an unprecedented transformation of instructional programs in responding institutes. This challenge can only be met by undertaking large scale research in engineering education.
Recognizing the need to re-engineer the engineering education, a recent report ‘Educating the Engineer of 2020’ [15] suggests that “the engineering education establishment should endorse research in engineering education as a valued and rewarded activity for engineering faculty as a means to enhance and personalize the connection to undergraduate students, to understand how they learn, and to appreciate the pedagogical approaches that excite them.”

One of the founding fathers of modern education, Franklin Bobbitt observed that curriculum should aim to teach those subjects that are not sufficiently learnt as a result of normal socialization. In 1920s, he proposed a five-step process for curriculum design: analysis of human experiences in a field, job analysis to identify specific activities, deriving objectives to identify the abilities required for specific activities, selecting objectives as the basis of students’ activities, and planning in detail. Paulsen and Peseau [16] proposed a framework of Zero Based Curriculum Review process that starts with first operationalising the curriculum goals as categories of required professional competencies, and then identifying appropriate knowledge base learning objectives and also behavioral objectives in terms of professional practices, and skills with respect to required professional competencies.

Woods et al [17] proposed the following process for engineering faculty: (i) identify the skills you wish your students to develop and communicate their importance to the students, (ii) use research, not personal intuition, to identify the target skills, share some of the research with the students, (iii) make explicit the implicit behavior associated with successful application of the skills, (iv) provide extensive practice in the application of the skills, using carefully structured activities, and provide prompt constructive feedback on the students’ efforts, (v) encourage monitoring, (vi) encourage reflection, (vii) grade the process, not just the product, and (viii) use a standard assessment and feedback form.

An exploratory informal discussion with a large number of faculty members of engineering institutes with teaching experience ranging from a few months to several decades, and coming from different departments of engineering, sciences, and management, it was found that most were not aware of any literature in educational research. Hence, by and large engineering
education methods have remained unaffected by such research. In 1982, Professor Richard Felder [9] presented a revolutionary thought that ‘does engineering education have anything to do with either one.’ The curriculum and educational committees of the ACM, IEEE, AIS, AITP, LACS, IFIP, etc. have mostly ignored the rich educational literature related to curriculum design, instruction design, assessment methods, theories of learning, human development, epistemology, and sustainable development. Only a few of the available theoretical models and frameworks in these education related areas have been examined, reviewed, and/or used by the researchers of software development education.

UNESCO has labeled 2005-2015 as the decade of education for sustainable development. In this decade, bodies like National Science Foundation (NSF), USA and the National Academy of Engineers (NAE), USA have emphasized the need of systematic research in ‘learning’ to transform engineering education. In 2006, NAE identified the following research areas for engineering education [18]:

1. Engineering Epistemologies: Research on what constitutes engineering thinking and knowledge (technical, social, and ethical aspects) within social contexts now, and into the future.

2. Engineering Learning Mechanisms: Research on engineering learners’ developing knowledge and competencies in context.


4. Engineering Diversity and Inclusiveness: Research on how diverse human talents contribute solutions to the social and global challenges and relevance of our profession.

Section 1.2: Evolution of Software Development Education

In this section, we discuss the evolution of software engineering education. Table 1.1 gives a list of some of important reports examined in this discussion.

<table>
<thead>
<tr>
<th>Table 1.1: Some important reports on computing curriculum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ACM curricula committee for CS (1965)</td>
</tr>
<tr>
<td>2. ACM Curricula for CS (UG and PG) (1968)</td>
</tr>
<tr>
<td>3. COSINE’ IEEE for CS in EE (1968)</td>
</tr>
<tr>
<td>4. COSINE’ IEEE for CS in EE (UG) (1971)</td>
</tr>
<tr>
<td>5. ACM curriculum on IS (UG) (1972)</td>
</tr>
<tr>
<td>6. ACM curriculum on IS (PG) (1973)</td>
</tr>
<tr>
<td>7. IEEE Model Curricula for CSE (UG) (1975)</td>
</tr>
<tr>
<td>8. IEEE Model Curricula for CSE (UG) (1977)</td>
</tr>
<tr>
<td>10. ACM Curricula for CS (UG) (1978)</td>
</tr>
<tr>
<td>11. ACM Curricula for CS (PG) (1981)</td>
</tr>
<tr>
<td>13. IFIP curriculum for CS (1984)</td>
</tr>
<tr>
<td>14. CMU curriculum for CS (UG) (1985)</td>
</tr>
<tr>
<td>15. LACS Model Curriculum for CS (UG) (1986)</td>
</tr>
<tr>
<td>16. ACM report on Computing as a discipline (UG and PG)</td>
</tr>
<tr>
<td>17. SEI model curriculum for SE (UG) (1990)</td>
</tr>
<tr>
<td>20. IFIP curriculum for CS (UG) (1994)</td>
</tr>
<tr>
<td>22. ACM curriculum on IS (UG) (1997)</td>
</tr>
<tr>
<td>23. IFIP curriculum for Informatics (UG) (2000)</td>
</tr>
<tr>
<td>25. AICTE curriculum for IT (UG) (2000)</td>
</tr>
<tr>
<td>27. SEI-CMU Software Engineering Body of Knowledge Ver 1.0 (1999)</td>
</tr>
<tr>
<td>32. ACM- IEEE curriculum for CS (2005)</td>
</tr>
<tr>
<td>33. ACM-IEEE curriculum for IT (2005)</td>
</tr>
<tr>
<td>34. LACS curriculum for CS (UG) (2007)</td>
</tr>
</tbody>
</table>

Beginning of Computing and Computing Education

Computing in the form of processing: understanding, creation, manipulation, communication, expression, and rendering of symbols has always been a very important natural activity of human mind. Though the use of the term computing is not limited to be used in the limited context of processing of formal mathematical symbols, computer software transcends such boundaries to support processing of diverse range of symbols. With the invention of computing machines, the field of computing has advanced beyond one’s imagination. Computing has transformed many aspects of everyday lives for a vast majority of mankind. The role of computing has been evolving from enhancing efficiencies through otherwise by-passable support systems to creating real-time mission critical systems. The initial application domains driving computing till 1960s were code breaking, engineering calculations, scientific simulation, as well as repetitive data processing in defense, space, government, insurance, banking, and some other large business organizations. Some attempts of language translation and information retrieval were also made even in 1950s. Outgrowing the initial goal of doing repetitive mathematical calculations, computers have already permeated almost all spheres of human activities even including arts and
sports. The socio-cultural effect of computing and communication technology is much wider, deeper, and faster than the effect of other technologies. Computing has also been used to expand our understanding of mind and reasoning.

