DESIGN OF INTERVENTIONS
FOR
INSTRUCTIONAL REFORM
IN
SOFTWARE DEVELIOPMENT EDUCATION
FOR
COMPETENCY ENHANCEMENT

1. Introduction

For the last four decades the demand of software developers has been increasing at an accelerated rate. 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. 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 exponentially growing Indian software industry.

To meet this challenge, the *Indian academic institutions have been able to expand fast and satisfy the industry need of software professionals quantitatively.* However, *the quality of 'most of' the professionals they generate is below the desired industry standard.* 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

[1][2]. Most of the software houses spend around six months to one year in their *post-induction, in-house software development education and training*. It clearly indicates that there is a significant gap in what technical education the academic institutions give to their software graduates, and what technical competencies the industry expects in them. ***A competence mismatch exists between academic technical offering and software industry employability.***

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***. For this purpose, it first attempts to identify critical and necessary competencies for software developers from various approaches, consolidates them in the form of a three dimensional taxonomy, and later elaborates following five main types of interventions for reforming the instruction in computing education.

1. Inquiry teaching
2. Project-inclusive teaching
3. Multilevel infusion
4. Integrative courses
5. Group and Community learning

All these interventions have been administered in some chosen set of computing courses. 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.

Research processes included a wide-ranging survey of published literature in diverse areas of Software development, Computer Science & IT Education, Engineering education, Professional and higher education, Learning Theories, Instruction Design, and Human Development. However, only a small subset of the most relevant references has been included in the synopsis. More work is included in the thesis. 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 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.

2. Identification of Pivotal & Critical Competencies for Software Engineers

Study Report on Critical Competencies for Engineers – with specific reference to Software Engineering

We first explore various studies related with the core competencies required for general engineering graduates and come up with the set of *general engineering competencies* normally accepted among the researchers [3]. With this set of competencies as a starting point, we do an extensive survey among software engineering practitioners, mainly to find out which *subset of engineering competencies are more important for the software engineering graduates.* After analyzing the survey data [4], it was found that the *identified critical competencies for general engineering graduates were also required by software engineers, but there were major difference in their order of importance*.

In this competency subset, called *critical engineering competencies for software*, *problem solving* was identified as the pivotal competence for software graduates. *Analysis/methodological skills* and b*asic engineering proficiency* were identified as the two highest rated competencies. *Development know-how, teamwork skills, English language skills, presentation skills, practical engineering experience, leadership skills,* and *communication skills* are seven other competencies considered critical by the industry. Thus, a total of *nine competencies* are categorized as critical for software engineers in the context of the Indian IT industry.

Necessary Competencies as Educational Outcomes for Software Engineers: Study Report on Accreditation Boards and Professional Societies' Approaches

Subsequently, an extensive study was done of *accreditation boards* of countries that are major players in IT industry, viz., USA, UK, Japan, Australia, and Singapore [5]. It was found that they

have transformed their *accreditation process from resource-based approach to outcome-based approach,* and they have stressed *outcome in terms of competencies*. The *necessary competencies* common among these boards are: *ability to apply knowledge, design skills, problem solving skills, technical competence, ability to work in multi-disciplinary teams, sensitivity to ethical and professional issues, and readiness to lifelong learning.* The major *international professional societies* like the IEEE and ACM, and Indian National Academy of Engineers have identified a few additional competencies, such as *system level perspective, analytical skills, critical and creative thinking,* as *necessary* as well. The importance of some of these competencies was not examined in the earlier ranking studies [3][4].

Different competencies do not have sharp boundaries to differentiate one from another, nor does there exist unanimity in naming them. As these required critical competencies are to be achieved through the academic process, to *visualize the acquisition of these competencies,* the author approached them from two distinct perspectives: *levels of cognition* and *dimensions of learning*.

Acquisition of Competencies and Level of Cognition

Bloom's taxonomy [6], which is well accepted among researchers [7][8], classifies six major levels of cognition, in a hierarchical order, that can be achieved through an education process. The six cognitive levels, beginning with simplest, with increase in level of complexity, are: *Knowledge, Comprehension, Application, Analysis, Synthesis, and Evaluation*.

The 'Knowledge' cognitive level exhibits recall of previously learned materials, such as facts, terms, basic concepts, and answers. 'Comprehension' is identified as understanding of facts and ideas, by comparing, translating, interpreting, and organizing them. 'Application' is about solving problems by applying acquired knowledge, techniques, rules, and abstractions in an unprompted way.

'Analysis' represents the act of examining and decomposing information into parts by identifying reasons or causes; making inferences and finding evidence to support generalizations. 'Synthesis' aims at integrating information in different ways by composing elements in new

patterns, or proposing alternative composition. 'Evaluation' is about presenting and defending opinions by making judgments about information, validity of ideas, or quality of work based on a set of criteria.

While the first three levels are part of basic cognition activities, the next three levels are considered to represent higher-level cognitive activities that require and develop mental faculties of *critical thinking, creativity, and innovative problem solving.*

Using Bloom's taxonomy as the basis for an empirical investigation, a study [9][10] was done which indicated that the present academic process of teaching-learning-evaluation process, *mostly engages students at the lower three levels* of cognition only. The Indian Accreditation Board, in complete variance with other international accreditation boards, follows the *traditional resource-based approach* for accreditation, and gives stress only on *content-based curricula* for education. It neither defines competencies as learning outcome, nor demands re-orientation of the academic process for *outcome-based approach*. The present academic process does not engage students in the three higher levels of cognition, resulting in *deficiency of many critical competencies* among them [10].

This study indicates that there is an urgent *need to transform the academic process from content-based curricula to process-based curricula, and to upgrade the students' engagement to the higher levels of cognition. Only then is it possible to develop the required critical competencies among students.*

Acquisition of Competencies and its relation with the Dimensions of Learning

Dimensions of learning [11] is a comprehensive model for learning and the learning process. It groups the various learning outcomes in the following five dimensions:

Dimension 1: Attitudes and Perceptions,
Dimension 2: Acquire and Integrate Knowledge,
Dimension 3: Extend and Refine Knowledge,

Dimension 4: Use Knowledge Meaningfully, and
Dimension 5: Productive Habits of Mind.