India’s decimal number system inspired ninth century Persian mathematician Mohammed ibn Musa al-Khowarizmi to write a book on calculating using this number system. Based on his name, *Algorism* slowly started referring to arithmetic operations in this number system. These algorithms were strictly mechanical procedures to manipulate symbols. They could be carried out by an ignorant person mechanically following simple rules, with no understanding of the theory of operation, requiring no cleverness and resulting in a correct answer. The word *Algorithm* was introduced by Markov in 1954 [53]. Before the 1920s, the word *computer* was used for human clerks that performed computations. In 1936, Turing and Zuse independently proposed their models of the computing machine that could perform any calculation that can be performed by humans. In the late 1940s, the use of electronic digital computing machinery based on stored program architecture became common.

In the late 1950s saw the arrival of high level languages. The Association of computing Machinery (ACM) was founded by Berkeley in 1947. It started its first journal in 1954. Mathematical logic and electrical engineering provided the foundation for building modern computers. The personnel training responsibility was largely taken up by the manufacturers themselves. Most early programmers were math graduates, many of them were women. In the 1950s, a large numbers of private computer schools emerged to fill the burgeoning demand [19]. The word *software* was coined by John Tukey, famous statistician, in 1958. The words *computer science, information systems, information technology, system analysis, and system design* were being used even before. Dunn of Boeing [20] defined Information Technology as a body of related disciplines which lead to methods, techniques, and equipment for establishing and operating information processing systems. He also provided a simple definition of information systems as a connective link between five basic management functions of defining objectives, planning, gathering resources, execution, and control. In 1968, the computer science study group of NATO Science Committee coined the word *software engineering* to imply the need to transform software design and development into an engineering-type discipline.
Till 1970’s, computing was often regarded as a subfield of one or more of a mixture of disciplines of mathematics, operation research, electrical engineering, statistics, industrial engineering, and management. Many of existing undergraduate programs of these disciplines were modified to accommodate some of the naturally fitting aspects of computer science. Mathematics departments taught practice and science of programming and numerical analysis. The electrical engineering department emphasized on design and construction of electronic digital computer, and management schools paid more attention of design of information systems. Initially, masters and later undergraduate degree programs and departments of computer science were emerging as offshoots of the mathematics departments in colleges of science and arts. Stanford established its computer science department in 1962, and by the late 1960s many universities in United States had started computer science departments. Concurrently, the management schools and others interested in business data processing applications focused on information systems, and started developing these programs. The engineering schools offered computer technology and computer science programs, and also computer as an option in various existing programs.

**Early Curriculum Recommendations by ACM**

The Association of Computing Machinery (ACM) unified the pioneering efforts of several universities and stimulated the process through its two independent curriculum committees established in the mid 1960s. The International Federation for Information Processing (IFIP) established a technical committee on education, TC-3, in 1963. Simultaneously, Various other professional agencies like the Computer society of the Institute of Electrical and Electronics Engineers (IEEE), and Data Processing Management Association (DPMA) made significant contributions in these efforts.

The first ACM Curriculum Committee on Computer Science (C³S) was formed in 1964. In its preliminary recommendations [21], the committee posited that computer science is concerned with information in much the same way as physics is concerned with energy. It mainly identified careers in systems programming and application programming for computer science students. It distinguished computer science from mathematics by highlighting that while mathematician is
interested in discovering the syntactic relation between elements based on a set of axioms which may have no physical reality, the computer scientist is interested in discovering the pragmatic means by which information can be transformed to model and analyze the information transformation in the real world. The final report, Curriculum’68, considered development of programming skills as an important by-product rather than the main purpose of the computer science programs. It emphasized that computer science programs must provide the student with the intellectual maturity to stay abreast of their discipline, and also interact with other disciplines through liberal education. The curriculum recommendation [22] identified three major categories of computer sciences subject areas. These were information structures and processes, information processing systems, and methodologies. The first category of information structures and processes concerned with representations and transformations of information structures, and theoretical models for such representations and transformations. It included data structures, programming languages, and models of computation. The second category of information processing systems included computer organization, translators and interpreters, computer and operating systems, and special purpose systems. The last division of methodologies focused on broad areas of applications of computing which have common structures, processes, and techniques. It incorporated numerical mathematics, data processing, symbol manipulation, text processing, computer graphics, simulation, information retrieval, artificial intelligence, process control, and instructional systems. The committee recommended the inclusion of at least two courses from each of three categories for a masters program in computer science. For the undergraduate program, the essential computer science courses included introduction to computing, computers and programming, introduction to discrete structures, numerical calculus, data structures, programming languages, computer organization, and systems programming. The committee recommended the inclusion of at least two of the following computer science courses for indicated specialization: (i) compiler construction for applied systems programming and data processing application programming, (ii) switching theory for all other than scientific application programming, (iii) sequential machines for computer organization and design, (iv) numerical analysis-I for scientific application programming, and (v) numerical analysis-II for scientific application programming.
Early Engineering Perspectives

Electrical engineering departments identified computers as one of their main components. The Committee on Computer Science in Electrical Engineering (COSINE Committee), National Academy of Engineers (NAE), USA published recommendations for infusing computer science in electrical engineering curriculum. This led to the formation of computer engineering programs in electrical engineering departments. Developments in computers started to help in developing new methods of solving engineering problems. The COSINE committee strongly recommended [23] a total reorientation of electrical engineering curricula from analog and continuous to digital and discrete. In 1968, the computer science study group of NATO science committee coined the word software engineering to imply the need to transform software design and development into an engineering type discipline. This, however, was given legitimate attention as an academic discipline in the late 1970s.

In 1971, the COSINE committee recommended the start of a new undergraduate program called computer engineering within electrical engineering departments. This program was conceived as an engineering program with emphasis on the concepts of design of software, hardware, and systems. It proposed three specialization options under this program: (1) digital systems engineering, (2) software systems engineering, and (3) theoretical computer science and engineering. A juxtaposition of the COSINE subject list with the list suggested in C³S’ Curriculum’68 for computer science shows that, while on one hand, C³S recommendations had ignored the hardware and design aspects, the COSINE recommendations ignored discrete structures and data structures. In 1975, IEEE computer society education committee identified and addressed this dichotomy in their recommendations, and proposed a new undergraduate program on computer science and engineering integrating courses in hardware systems, software systems, and theory of computing [24]. These courses were expected to constitute approximately 50% course requirement. The remaining 50% courses were to be in the areas of humanities and social sciences, physics, chemistry, communication, mathematics, economics, electronics, and engineering sciences as per Engineers Council for Professional Development (ECPD) guidelines. Sloan [25] and Engel [26] compared the new evolving recommendations of C³S and model curriculum of the IEEE Computer society and concluded that the two were virtually same with
respect to their recommendations in the area of software engineering and program design. Their emphasis differed with respect to hardware and logic design on one hand and theory on another.

**Early Information Systems Perspective**

The Curriculum Committee on Computer Education in Management (C³EM) of ACM published a position paper in 1971 [27]. Education for improving organizational productivity through information technology was the main motivation for this and subsequent committees in this area. This committee felt concerned about the unfavorable attitude of computer science departments towards applied problems. A few years later, this committee evolved into the ACM Curriculum Committee in Information Systems (C²IS). In its recommendation report submitted in 1972 and 1973 [28-29], it identified requisite knowledge and abilities of information system graduates and grouped these into six categories of people, models, systems, computers, organization, and society. The ACM curriculum committee of computer science did not pay specific attention to this aspect until 1980s, and depended on general liberal education to provide the necessary breadth without specifying their specific recommendations.

These two C³EM reports explicitly recognized two categories of information system programs at masters as well as undergraduate level: (1) technically trained systems designers, and (2) managerially oriented information analysts. The committee recommended the inclusion of five major topic areas of computer science, information systems, management, operations research, and systems design techniques. In 1973, this committee published its recommendations for undergraduate programs, and strongly argued for starting undergraduate programs in information systems in the light of very high manpower requirement at programmer and systems analyst level. It encouraged the computing centers as well as departments of computer science, business, electrical engineering, and industrial engineering to start undergraduate programs with their chosen concentration options on technology or organization. The committee also recommended one-year masters program in information systems for these students.

A few years later, this committee evolved into the ACM curriculum committee in information systems (C²IS). It is not clear why the committee chose not to explicitly include computer
programming as a compulsory course in the technology concentration. This anomaly was corrected in the 1982 recommendations of C²IS.

In later decades, a new trend of domain specific computing programs emerged. This trend resulted in establishment of many programs like medical or health informatics, geo-informatics, bio-informatics, chem-informatics, social informatics, and so on. In 1978, the ACM curriculum committee on health computing published its recommendations [30]. In many of these domain specific programs, up to 50% of the course content was related to domain specific foundations and domain specific aspects of informatics. The remaining courses focused on generic mathematics, statistics, information systems, computer science, and general education. ACM curriculum committee cautioned against somewhat frivolous proliferation of specialized programs [31]. However, in current era, specialized programs addressing the needs of specific domains are becoming important.

The 1981 report of C²IS [32] emphasized that the demand of personnel with technical and organizational skills is relatively much greater than the demand for solely technical skills or organizational skills. It expressed its general concern over the ad-hoc basis of instruction of systems analysis and design. In its 1982 report [33], this committee proposed separate MS and MBA programs for the two streams of information systems.

As per the 1982 recommendations of C²IS, considering the nature of the professional work of information system specialists, a strong emphasis (more than 20%) was placed on social sciences and humanities including economics, psychology, and English. It was argued that such a background helps in development of many essential attributes of requirement and systems analysts. The hiring of computing professionals in India has always been highest for information systems and software engineering related work. However, it is surprising that such undergraduate engineering programs have not been developed in India. The three year Master of Computer Application (MCA) programs also have a relatively heavier dominance of computer science and management related courses, and pay only little attention to these aspects related to the computing profession. The lack of strong industrial participation in curriculum design,
professional inclinations of curriculum designers, and educational politics in India may have contributed to this phenomenon.

**Liberal Arts Perspective**

The model curriculum recommended by Liberal Arts Computer Science Consortium (LACS) attempted to define computing program in terms of their approach towards data structures and algorithms [34]. It proposed that a computer science program is more interested in the formal properties of data structures and algorithms, a computer engineering program focuses more on their realization, and an information systems program is more orientated towards applications. Even after two decades with many changes in computing arena, in its 2007 model curriculum, LACS has only slightly modified their original definition of computing programs. The realization part has now been partitioned into two categories of linguistic and hardware realization.

The 1986 report and all subsequent reports of LACS, put more emphasis on discrete mathematics and place it along with first introductory computing course before other mathematics courses. In addition to two introductory computing courses, the 1986 report proposed four core computing courses on computer organization, algorithms, theory of computation, and principles of programming languages. These recommendations were only marginally revised by the consortium even after ten year [35]. In its 2007 recommendations, software development has been added to this category.