As per this model, *all learning takes place in the backdrop of the learner's attitudes and perceptions (Dimension 1), and the usage of productive habits of mind (Dimension 5).* This means that when a learner extends and refines knowledge *(Dimension 3),* he continues to acquire and integrate knowledge *(Dimension 1),* and when he uses knowledge meaningfully *(Dimension 4)* he continues to acquire and extend his knowledge. In other words, Dimensions 2, 3, and 4 represent different aspects of knowledge learning in three hierarchical levels. As there are no orthogonal relations among them, in this discourse, they are merged into one. The new merged dimension can be viewed as having *three internal hierarchical sub-levels*. Further, we also include *values* as integral to the first dimension. Consequently, we assert that, in essence, there are only *three orthogonal, non-overlapping dimensions of learning.* They are:

Dimension 1: Attitudes and Perceptions, and Values,
Dimension 2: Productive Habits of Mind, and
Dimension 3: Acquisition, Integration, Extension, and Meaningful Usage of Knowledge.

Learners' *attitudes, perceptions, and values* about the purpose of learning, as well as roles of teacher, self, and peers determine their motivation, and very significantly influence depth and performance of their learning. *Productive habits of mind* - critical thinking, creative thinking, and self regulation facilitate their learning process.

The set of critical competencies (for software engineers) that were studied and identified (starting with the set of competencies for general engineers) earlier were mapped in these three dimensions of learning [5]. It was *surprising to find that most of the required critical competencies fall under Dimension 3, the other two dimensions were not sufficiently addressed*. Even in Dimension 3, only the acquisition and integration of knowledge is addressed, the *higher two sub-levels*, extension and meaningful usage of knowledge, are also not sufficiently addressed.

In other words, the commonly prevailing academic process does not directly help the *students to develop those competencies that are outcomes of learning in Dimensions 1 and 2, and also that of the two higher levels of Dimension 3!*

The Competency Mismatch: The Uncovered Required Critical Competencies

The above conclusion prompted us to conduct an *extensive survey*, based on the taxonomy of the proposed three dimensions of learning (encompassing thirty five types of competencies), *to find out those critical competencies that represent the uncovered dimensions of learning.* The survey covered - (i) large companies engaged in software services, (ii) large and mid-size companies engaged in product development, and (iii) small size companies involved in product development. The findings of the survey are most revealing, and are as follows:

For the *Software Services Industry*, the ranked list of top competencies recommended were: *(i) ability to work in team, (ii) abilities related to perseverance, commitment and hard work, and (iii) listening skills.* Interestingly, all these competencies can be achieved as *outcomes of learning Dimension 1*, usually not covered by the present academic process.

For large and mid-size IT Product Development Industry, the required pivotal/critical competencies were: *(i) ability to work in teams, (ii) ability to apply knowledge, (iii) abilities related to perseverance, commitment and hard work, (iv) accountability and responsibility, (v) analytical skills, (vi) problem solving skills, and (vii) research skills*. Here again, all these competencies can be achieved as *outcomes of learning Dimensions 1 and 2*, that are usually not covered by present academic process.

For small size IT Product Development Industry, the required pivotal/critical competencies comprise of all that are required for a large product company (with some minor change in their ranks), along with a few *additional* critical competencies: *(i) attention to detail, (ii) readiness for lifelong learning, (iii) quality consciousness and pursuit of excellence*.

It clearly *indicates the gap which needs to be filled.* The goal must be to address the ***learning of Dimensions 1 and 2, and also the higher level of Dimension 3***. It is also imperative to understand that these learning outcomes can be achieved mostly through *changes in academic process*, and also *inclusion of a few additional courses*, so that all required changes in the academic process can be facilitated.

3. Distinguishing Features of Software Development

As our study and investigations showed, most of competencies required for general engineering are also required for *software engineering*, but the latter *does require additional competencies* that are critical for their profession. This prompted us to *investigate the distinguishing features of software development*, so that we would be able to propose what *types of instructional reforms would result in addressing the competency mismatch.* It would also help formulate what *changes in curricula* (deletion / modification / addition of courses) would be necessary to facilitate the changes.

Programming as an Art to Software Engineering

In the initial days, programs were mostly written by individuals, and the programming tasks were handled more like an art. A set of mathematical and algorithmic courses formed the major portion of computing education.

As the programs started handling more complex tasks of various applications, their size increased. Large programs need modifications and enhancements throughout the period of their active use. The development and maintenance of large programs is no more an individual effort, but a team work which requires disciplined engineering processes with systematic documentation of activities of analysis, design, coding, testing, deployment, and maintenance. This is a qualitative jump in the software development process, and by the mid 80s various courses covering different aspects of Software Engineering became an integral part of computing education. Software development evolved out of a mathematical art to an engineering activity for

handling complex tasks. Similar to other engineering activities, *problem solving and technical competence would be the key critical competencies required for software engineering.*

Software as Integral Part of Business, and Need for Comprehension for Software Maintenance

The engineering approach to software development, coupled with continuous exponential advancement in computer hardware technology, brought higher level confidence among its users, and they started to look at this integrated field as 'information technology'. Software has now become an integral part of business processes of a large number of organizations. Today, any development of business/application software must presume that the developed software would go through repeated changes. To satisfy their clients by the quality of their service, the software professionals must be capable of comprehending the existing client software, and then perform the required modifications/enhancements as per their evolving requirements.

The ability to comprehend an existing software is one of the most basic and important ability required by the software industry. This is presently not at all addressed by computing educational curricula. Therefore, *software developers need to develop the ability to comprehend the software developed by others, and also write software that can be easily comprehended by other developers***.** Increasing dependence on large amounts of Free and Open Source Software (FOSS) makes it even more crucial.