A typical liberal arts computer science program is more broad-based than specialized programs, and it includes more than 50% non science courses in the area of humanities, social sciences, etc., [36] It is unfortunate, that such programs do not exist in India, and software development education is mainly linked with engineering programs. This possibly has contributed to a nearly non-existing or marginal inter-disciplinary activity between computer science and these areas. In the west, it is not uncommon to have a degree in computing and philosophy, computing and art, and so on. Perhaps, it is time to consider the option of a liberal arts oriented design degree with specialization in computing in India.
**Changing Role of Mathematics in Computing Curriculum Recommendations**

In the first decade, the computer science curriculum was lesser oriented towards business data processing needs. Interestingly, discrete structures and three courses in numerical methods were not considered as part of mathematics courses. Instead they were included as essential computer science courses. The committee further suggested a minimum of six mathematics courses for undergraduate programs. The committee proposed essential inclusion of courses in related areas of mathematics, statistics, electrical engineering, philosophy, linguistics, industrial engineering, and management. Overspecialization at undergraduate level was discouraged by the committee, and it also encouraged the deep involvement of computer science faculty in computer applications. Scientific simulation and engineering calculation oriented applications encouraged to put a strong emphasis on numerical methods.

The strong emphasis on numerical methods decreased gradually through subsequent recommendations, and it was eliminated from the core in nearly all subsequent recommendations of the ACM, IEEE Computer society, as well as other bodies except International Federation of Information Processing (IFIP). Computing curricula [37] does not specify any minimum required weight of numerical techniques for any of the five computing discipline – computer science, computer engineering, information systems, software engineering, or information technology. It is not recommended even as an elective course for the later three disciplines.

On the other hand, discrete mathematics was increasingly being recognized as more central and fundamental for computer science than calculus [38-40]. There were proposals to teach discrete mathematics as the first mathematics course, and the model curriculum for liberal arts degree in computer science responded favorably [41] [34]. In 2001, 76% faculty members are reported to have felt that discrete mathematics should be a prerequisite to data structures [42]. However, many universities and institutions were slow to respond to this change. A survey [43] showed that even in late 1980s, nearly 30% universities and institutions in USA did not include discrete mathematics, and nearly 27% maintained numerical algorithms in the core curriculum of computer science.
Possibly because of IFIP influence, for quite some time, numerical techniques continued to be part of the core curriculum of many computing programs in India for some time. The current model curriculum recommended by the All India Council for Technical Education, India [44-45] has not included numerical mathematics as a core course for both the commonly offered undergraduate computing programs of engineering institutes: (1) computer science and engineering, (2) information technology. Unfortunately, even discrete mathematics is excluded from the list of AICTE’s information technology curriculum.

Over the decades, with the advent of faster, cheaper, smaller, reliable, networked, and mobile hardware, as well as user friendly and multi-layered software, the computer applications have rapidly expanded much beyond the scope of computational science around numerical techniques, modeling and simulation, and operation research. Lethbridge [46-48] found that in the list of the most important twenty-five subject topics of the university curriculum, professional software engineers did not include a single topic of mathematic. Though computational science is recognized as an extremely valuable closely related discipline, the recommended core body of knowledge of computing curricula with specialization in computer science, computer engineering, software engineering, information systems, or information technology, do not include these courses any more [49-50].

Further, the ACM-AIS-IEEE joint report [51] has recommended a lowered minimum requirement for mathematical foundation for programs in software engineering, information systems, and information technology. ACM-IEEE joint curriculum recommendation on software engineering [52] has included only one topic of mathematics ‘discrete mathematics’ as part of the essential core. Recently, differentiating computer science from mathematics, Fant [53] argues that rather than computational issues, computer science is more concerned with issues related to creation and actualization of process expressions.

**Human and Social Aspects in Computing Curriculum**

Till the 1970s, sociological, economic, and educational implications of developments in computer science were not considered as major responsibility of computer science. The report
recommended that computer science faculty should cooperate with concerned departments to
develop courses in these areas, and computer science students should be encouraged to take these
courses. However, computers were been increasing recognized as agents of social change.
Professional bodies started paying more attention to understanding the social impact of
computing.

In 1976, IFIP added a new technical committee, TC-9: Relationship between Computers and
Society. The ACM curriculum committee also responded to this trend, and included *computers
and society* as a strongly recommended elective in Curriculum’78 [54]. It was also suggested that
such a course should be taught by computer science faculty. The committee recommended that
meaningful computer applications should be cited and reviewed throughout the elementary
material. The committee posited that structured programming along with social, philosophical,
and ethical considerations are of such importance to the development of computer scientists that
they must permeate the instructions at elementary levels. In all subsequent recommendations of
the ACM, IEEE, IFIP, and others this proposal was further strengthened and this course was
often included in the core. Most of the subsequent recommendations provided a more central
position to this area. IFIP [55] recommended *computer and society* as part of the core for six
variants of computing programs. Computing curricula [37] specifies ‘2’ as the minimum weight
of legal, professional, ethical, and social aspects on a scale of 0-5 for all their five forms of
undergraduate computing discipline. However, some studies [46-48] showed that in spite of
strong recommendations from professional bodies, this area received lesser than required
attention during formal education in the opinion of responding practitioners.

The C³S published a survey of computer science education [56]. This report was a mere catalog
of various reports and papers without any observations or conclusions. It badly failed to critically
review the previous literature or propose future trends. A year later the committee on computer
science published their new recommendations, Curriculum’78. Mathematics requirements were
mostly unchanged, and the report was criticized for being retrogressive in this aspect. The
committee posited that structured programming along with *social, philosophical, and ethical
considerations* are of such importance to the development of computer scientist that they must
permeate the instructions at elementary levels. The core computer science and mathematics
courses constituted less than 50% of course requirement. Additional course requirements were proposed to be fulfilled through electives and courses in humanities, sciences, engineering, and social sciences. General liberal arts requirements were expected to give breadth to the program. The report was criticized for taking a fragmented approach [38].

In 1981, the C³S submitted its recommendations for master’s level program in computer science. It prescribed that the basic intention of master program is to develop students’ critical and professional thinking and intuition to enable the graduates to take sound professional decisions with awareness of ACM code of ethics. Development of written and oral communication skills, cognizance with pertinent literature in their field of choice, teamwork, and leadership skills were also included among the prescribed goals. However, the committee did not make any specific recommendations to ensure that the curriculum meets the stated objectives. It recommended a list of thirty masters level courses, and classified into following five categories: (i) programming languages (six courses), (ii) operating systems and computer architecture (seven courses, including computer communication networks), (iii) theoretical computer science (four courses), (iv) data and file structures (four courses), and (v) other topics (nine courses). The C³S failed to use this opportunity to make a defining and novel contribution towards curriculum design through these reports of late 1970s and early 1980s. The curriculum committee’s reports of late 70s and early 80s have been later criticized for being reactive rather than proactive [57].

In the last few years, with the emergence of new specialization tracks of human computer interaction and also entertainment computing, sociology, art, philosophy, and psychology related courses have become even more important. Some of the recent programs include many courses from these areas by replacing courses of natural science, management, and electronics [58]. Currently, out of thirteen technical committees of International Federation for Information Processing (IFIP), four committees directly relate to human aspect of computing: (1) Education (working since 1963), (2) Relationship between Computer and Society (established in 1976), (3) Human-Computer Interaction (working since 1989), and the most recent (4) Entertainment Computing (founded in 2002). These committees seek to promote use of models, theories, and methods of social science, human sciences, ethics, psychology, culture, education, and aesthetics.
in both design and evaluation of user orientated computer systems and humanization of system design process.

The AICTE model curricula for computing disciplines [44-45] have not taken cognizance of these developments and place the curricula only in the limited context of natural science, mathematics, physical aspects of engineering, and business. The important and pervasive context of human culture and society has not even been included in the agenda.

**Beginning of Consolidation**

The 1980s was the period of maturation and organized growth of computer science programs in many countries, including India. United Nations Educational, Scientific and Cultural Organization (UNESCO) commission IFIP to propose a modular curriculum especially for developing countries. IFIP submitted its first recommendations in 1984, and revised recommendations in 1994. The IEEE Computer society and ACM jointly specified criteria for the computer science curriculum [59-60]. It mandated a broad based computer science core giving even emphasis on computer theory, algorithms, data structures, programming concepts and languages, and computer elements and architecture. It insisted on inclusion of social implications of computing within the core computer science segment of the program. Mathematics and science were recognized as supporting disciplines, and the criteria sought to provide breadth through humanities, social sciences, and other disciplines. Advanced computer science topics were recommended to be addressed through electives.

The ACM task force in cooperation with Computer society of IEEE [61] started to define the computing discipline and observed that the three paradigms of theory, abstraction, and design are equally important and fundamental to computing. Computer science mainly deals with theory and abstraction, whereas computer engineering deals with abstraction and design. The task force identified two broad area of competency development: (1) discipline oriented thinking, and (2) tool usage, with the first being the primary goal of curriculum. It felt concerned about the neglect of laboratory exercises, team projects, and inter-disciplinary studies. The task force identified three purposes of laboratories in computing courses: (1) demonstrate how principles covered in lectures apply to design, implementation, and testing of software and hardware, (2) emphasize
the use of tools and processes, and (3) introduce experimental methods. Further, the task force provided a novel curriculum design framework by dividing each of these sub-areas into three parts of theory, abstraction, and design. The task force identified nine sub-areas of computing. It observed the need of diversity and well-intentioned experimentation in computing curricula.

The joint ACM/IEEE-CS curriculum task force published its report in 1991. The report [62] represented a unified set of recommendations from two major societies in a variety of academic contexts, including liberal arts, sciences, and engineering. This task force chose to exclude information systems from its agenda, and included all other variants like computer science, computer engineering, computer science and engineering, informatics and other similar program under the single title of computing. It emphasized the importance of breadth, laboratories, social, ethical, and professional issues, theoretical foundations, communications skills, design experience, and teamwork. It strongly advocated the integration of social and professional context of computing along with theory, abstraction, and design into the curriculum. The task force also identified twelve unifying and recurring concepts that are pervasive throughout the discipline.

In 1990s that accreditation agencies of engineering programs in some countries, mainly USA, UK, Australia, Canada, Singapore, and Japan, became explicitly concerned about desired educational outcome. USA’s Accreditation Board for Engineering and Technology (ABET) played a stimulating role in this movement.

Goldweber et al [57] reviewed the previous curriculum related literature incorporating some educational literature. They classified the various pedagogical approaches into six different categories of viewing computing as (i) mathematics, (ii) engineering and design, (iii) art, (iv) science, (v) social science, and (vi) inter-disciplinary. They identified anthropology, applied psychology, computer science, cultural studies, economics, ergonomics, ethics, history, linguistics, management, mathematics, philology, philosophy, semiology, sociology, and politics as relevant disciplines. It criticized the Curriculum’91 for its coverage of social and professional context as an afterthought. This group considered the development of truly inter-disciplinary computing curriculum as the next challenge.
Software Engineering Perspective

Wassermann and Freeman [63] argued that computer science forms only a small portion of necessary education of a software engineer, and software engineering differed from other engineering that have their foundation in natural sciences. This was a novel observation that deserved more attention. This observation may have encouraged the subsequent committees to integrate more content about social and human sciences into mainstream computing courses, as was observed in some of the later recommendations. They considered a software engineer as a generalist, and drew an interesting analogy with a family physician who must have wide range of skills in addition to the core knowledge of medicines and diseases. They posited that a software engineering is an applied computer scientist, and the curriculum content must include problem solving, design, implementation, management, and communication skills. In addition to writing and speaking, the recommended communication skills included willingness to listen to others and sensitivity to the viewpoints and value systems of others. They also recommended the inclusion of accounting or economics or business administration, psychology, industrial engineering practices, and history or political science in the software engineering curriculum.

In his much debated talk called “On the cruelty of really teaching computing science,” Dijkstra emphasized on formalism [64]. He declared software engineering as a self-contradictory doomed discipline. He called for banning the anthropomorphic metaphor in computer science, and insisted that programmer must also give formal proofs for the correctness of their programs. He advised that an introductory programming course should be taught as a formal mathematics course, and students should not be required to test their programs through implementation.