Software Design: Social Sensitivity and Empathy

Software development in many ways is all about people – users, customers, developers, managers [12]. Designs that are suitable for a context are not necessarily as suitable for other contexts. *Software developers need to understand users' requirements from multiple perspectives.* The software developers' main concern has been gradually shifting from making "inexpensive" software to "quality" software to "appropriate" software. Thus, *social sensitivity and empathy* are the critical competencies required for development of appropriate and quality software. Development of social sensitivity is not to be taken at the periphery, but ***at the core of the software developers' education program.***

Software Engineering: Process Centric System Development and Maintenance

The term 'engineering' in the context of software development refers to having a ***systems approach*** to problem solving, and also following a disciplined ***process centric/oriented approach*** for assuring the quality of the deliverable software system, and their ***maintenance and evolution throughout their life cycle***. The approach of traditional engineering education mostly does not pay much attention to user interaction and evolution, and hence is not suitable for software. The various aspects of the system development process mainly deals with human processes and engineering management processes. Consequently, it is imperative for software developers to develop the competencies of process centric system approach and continuous evolution.

Software Contract: Project Scoping and Estimation

Usually the software projects are based on **contracts** between the clients and vendors of software services. Earlier, in most of the on-shore projects, the clients were charged on the basis of manpower engagements. The software industry is now ready to take up software contract on fixed cost basis. **Project scoping and estimation are the two most challenging tasks of the software development process, which are not adequately covered by current computing educational curricula.**

Software Development: Learning New Domain and Knowledge Structuring

The software development processes essentially try to map the application domain requirements to programming constructs. Domain training and even certifications have become common part of continuous training programs of software developers. Armour [13][14] has viewed software development as a ***learning activity,*** rather than a production activity, and advocated that software developers need more training in ***learning, and knowledge structuring mechanisms*** rather than in software itself. The software development education program needs ***to expose the students to diverse types of domains, and also nurture the ability to learn nuances of newer domains***.

Software Development Process for Ill-defined Problems

As application domain requirements are embedded in real experiences, it is usually very difficult to map and concisely describe it as a software problem. Therefore, real life software problems are ***mostly ill-defined problems*.** Subject centric teaching is effective for developing competence in solving well-defined problems using well-structured procedures, but to solve ill-defined problems one has to follow ill-structured procedures. Often multiple iterations and representations are required to define the problems. Further, usually there are no unique best solutions, only multiple acceptable solutions to partially solve the real life problems. This characteristic of software problems, and the development process, also contributes to make it a multi-dimensional activity. ***Agile methods*** are increasingly being accepted to develop software primarily to address this aspect of software problems. *Multi-perspective thinking, critical thinking, creative thinking, and innovative problem solving significantly contribute in transforming complex ill-defined problems into simpler well-defined problems.*

Software Development: Whole-Brain Activity

Like all long-term system development activities, software development also requires several cycles of left- and right-brain activities – abstraction and concretization, logic and intuition, critical thinking and creativity, reflection and experimentation, micro-scoping and macro-scoping, as well as reduction and holistic thinking. *Diverse types of left- as well as right-brain thinking skills are integrated to create good software.* ***Software development is a whole-brain activity.***

Thus, to develop good software professionals, the computing education must train professionals to understand a problem, looking at all ***its multi-dimensional aspects with multi-perspective approaches.***

In addition, to solve any ill-defined problem, it is imperative ***to develop diverse types of thinking skills, comprising whole-brain activity, among software professionals.*** **Problem centric learning methods has been found to be more effective for developing the competence to solve ill-defined problems [15].**

Critical Competencies for Software Development and Dimensions of Learning

Identification of these distinguishing features of software in this section has helped in further expanding and elaborating the list of thirty-five competencies discussed in Section 2 of the synopsis. After going through various theoretical and empirical studies, various empirical surveys, and also based on an in-depth analysis of distinguishing characteristics of software development, the thesis has enumerated and summarized all the required critical competencies for software development. This complete set of required competencies will be discussed in more detail in the thesis.

With an objective of developing and strengthening these required critical competencies through formal education, these competencies have been categorized into three dimensions of learning. To overcome the competency deficiency, appropriate interventions for instructional reform in computing education can be visualized, and experimented.

The most important competencies required for software developers within the three dimensions are consolidated below.

Dimension 1: Attitudes, Perceptions, and Values

Attitudes, perceptions, and values affect a professional's ability to practice. The most important element of education should be to develop required attitudes, perceptions, and values. The following list enumerates the required attitudes, perceptions, and values for software developers:

1. Urge to create/ improve things and open-mindedness.
2. System-level perspective: Inclination for reuse and synthesis by integration. Ability to understand and also build upon other's work. Ability to work such that others can easily understand and build upon.
3. Accountability and responsibility, strength of conviction, and self-regulation.
4. Curiosity with humility: self-learning, ability to develop good understanding of domains' vocabulary, semantics, and thinking processes, faith in reason, and review.

5. Ability to accommodate himself to others.
6. Ability to see the self as bound to all humans with ties of recognition and concern: sensitivity towards global, societal, environmental, moral, ethical and professional issues, and sustainability.

Dimension 2: Productive habits of mind

The most effective professionals develop powerful habits of mind that enable them to think critically and creatively. The mental habits that are important for software developers are listed below:

1. Problem solving: ability to synthesize the competencies of the first dimension with those of the third dimension in the context of new settings and complex problems. Ability to convert ill-defined problematic situations into software solvable problem. Project scoping and estimation.
2. Attention to details.
3. Abstraction and transition between levels of abstraction.
4. Algorithmic and structured thinking.
5. Critical and reflective thinking.
6. Creativity and innovation

Dimension 3: Meaningful usage, extension, and acquisition of knowledge:

Motivated by their values, attitudes, and perceptions, professionals use their productive habits of mind to acquire and integrate knowledge. Attitudes, perceptions, and productive habits help them to extend, refine, and use knowledge for meaningful tasks. The following competencies are important with reference to software development activities:

1. Technical and domain competence.
2. Communication skills.
3. Analytical, design, and debugging skills.
4. Decision making skills.
5. Project planning and management.

4. The Phenomenon of Learning

Limitations of subject-intensive teacher-centric instruction

In the traditional form of engineering education [16], based on teacher-centered one to many learning, the teacher is seen as the source of information. Success in learning is often seen as the reproduction or direct application of what the teacher has taught. Such instruction, in which abstraction precedes the instantiation and concretization, helps students in *developing skills in deductive reasoning* and succeeds in *creating a knowledge-base as an inventory of concepts. It also trains students in linear thinking.*

In our survey, working engineers and managers in Indian and multi-national IT companies rated lectures as one of the most ineffective teaching method [17]. Our survey about the value orientation of undergraduate computing students showed that most responding students felt that most of their peers lacked the values of self-direction, benevolence, and universalism. Since, our required competency set of critical competencies also incorporates these values; the development of these values has to be addressed by the computing education process. The traditional text book centric approach to teaching fails to stimulate the learners sufficiently to develop these, as well as many other required competencies, listed in our three dimensional taxonomy .

Alternate learning approaches

A rich body of literature regarding classical, as well as contemporary, learning theories exists, and offers the way out to address the competency mismatch and learning deficiencies [6][11][15][16][18][19][20]. We reflected upon this body of evolving literature to examine its suitability for computing education [21][22][23][24]. This reflective review facilitated the identification of the core instructional principles that can help in designing the appropriate interventions for instructional reform in computing education, with special reference to software development. **Students' active engagement in inquiry, semester-long projects, collaborative work, and an integrated curriculum have been identified as** such core principles. The thesis will provide an overview of this review and analysis.

5. Interventions for Instructional Reform

The visualized interventions for instructional reform in computing education with special reference to software development range from *modifying teaching style*, *evaluation method*, and *course content* of some regular computing courses. In addition, some *new courses* have also been conceived and created.

In the literature, various approaches in instructional reform have been experimented and studied to achieve the desired learning outcomes. To overcome the competency deficiencies enumerated earlier, the thesis considered the following two approaches for designing interventions for instructional reform in computing education.

5A. Student-Centric Active Learning Approach

Many studies [25] have shown that best learning is achieved when the learner is engaged in doing and explaining their acts of tasks. Other studies [26] concluded that the knowledge to understand, frame, and solve problems evolves during the problem solving process itself. As part of the study, the computing students' learning experiences in terms of Bloom's taxonomy was evaluated, and the findings validate the constructivists belief that *knowledge is a learner's activity, and learning is an individual's struggle with the issues within a domain* [9][10].

Active learning requires learners' *engagement in various activities relevant to course content*, rather than mainly depending upon content's acquisition from external sources. *Student-centric active learning* treats students' active engagement as the primary source of learning even at the beginning of every learning cycle. Further, it offers the freedom of different kind of activities for different students. Depending upon their prior experience and interests, students can choose or even define their activities. This helps in enhancing their feeling of *autonomy*, and hence, their *motivation*, and also *happiness*.

Cognitive Dissonance and Problem Centric Approach

Many of the attitudes, perceptions, and values (components of Learning Dimension 1) considered being important for software development activities cannot be assumed to naturally develop as a natural result of traditional computing education. The Cognitive Dissonance Theory [27] postulates the following:

- Humans are sensitive to inconsistencies between actions and beliefs,
- Recognition of an inconsistency would result into cognitive dissonance, and would motivate individual to resolve the dissonance, and
- Dissonance would be resolved in one of three ways: change in beliefs, change actions, *or change perception of actions*.

Based on the cognitive dissonance theory, it has been shown by *Structured Design for Attitudinal Instructions* [28] that instruction can be designed to create short term dissonance. This dissonance facilitates the learners to first recognize the need to change the attitude, and then they should be guided through progressive changes to resolve the dissonance. In a similar approach, Kort et al [29] view learning as a spiral process of construction and de-construction (of misconceptions) phases through positive as well as negative emotions.

The thesis has proposed to transform computing education by creating dissonance for a short period, and then the teacher should guide them progressively to resolve the dissonance to higher levels of learning. This approach would overcome the competency deficiencies of Learning Dimension 1.

The traditional teacher-centric lectures do not create any dissonance among learners. The most natural way to *create dissonance* would be to lead learners *through problem-centric approach*. The problem-centric approach, when applied to teaching, can result into *two* types of *interventions for instructional reform*: ***Inquiry Teaching***, and ***Project Inclusive Teaching***.

Inquiry Teaching

Importance of inquiry in the learning process

Bruner and other constructivists [30][31] recommend that instruction should allow the learner to discover principles for themselves through active dialogue. Instead of aiming to teach some general rules and theories, inquiry teaching aims to teach how to discover the general rules and theories.

Empirical study

In an empirical study, covering senior students, to recollect the best lectures that gave the respondents most effective learning experience, it was found to be either in the form of active learning, or collaborative learning, involving problem solving, group work, discussions, critique, and so on. Similar exercise with teachers suggested that most of the faculty members consider those lectures as their best in which they are able to put the students into one or the other kind of activities like problem solving, seeking clarifications, design, group work, and so on [32].

It was also found that many students feel that unexpected results, engagement in reasoning, and even unsatisfactory answers from authority have made significant contribution to forward their learning. Inquiry teaching is particularly effective in exposing learners' misconceptions. *It is particularly suited for developing curiosity, self learning, analytical skills, humbleness, inductive and lateral thinking skills, and hence in facilitating deep learning.*

Inquiry Teaching revolves around questions. This will require the teachers and also the students to ask many more questions in their classes. The lecture delivery process of some courses, viz., Data Structures, Computer Graphics, and Orientation to Engineering, were transformed through Inquiry Teaching. The details of these attempts are discussed in the thesis.