Certainly, mathematics education helps in developing some type of problem solving skills. However, by reducing computer science to formal mathematics, one of the founding fathers of computer science was under-estimating the huge growth of the software industry, and the important role software was to play in everyday life. In this debate, some supported him and others like Hamming, Parnas, Karp, Sherlis and Winograd criticized his ‘extremism’ and reminded that proofs are tedious and fallible, and engineering is not about optimality or perfection, it is reasonableness in terms of reliability, cost, time, and effort.
The serious shortfall of manpower and software crisis provided the necessary enabling conditions for the fast emergence of the ‘doomed discipline’ of software engineering as applied computer science that called for an engineering approach. The Software Engineering Institute (SEI) was founded in 1984 at the Carnegie Mellon University. This institute made significant contributions to the development of educational programs in software engineering. This was the start of some specialized programs in software engineering in USA, and also in Europe [65-66a].

In 1990, SEI presented a model curriculum for undergraduate engineering program in software engineering. As compared to ABET’s accreditation criteria of engineering program, in this curriculum, the humanities and social sciences requirement was increased by reducing electives and mathematics and science components. Further, two ABET categories of engineering science and engineering design were merged into a single category of software engineering sciences and design. None of basic engineering science course was retained in this curriculum. In many ways, this curriculum was a reflection of a twelve year old proposal [63].

A new kind of engineering discipline was finally beginning to get its recognition, which claimed its foundations in the science of artificial constructs, mind, society, and engineering methods rather than material. This is a phenomenon that has been largely ignored by Indian engineering educators, even after so many decades. The curriculum recommendations categorized computing courses into four categories: (1) software analysis, (2) software architectures, (3) computer systems, and (4) software process. This indicated the signs of the beginning of integrated curriculum in computing.

In 1999, SEI-CMU published a report to define the discipline of Software Engineering [67]. The mathematics requirements included mathematical logic and proof systems, discrete mathematical structures, formal systems, combinatorics, and probability and statistics. Topics in numerical methods or calculus were not included. This report also included the computing topics of data structures and algorithms, computer architecture, operating systems, and programming languages. The software product engineering related areas were identified as software requirement, design, coding, testing, and operation and maintenance. Software management
areas encompassed management of process, risks, quality, configuration, process, and acquisition.

Based on a long industry-academia consultative process, SWEBOK [68] provided an excellent document that elaborates upon ten main knowledge areas under the categories of software requirements, software design, software construction, software testing, maintenance, software configuration management, software engineering management, software engineering process, software engineering tools and methods, and software quality. In a very sketchy manner, SWEBOK also elaborates upon the desirable topics of related disciplines of mathematics, computer science, computer engineering, management, project management, quality management, software ergonomics, and systems engineering.

For the first time in its history of nearly forty years, a computing curriculum recommendation made some reference to some education theories. SWEBOK elaborates upon technical competencies that software engineers with four years of experience should have. It identifies ten knowledge areas. Appendix D in their report suggest the desired level of competence as per Bloom’s taxonomy to classify various knowledge areas with reference to ten knowledge areas of software requirements, design, construction, testing, maintenance, configuration management, engineering management, engineering process, tools and methods, and quality. This report is currently undergoing a revision exercise, and some more knowledge areas like software engineering economics are being considered for inclusion.

**Deficient Educational Perspective Till the End of Last Century**

The 1991 report of the ACM/IEEE-CS curriculum task force was seminal as it approached the issue with broader educational objectives and looked at the curriculum as a unified artifact. Leaving the former fragmented approach to curriculum design, this committee tried to create a connected curriculum [69]. However, this as well as all earlier mentioned curriculum recommendations related to computer science and engineering, appear to have over-sighted or ignored the simultaneously growing literature in educational research and curriculum design to theoretically ground their approach and broaden their perspective.
In the absence of such a theoretically grounded perspective of ‘education,’ the recommendations were highly skewed towards content and application with academic and technology orientation for curriculum design. These recommendations did not pay sufficient attention to other aspects of education that are better addressed through incorporation of complementary orientations for curriculum design. These orientations were cognitive process, society centered, and humanistic approach for curriculum design [70]. Scragg et al [71] called for developing insight based curriculum through insight-building activities. They argued that computer science is a fundamentally creative endeavor, and expressed concern at the lack of appropriate vocabulary in computer science curriculum.

Gersting and Young [72] in their paper “Content + Experience = Curriculum” proposed experiential aspect of computer science curriculum to complement the content part, and argued that providing and evaluating experiences is a major responsibility of the faculty. However, even they did not ground their proposal into educational theories. Meanwhile, Carson [73] argued that it is not its application, but effect on thinking that makes sciences relevant. He suggested that teaching within the discipline needs to be subordinated to the central task of teaching about the whole culture. He expressed concern at the substitution of liberal education’s curriculum goals of humanism and citizenship with economic and political goals. Clarke and Reichgelt [74] examined the curriculum of sixty universities and colleges and found that most provided only a list of the courses, and a summary of the objectives.

**Indian Approach**

Recognizing the growth potential, Government of India sponsored Indian Society for Technical Education (ISTE) to propose the first model curriculum in this area. The ISTE interacted with academia, industry, and professional bodies like Computer Society of India (CSI) and Institution of Electronics and Telecommunication Engineers (IETE) and proposed a curriculum in 1987. The group over-sighted most of the important international up-to-date recommendations and manpower requirement projections with respect to computing education, especially with respect to information systems and software engineering. It nearly failed to foresee the tremendous growth of offshore and outsourcing software service industry that already existed even in the
1970s, started to take off in the mid 1980s, and was growing fast in the late 1980s and the early 1990s.