Challenges for Inquiry Teaching

The success of Inquiry Teaching mainly depends upon students' active participation in the inquiry process. It requires and also furthers the *transformation of students' perception about their own role in the process of learning from an information receiver to an active contributor to*

meaning making. However, for many students, their old habits formed through prior long experiences with exposition based teaching, can hinder their enthusiastic participation as an active learner in the classroom, especially in large and unresponsive classes. Such students find inquiry teaching to be unsatisfactory and miss the opportunity of not only deep but also surface learning. Therefore, it is most important to sensitize students to this method of learning in their early courses. *For maximizing the benefits of inquiry teaching, students need to 'learn to learn' through this method.*

Developing Habit for Inquiry Learning through Puzzle Solving

Puzzle solving task is another example of inquiry learning. Puzzle solving activity demands that the teachers start their sessions with problems rather than concept. Many software companies include puzzle solving in their selection criteria of new software developers. Puzzle solving sharpens critical thinking and problem solving ability, and offers a higher potential to develop many of the multifaceted thinking skills. Therefore, an experiment is underway *to redesign the delivery strategy of the first computing course by starting it with puzzle solving activity, even before the introduction of the basic syntax of any programming language. A large number of* students have reported benefits along all the three dimensions of our taxonomy in terms of enhancements in logical, creative, multi-perspective, and out of box thinking, attention, focus, concentration, patience, comprehension, urge for creation etc.

Project Inclusive Teaching

Importance of Semester long project

According to our survey, on effectiveness of eight types of teaching methods done with working engineers and managers in Indian and multi-national IT companies, *group projects, individual projects, and practical training were rated as more effective* teaching methods than industrial training/internship, lecture, seminars, and written studies.

Projects offer the opportunities to engage the students in multifaceted academic engagements like *synthesis, application, making judgements, and multi-perspective thinking.* Students rated that all these experiences make maximum/excellent contribution in developing their

competencies. *A semester-long project serves a larger academic purpose by giving highly valuable process related experience of team formation, collaboration, problem formulation, literature survey, planning, analysis, design, implementation, testing, deployment, and documentation.* Often projects demand that the students also learn some additional concepts and theories. In order to complete the project, they usually also have to integrate the learning of many earlier courses and also some application domain. Further, evaluation of project work requires students to write project reports, present and demonstrate their work, and also face viva-voce.

Semester-long project experience helps in developing multidimensional competencies in all the dimensions. With respect to the first dimension's critical competencies, it is particularly helpful for developing the *urge to create/improve things, self learning, system perspective, accountability and responsibility.* Further, team projects also develop their *ability to accommodate themselves to others, and some projects also have the potential of developing the ability to see the self as bound to other humans with ties of recognition and concern. Project experience helps in developing all of the second dimension's competencies* of multifaceted thinking habits.

Finally, as regards competencies of the third dimension, projects are particularly effective for enhancing *technical and domain competence, analytical and debugging skills, design skills, decision making skills, and communication skills.* Hence semester-long projects have the potential to facilitate deeper learning in many significant ways.

Project-Inclusive Regular Courses

Usually project work is not included as part of the regular courses. This limits the effect of the courses in terms of developing their competencies. The constructivist paradigm of *project-inclusive teaching* challenges this assumption. Rather than viewing a *project* as the culmination activity, *it is viewed as the instrument of creating richer context for learning the subject matter*. However, as the traditional textbooks are normally not written with this objective in mind, the *project inclusive course teaching requires a change in delivery strategy*. *Two different models,*

viz., project-centric evolutionary instruction and project-oriented instruction, have been proposed for achieving this goal.

i. *Project-Centric Evolutionary Teaching*

In project-centric teaching, the student projects are planned in such a way that most of the planned topics of the subject can be used in all projects. At the beginning of the course, the students are required to formulate the initial project problem. Project scope is *incrementally enhanced and made more sophisticated later* in the semester, essentially to create the context for the forthcoming concepts and topics of the subject matter. *The project scope is refined before introducing every main topic.*

In addition to offering all the benefits of project-inclusive teaching, project-centric teaching offers additional advantages of giving the experience of evolutionary software development. *It makes students more open-minded for software evolution, engages them in reflection, and gives them experience of developing software for future enhancements.* During the course of this research, project-centric evolutionary teaching was evolved for Enterprise Application Development [33]. Recently, it has been further expanded to many other courses. The thesis discusses the details.

ii. *Project-Oriented Teaching*

In many courses, it is not feasible to plan semester-long projects using all the main topics of a given subject. In such cases, it is better to plan projects on different topics. However, this scheme imposes some challenges regarding synchronising the project activity with the content delivery in the class. In order to partially overcome this limitation, a *two-level content delivery scheme* has been tried out in some computing courses like Microprocessors and Microcontrollers, Operating Systems, Computer Networks, and Complier Design. As per this scheme, the entire course is delivered in two phases. In the first phase, lasting approximately two to three weeks, an extended introduction of the course gives a comprehensive macroscopic view of the entire subject matter. In the second phase, the topics are sequentially picked up for in-depth classroom discussion.

5B. Integrated Approach for Cognitive Flexibility

As the domains of employment for fresh software graduates are so widely spread, it is important that they view the curricula as an integrated system, and also learn to integrate concepts across the domain boundaries. In order to prepare themselves *industry-ready*, they must acquire ***cognitive flexibility*** [34], the ability to apply the learned concept to different situation. However, in the teaching of most of the courses of mathematics, basic sciences, and general engineering disciplines, there is hardly any horizontal reference to other courses. The course contents are designed in a *vertically intensive way*, and the education process succeeds in teaching rules and theories, but fails to develop an integrated conceptual understanding. The *learning* though intensive, is completely *fragmented* and *inflexible*.

In computing, many concepts, technologies, and practices are highly *pervasive*, and hence, they must be understood in *integrated* way. Hence, in order to provide an integrated approach to computing education, and to impart cognitive flexibility with special reference to software development, the thesis has visualized *three types of interventions for instructional reform*, viz., ***Multilevel Infusion*** of key technologies and professional practices, ***Integrative Courses***, and ***Group and Community Learning***.

Multilevel Infusion

As computing is a continuously evolving field, there is always a pressure to introduce courses to address contemporary trends. Often introduction of new developments is carried out by either replacing some classical and core computing courses, or as electives, resulting in fragmented cognition among students.