The model curriculum proposed by this committee and published by Rajaraman [75] did not make a mention of this growth or any up-to-date study related to manpower requirement. It only included an outdated report of 1980 on manpower requirement by the Indian Planning Commission. He did not mention any rational reasons or arguments for this retrogressive curriculum that did not find it suitable to put even a single computing course in the first year, and chose to put discrete mathematics in the fourth semester. The committee ignored the already well recognized developments in database management and software engineering. This paper also did not relate itself with the large body of educational research literature. Most surprisingly, none of the ACM or IEEE reports related to curriculum recommendations are included in the reference list. Instead only one UNESCO-IFIP [55] recommendation was included as a reference. However, possibly as an afterthought, for comparison purpose, Denning et al [61] was referred.

Rajaraman [75] distinguished the proposed Indian curriculum from the western model [61] as one with a bias towards electrical engineering. He did not respond well to the real demands and trends of the local or global industry. The growth of undergraduate computing education was slow till the early 1990s. Even in 1993, approximately 3000 students were completing their undergraduate engineering degree in this discipline. However, the growth of Indian education programs in this area has been phenomenal in the subsequent years, and this number has multiplied by more than fifty times in the last last fifteen years. Availability of low-cost desktop computers is the main contributing factor to this growth. It has fuelled the demand for more software, and hence trained manpower, especially in the software sector. The setting up of computational facilities in educational institutes became much cheaper. This phenomenon was largely over-sighted or under-estimated by the curriculum designers. Even today, the curriculum of many universities has not deviated much from the earlier model curriculum. Rajaraman’s paper raised the issue of faculty shortage; the issue is much more serious today. Every year, more than 2,00,000 undergraduate students enter colleges to study computing courses. However, most of the required knowledge related to information systems and software engineering is picked up on the job.
The model curricula designed by AICTE, India [44-45] for undergraduate engineering programs in computer science and engineering and information technology totally ignore the integration and experiential aspects of curriculum design. Most carelessly, the curricula even failed to project basic working definitions of either of the disciplines. With reference to humanities and social studies courses, the committee seems to have totally succumbed to the short sighted economic goals. There is only one language/communication course in the first semester that can qualify as a non-management humanities course. All other humanities courses have been replaced by management courses. It seems that to the curricula have been designed without seriously examining any of the earlier recommendations of any of the educational research literature or even specific curriculum related recommendations of international professional bodies, like the ACM, IEEE, or IFIP.

Section 1.3: Research Approach
Community and culture significantly influence value orientation, perceived needs, and motivation as well as provide the ground for creating shared understanding. All disciplines have their own cultures, and all cultures evolve through cross-cultural exchanges. The computing community has created and documented a sound body of knowledge of software engineering [68]. It is one of finest examples of multi-cultural synthesis of many disciplines especially engineering, computer science, and even social sciences. In the last decade, the disciplines of design and aesthetics are also providing very interesting enrichment opportunities for this body of knowledge. With the very large scale worldwide endeavor on computing or software engineering education, it is now time to leverage education and ‘learning’ related research to create and document a theoretically sound body of knowledge of software developers’ education. Such a body of knowledge should naturally require us to synthesis the evolving disciplines of software engineering and higher education.

The phenomenon of ‘learning’ has been extensively studied by psychologists, educationists, sociologists, philosophers, engineering educators, and even computer scientists working in artificial intelligence and e-learning. Computing educators take very important curricular and educational decisions without referring the rich theories of curriculum design or education. This
oversight is analogous to the misconception that "software engineering = programming" which just requires knowledge of some programming language.

In late 1980’s, engineering methods had to be combined with the elements of computer science to create large scale software systems. Similarly, now with the exponential growth of education in computing disciplines, the scale of the impact of the computing faculty’s decisions is far reaching. The computing student community is no more limited to highly gifted few any more. The scale of computing faculty’s educational responsibilities is continuously expanding. Quality of software development education is an important issue that needs to be urgently addressed. Hence, there is an urgent need to enrich the culture of software development education with the help of educational research. For sustaining this unprecedented expanding scale of computing education, we now need theoretically sound educational frameworks. More so because of severe shortage of experienced faculty, especially in countries like India where this expansion has been exponential, resulting in quality difference between the best and worst programs to be even more than an order of magnitude.

The published research in computing education or software engineering education does not sufficiently leverage this research in education. In the various curriculum reports of 1960s to 1980s by the ACM as well as IEEE, there is no reference to educational models or theories. Even in the 1990s, we find few such attempts. In the absence of such references, it is not surprising that the curriculum committees limited their goal to cataloguing various content areas and describing and sequencing the required courses, resulting in a fragmented curriculum. They did not attempt to argue or propose curriculum models for holistic education of computing professionals.

An attempt of this type may have encouraged the curriculum designers and educators to create an integrated curriculum, as was happening in some other disciplines. Aning et al [76] have observed that in general, engineering faculty is not aware of cognitive science research that has potential to improve engineering pedagogy and mention about recent efforts by NSF to bring together engineering and education faculty. It is not surprising that the computing curriculum
designers not only ignored the pure education research, but also applied educational research such as science education.

Subsequently, the trend started changing, and some authors at annual computing education conferences like the ACM SIGCSE, ACM SIGITE, ASEE-IEEE FIE, etc., started examining, reviewing, and/or using some well established theoretical models and frameworks like Bloom’s taxonomy and Kolb’s experiential learning. However, the papers presented at IEEE CSEE&T show a very poor record of leveraging even such highly popular theories. Interestingly, some learning theories have also been used by the HCI, Information systems, and multimedia communities for guiding their design objectives and processes. A large number of papers in the ACM SIGCSE, ACM SIGITE, IEEE CSEE&T, or IEEE Transaction of Education are like experience reports, and do not make a good attempt to theoretically ground their work in educational research. However, many other streams of higher education, including engineering and science education, have leveraged educational research to enrich their research.

A meta-analysis [77] of computer science education research posits that the majority of the work done in the past has been done by computer scientists reflecting on their own teaching practice. These authors stress that there is a need for more dedicated researchers in computer science education. They observe that in more established educational research, like science education research, the studies carried out are not limited to researchers’ own teaching practices so much as on other teachers’ practices. Not many such studies have been reported in computer science and engineering education. The research method developed and used in this research is an attempt to fill this gap.