Contemporary *technologies like Web, Mobile, Multimedia and Security* and professional *practices like Estimation, Design, Open Source Usage, and Defect Tracking* are highly pervasive. *The thesis proposes a model for enriching the existing courses through multi-level infusion of selected elements of contemporary technologies, and professional practices.* This infusion can be incrementally carried out across most core courses. Consequently, even before doing the dedicated courses on these topics, the students are fairly well exposed to all these areas

as problem domains of junior level courses. In addition to increasing *cognitive flexibility*, multi-level infusion of selected topics enhances *open-mindedness and integrative thinking*. It also nurtures the habit of *reflection.*

A new graphic notation for modeling the software problems has been developed and administered in some courses. Faculty and students have found this concept mapping technique to be very useful. In order to develop estimation skills, the programming process related data as adapted from the Personal Software Process (PSP) [35] has been administered in many laboratory courses. Majority of the students have reported benefits in terms of programming efficiency enhancement, defect rate reduction, activity record and reflection. Many of them have also reported benefits in terms of improvement in estimation and planning skills. The standard PSP format has been modified to gain even higher gains in estimation. In order to infuse debugging experience, taxonomy of software bugs has been prepared with an objective of designing debugging related assignments in various computing courses [36]. The details of multi-level infusion of various technologies and professional practices are discussed in the thesis.

Integrative courses

While multi-level infusion helps to integrate the computing curriculum, some integrative capstone courses can further strengthen the unification of some important computing concepts, and also integrate the computing concepts with other disciplines. Attempts have been made to design some courses to bring integration of otherwise widely spread computing concepts. The integrative courses help in strengthening *nonlinear, integrative and systems thinking and flexible learning*. They also have the potential to integrate, and hence, consolidate the already learnt material and also provide a stronger foundation for further studies. The thesis elaborates upon some experiences in designing and delivering some courses that try to contextualise and integrate computing with selected elements of human sciences. These human science concepts have been carefully chosen based on their relevance and importance for enhancing some critical competencies.

Courses for Integration with selected elements of Human Sciences

In modern times, the integration of seemingly disconnected disciplines of human knowledge offers very exciting research and learning opportunities. Armour [13][14] viewed software development as a learning activity rather than a production activity, and advocated that software developers need more training in learning, and knowledge structuring mechanisms rather than in software itself. In a course of *'Theory of Knowledge, Learning, and Research'* at Jaypee Institute of Information Technology, the students are exposed to a spectrum of learning theories. All these theories are also used for reflecting about learning in general, and also with specific reference to software development. It enhances their understanding of their own learning process and also helps them identify their misconception about learning. These theories and models help sharpen students' *questioning skill, critical thinking, and reflective thinking*. It helps the students to become *better learners*. Understanding of the diversity of learning styles also prepares them with enhanced *ability of self-regulation, ability to accommodate themselves to others and also makes better prepared for understanding of domains' thinking processes.* Further, there is strong tradition of formally teaching 'Research Methodology' in many non engineering disciplines. However, such content is not usually offered in engineering disciplines. The research in the field of software development combines the research methods of engineering as well as social sciences. This is included in this course, and it also helps in enhancing the *critical and integrative thinking, analytical skills, and also self-learning.*

The other multi-disciplinary integrative course *'Human Aspects for Information Technology'* at Jaypee Institute of Information Technology, aims to explore the humanistic grounds for information technology and software development. The students evaluate the information technology and also the activity of software development with respect to multidimensional aspects of social welfare and professional decision making. Various activities of this course help the students to understand the meaning and importance of professional *responsibility*. The other part of this course is about *creativity and innovative problems solving.*

Group and Community learning

Learning in a collaborative environment is a process that could be subject of two different perspectives: individual effort and social sharing of knowledge [37]. However, sharing of knowledge is useful only if students are engaged in collaborative activities**.** Engagement in collaborative activities causes individuals to master something that they could not do before the collaboration [38]. Two models have been tried out – pair programming prepares the students for team work and benevolence while learning pyramid is aimed for preparing them for larger organizational concerns, universalism, and responsible citizenship of larger communities. Group and community learning helps in enhancing students' *ability to accommodate themselves to others, to see themselves as bound to all humans with ties of recognition and concern, sense of accountability and responsibility, as well as multi-perspective and creative thinking.*

Pair Programming – A Novel Approach

In the popular form of pair programming, the pair sits on the same computer all the time and takes turns to write the code. This technique has a major disadvantage when one of the students in a pair is dominant, or a very high-scoring student, as compared to the other student. The stronger student does most of the work, diminishing the learning experience for the weaker partner. In this thesis, a *novel approach for pair programming* is discussed [39]. Each laboratory exercise for the first programming course was designed in two parts. The first part required two students to work independently. After solving their individual task, they were given a combined task. The difficulty of the combined task was higher compared to the individual tasks, and it was designed such that its solution required the students to combine the concepts, logic, and code of both the individual solutions. The students reported benefits like enhancement of *problem solving skills, efficiency, quality, trust, and teamwork skills*. Further, it provides experience in *reading and understanding foreign code, writing code for others' understanding, and integrating one's code with foreign code*.

Learners' Pyramid

The students of senior level undergraduate courses at Jaypee Institute of Information Technology are designated as *mentors of a junior level project*. Through this arrangement, a pyramid of large

number of learners can be created wherein the senior students learn the *leadership* and *mentoring* skills, while the juniors have the benefit of more easily accessible and friendly guidance. As the seniors guide the juniors, and sometimes also help them in debugging their work, it gives them the practice of reading and comprehending foreign code. It gives the opportunity to *refresh their basics, and also enhances their knowledge,* by asking more questions related to 'how,' 'why,' and 'why not'. It helps to visualize the same concepts from another perspective. This deepens and consolidates their learning, and helps appreciate the interrelationship of advanced level courses with junior level courses. Mentors have reported several other benefits for themselves – *experience of joy and satisfaction, enhanced confidence, improved understanding of self and others, appreciation of diversity, development of patience, empathy, multi-perspective and out of box thinking, improvement of analytical and debugging skills, as well as enhancement of communication, collaboration, leadership and decision making skills.*