The data collection and analysis goals have gone much beyond the boundaries of the courses taught by the researcher. An attempt has been made to integrate the techniques of qualitative as well quantitative research methods to take the advantages of both. Research processes included a wide-ranging survey of published literature in diverse areas of Software development, computer science and IT education, engineering education, professional and higher education, learning theories, instruction design, and human development. Research also included study of a large number of comments written by professional software developers about contemporary issues
related to software development processes, required competencies, endorsements, etc., in various professional forums. More than three hundred professionals of more than sixty organizations from various countries have been consulted and/or surveyed on various issues. More than one thousand undergraduate computing students, and more than one hundred faculty members, have also been surveyed on selected issues.

This dissertation is concerned with understanding and suggesting ways to expand the context of software development education with the help of existing theories on ‘learning’, epistemology, human development, education, and instruction by applying analytical, qualitative, and quantitative methods to investigate the following types of questions:

1. How has software development education evolved, specifically with reference to educational research?

2. What is meant by competent and professionally oriented computing engineers, especially with respect to software engineering? What are the essential attributes? What is the relative importance of these attributes?

3. What is the degree with which the various components of traditional processes of engineering education succeed in creating opportunities for enhancing these competencies? What students think about their educational experiences? What students think works well for them? What processes do professional engineers recommend?

4. What pedagogical practices succeed in developing competencies, and under what circumstances? What comes in the way of implementing these strategies? What kinds of lectures are effective for ‘learning’ in the views of students and faculty? What factors block students from effective ‘learning’? How to overcome these difficulties?

5. What kind of instructional interventions are required? How can the existing education theories/strategies/methodologies be used to educate competent computing engineers? Do we need new theories of ‘learning’ for software development education? If so, what would be main aspects of such a theory of ‘learning’?
In this thesis, we propose a comprehensive framework of pedagogic engagement in computing courses for developing multi-dimensional competencies with respect to the requirements of software development. We have fairly comprehensively examined the published record of major developments and ideas in the history of evolution of computing curriculum since the 1950s. Further, we have identified the distinguishing characteristics of software development. We referred to published literature, and also carried out several exploratory surveys and polls among software developers to understand the profession from their perspectives. We take a position that software development is not an extension of any single discipline.

With respect to the needs of this distinguished profession, we have studied and collated the published recommendations by several accreditation boards, professional bodies, and researchers. We have also carried out several surveys among working professionals to understand their perspectives about the required competencies that must be emphasized by the educational process of software developers.

Based on these studies and surveys, we have identified twelve core competencies for software developers from various approaches, and organize these in the form of a three-tier taxonomy. We then elaborate upon the context and meaning of each of the twelve core competencies in the light of various multi-disciplinary theories and findings, and also our own reflections, empirical results, and interpretations.

During the course of this study, we have studied a large number of theories of education, ‘learning’, intelligence, human development, curriculum design, and thinking. Tables A’1.1a and A’1.1b in Annexure AN1 list some of these important theories and modes. We have selected some of these, and used them for designing our generic framework of pedagogic engagements as well as specific interventions for instructional reform in software development education.

Our proposed framework of pedagogic engagements in software development education includes (i) core activities of software development, (ii) distinguishing characteristics of software development profession, (iii) three-tier taxonomy of twelve core competencies, (iv) five-dimensional ladder of professional and human
development, (v) three-dimensional perspective of the knowledge domain of software development, (vi) two core principles for facilitating deep learning, and (vii) a four-dimensional taxonomy of pedagogic engagements over (v).

Finally, as exemplar case studies, we also elaborate upon some instructional interventions designed and administered by us in some chosen set of computing courses. These interventions are manifestations of some aspects of our proposed framework of pedagogic engagements for software development education. Some new courses have also been developed in the process.

Investigations related to curricular aspects like specific programming languages, methodologies, or formalism are not included within the scope of this work. We believe that the proposed framework is fairly comprehensive, reusable, and robust. It can be used to design many more interventions in software development education. Designers of educational programs for other professions can also use this framework and methodology.

**Section 1.4: Thesis Layout**

The first chapter of the thesis gives an overview of the motivation, objective, background, research method, and results of the reported work. In addition, we also discuss the evolution of computing curriculum in the last five decades.

In the second chapter, the required core competencies for software developers are explored with the help of published recommendations of accreditation agencies, professional societies, and published research. Fresh survey has also been carried out for this investigation. These competencies are then consolidated into a three-dimensional taxonomy. More literature is explored to consolidate the competency requirements of the software services and software product companies.

The third chapter analyzes the distinguishing features and multidimensional aspects of software development with a view to further analyze the required competencies. In this process, a large number of software professionals were consulted on various issues related to software development and required educational inputs. The three-dimensional taxonomy of competencies
proposed in the second chapter is distilled and revised into a three-tier taxonomy of twelve competencies.

In the fourth to sixth chapters, we discuss the meaning of the identified twelve competencies in the context of software development work. The basic competencies are discussed in fourth chapter. The competency driver-habits of mind are elaborated in fifth chapter and competency conditioning attitudes and perceptions are discussed in sixth chapter. We draw upon multi-disciplinary published literature and empirical studies in the process. Each of these chapters deals with a different category of competencies as per our taxonomy.

The seventh chapter gives an overview of various quantitative and qualitative surveys among computing students, software developers, and faculty of engineering institutes. We conducted these surveys to empirically investigate the phenomenon of ‘learning’ in computing/engineering disciplines. In this chapter, we essentially discuss the rationale for student-centric active learning.

The eighth chapter gives the most significant theoretical contribution in this work. We consolidate all our earlier findings discussed in the earlier chapters with the results of carefully chosen classical and contemporary ‘learning’ theories. These theories have been chosen with respect to their applicability for software development education. We propose a unified framework of pedagogic engagements in software development education. This framework focuses on development of required core competencies for software development as consolidated in the third chapter and discussed in the fourth, fifth, and sixth chapters.

Some aspects of this framework are manifested in some instructional interventions discussed in the ninth chapter. The tenth chapter provides a summary, and suggests future scope of research.