6. Summary and Future Scope of Work

We have made efforts to identify the critical competencies for the software engineers. Further, we have reviewed the educational literature to examine its applicability for developing these competencies through appropriate interventions for instructional reform. We have finally proposed the following *five main types of interventions for instructional reform in computing course, with special reference to software development:*

1. Inquiry teaching
2. Project-inclusive teaching: project-centric and project-oriented teaching
3. Multilevel infusion
4. Integrative courses
5. Group and Community learning: pair programming and learners' pyramid

All these interventions were administered in some chosen set of computing courses. Inquiry teaching has been tried out in some core courses. It was found out that many students are not able to change their earlier learning habits, and hence, could not experience the advantages of deeper learning using this technique. Hence, in order to develop inquiry learning habit, puzzle solving has been integrated as the first component of introductory programming course. Initial results are very encouraging. Future research is required for its impact analysis, and also to investigate the

applicability of inquiry teaching in the context of different computing courses. Both forms of project-inclusive teaching have been adapted in many computing courses. More systematic studies are required to validate the effectiveness of the model in the context of specific computing courses.

A new graphic notation for modeling the software problems has been developed and infused in some courses. In order to develop estimation skills, the process data as adapted from PSP has been infused in many laboratory courses. In order to infuse debugging experience, taxonomy of software bugs has been prepared with an objective of designing debugging related assignments in various computing courses. Through multilevel infusion, the students are also becoming familiar with web, graphics, multimedia, mobile, security, etc., very early in their course work. This is bringing deeper integrated learning, higher levels of enthusiasm, and challenge in the courses. Further, some new courses have been designed to strengthen the integrative thinking. Some of these courses make an attempt to integrate several computing areas, while some other make an attempt to integrate computing content with human sciences. Future research is required to evaluate and refine the work.

New form of collaborative learning have been proposed and tried out. A novel approach of pair programming has been tested once in a large first year introductory programming course. A novel form of collaborative learning, Learning Pyramid, has been evolved, tested and scaled up. The results of sample tests were found to very encouraging. Further research is required to examine the impact, and also to investigate the ways of pervasively integrating it into the educational system.

With regards to future possibilities, there is a need for redefining, administering, and validating the multidimensional learning objectives and student engagements with respect to each computing course. Large classes offer a huge challenge. There is a need to explore the possibility of a complete revamp of the computing education and curriculum through multilevel infusion integrated with inquiry teaching, project-centric teaching, and group and community learning with special reference to software development. While some interventions have been

successfully tested with large classes, others were not as successful for large numbers. Hence, new ways of motivating students need to be explored. More work is required for creating an integrated curriculum, where not only the computing course, but also other courses offered by other departments for computing students, will also be well integrated into a single whole. More work is required on finding out the ways of increasing domain sensitivity and expertise of graduating software developers.

Thesis layout

The first chapter of the thesis introduces the thesis and also gives an overview of the motivation, objective, background, research method, and results of the reported work. In the second chapter, the required critical 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. Another survey was conducted to find out the distinguishing requirements of the software services and software product companies.

In the third chapter, the distinguishing features and multidimensional aspects of software development are analyzed with a view to further analyze the required competencies. In this process, a large number of software professionals have been consulted on various issues related to software development and required educational inputs. The three-dimensional taxonomy of competencies proposed in the second chapter is further refined in the light of newer findings.

The fourth chapter gives an overview of the chosen classical and contemporary learning theories. We review these theories with respect to their applicability for computing education, with a focus on development of required critical competencies for software development as consolidated in the third chapter. In this chapter, we also identify the core instructional principles that can help in designing the appropriate interventions for instructional reform in computing education. These principles are then used for designing the interventions for instructional reform discussed in the fifth and sixth chapter.

The fifth chapter focuses on the rationale for student-centric active learning. We examine the process of creating such a learning environment using inquiry teaching and project-inclusive teaching – project-centric as well as project-oriented. The notion of **cognitive dissonance** is central to our discussions in this chapter. Further, we also elaborate out attempts to apply these models to design some interventions for instructional reform in some regular undergraduate computing courses. A project-oriented approach does not necessarily require a transformation in the way of delivering the lectures. However, delivering courses through project-centric and inquiry teaching methods require a total transformation in the attitude of faculty and students.

The sixth chapter elaborates the interventions around the idea of integrated curriculum and collaborative learning through multilevel infusion, integrative courses, and group and community learning. The idea of **cognitive flexibility** forms the underlying theme of the discussions in this chapter. Many computing courses have been enriched by multilevel infusion of some pervasive computing concepts, processes, and technologies. In order to enhance cognitive flexibility, some courses attempt to integrate computing with human science. These human science concepts have been carefully chosen based on their relevance and importance for enhancing some critical competencies.

The seventh chapter provides a summary, and suggests future scope of research. In addition, an annexure is also included in the thesis to provide a review of published literature on evolution of computing curriculum in the last five decades.

Acknowledgements

This work is the result of a long personal journey across a variety of professional experiences – learning, designing, teaching, and also leading teams. This work has humbled me and has made me realize my own ocean size 'ignorance' about learning and also many intricacies of software development. I had the good fortune of wonderful and engaging interactions with experts and scholars of diverse disciplines.

To top the list, I am highly grateful to hundreds of software professionals who have participated in many surveys and discussions during the course of this research. I am thankful to all my past, present, and future students, who are my main inspiration for this work. Some of the past students are my most valued professional consultants, collaborator, and critiques. I also want to express my gratitude to all my enthusiastic colleagues in the Department of Computer Science and Engineering at Jaypee Institute of Information Technology for their confidence, support, and active collaboration in contextualizing and administering many ideas in their various courses. Many of them are highly committed to excellence in teaching, and are open to new educational ideas. Based on their enthusiasm, I am very hopeful that within a few years, this group will be able to make some trend setting innovations in the field of computing education.

I am most indebted to Dr. Mukul K. Sinha, my mentor for the last twenty years, for innumerable professional lessons and values that I have learnt from him. Only a few blessed people have the good fortune of receiving such selfless mentoring. I am also highly thankful to Prof. J.P. Gupta for his guidance, patience, tolerance, and support.

Numerous discussions with Prof. M.N. Faruqui encouraged me to maintain my enthusiasm for carrying out this research. I have learnt many lessons about curriculum design, and also critical inquiry, from Prof. S.L. Maskara, whose benchmarks, eye for detail, and commitment to excellence in engineering education are worth striving for. Several discussions with Prof. A.B. Bhattacharya and Mr. H.S. Dagar were very enriching. Moral support extended by Prof. S.K. Khanna, Prof. (Late) Prof. C.S. Jha, and Dr. Y. Medury has been very encouraging in this long journey. I am also highly grateful to the co-authors, reviewers, and editors of all my papers.

It will not be proper if I forget to acknowledge the lessons I learnt about the value of context and holistic thinking at Indira Gandhi National Centre for the Arts (IGNCA) during my stint ther from 1995 to 2002. My way of thinking and perception about scholarship, education, and its relationship with human life significantly evolved after innumerable interactions with Dr. Kapila Vatsyayan, Prof. P.S. Filliozat, Prof. T.S. Maxwell, Prof. Saskia Kersenboon, Prof. R.

Nagaswamy, Prof. Aditya Malik, Prof. B.N. Saraswati and many others during the course of designing several interactive multimedia learning systems at IGNCA.

Last and equally importantly, I am highly thankful to my family for their care and tolerance for my carelessness.

References

[1] *The Times of India* (2005), Learn to work, Editorial, June 23, India.

[2] ValueNotes (2004), R&D Outsourcing – The India edge: Key insights and success factors, Aug 2004, retrieved from http://www.researchandmarkets.com/reportinfo.asp?report_id=224141&t=e&cat_id=2, last accessed on October 15, 2005.

[3] C. Bodmer, A. Leu, L. Mira & H. Rütter (2002), SPINE: Successful practices in international engineering education, pp 92-102, retrieved from http://www.ingch.ch/pdfs/spinereport.pdf, last accessed on October 14, 2005.

[4] Sanjay Goel (2006), Investigations on required core competencies for engineering graduates with reference to Indian IT industry, *European Journal of Engineering Education,* Taylor & Francis, UK, October, pp 607-617.

[5] Sanjay Goel (2006), Competency based engineering education with reference to IT related disciplines: Is Indian system ready for transformation?, *Journal of Information Technology Education*, Informing Science Institute, USA, pp 27-52.

[6] Benjamin S. Bloom and David R. Krathwohl (1956). *Taxonomy of Educational Objectives: The Classification of Educational Goals*, by a committee of college and university examiners. Handbook I: Cognitive Domain, New York, Longmans, Green.

[7] P. Bourque L. Buglione A. Abran A. April (2003), Bloom's taxonomy levels for three software engineer profiles**,** *Proceedings of Eleventh Annual International Workshop on Software Technology and Engineering Practice*, 19-21 Sept., IEEE, pp 123 - 129

[8] Christopher W. Starr, Bill Manaris, RoxAnn H. Stalvey (2008), Bloom's taxonomy revisited: specifying assessable learning objectives in computer science, *Proceedings of the 39th*

SIGCSE technical symposium on Computer science education, February, USA, ACM, pp 261-265.

[9] Sanjay Goel (2004), What is high about higher education : Examining Engineering Education Through Bloom's Taxonomy, *The National Teaching & Learning Forum*, James Rhem & Associates, USA, Vol. 13 Number 4, pp 1-5.

[10] Sanjay Goel and Nalin Sharda (2004) What do engineers want? Examining engineering education through Bloom's taxonomy, *Proceedings of 15th Annual AASEE Conference*, Australia, pp 173-185.

[11] Robert J. Marzano, Debra J. Pickering, Daisy E. Arredondo, Diane E. Paynter, Guy J. Blackburn, Ronald S. Brandt, Cerylle A. Moffett, Jane E. Pollock, Jo Sue Whisler (1997). *Dimensions of Learning: Teacher's Manual*, 2nd ed. ASCD, UK, pp. 1-6.

[12] James Miller (2008), Triangulation as a basis got knowledge discovery in software engineering, *Journal of Empirical Software Engineering*, Springer, pp 223-228.

[13] Philip G. Armour (2000), The case for a new Business Model: Is software a product or a medium, Communications of ACM, USA, August, Vol. 43 No.8, pp 19-22.

[14] Philip G. Armour (2000), The Five orders of Ignorance: Viewing software development as knowledge acquisition and ignorance reduction, Communications of ACM, USA, October, Vol. 43 No.10, pp 19-20.

[15] Aaron Fried, Karen Zannini, Don Wheeler, Yongjin Lee, and Jose Cortez (2005), *Instructional Design Theory Database Project*, Syracuse University, retrieved from http://web.cortland.edu/frieda/ID/IDdatabase.html.

[16] R. M. Felder (1982), Does engineering education have anything to do with either one: Toward a systems approach to training engineers. R.J. Reynolds Industries Award Distinguished Lecture Series, North Carolina State University, USA. retrieved from http://www.ncsu.edu/felder-public/Papers/RJR%20Monograph.pdf.

[17] Sanjay Goel (2008), Successful Teaching methods for engineering education with reference to the Indian IT industry, *Journal of Science, Technology, Engineering and Mathematics Education*, USA, Accepted for publication with revision.

[18] Greg Kearsley (1994-2009), *Explorations in Learning & Instruction: The Theory Into Practice Database*, retrieved from http://tip.psychology.org

[19] F. Coffield, D. Moseley, E. Hall, K. Ecclestone (2004), Learning styles and pedagogy in post-16 learning: A systematic and critical review, Learning and Skills Research Centre, UK.

[20] George D. Kuh, et al, Exploring Different Dimensions of Student Engagement 2005 Annual Survey Results, National Survey Of Student Engagement, Indiana University, USA, 2005.

[21] R. M. Felder & R. Brent (2003), Designing and teaching courses to satisfy the ABET engineering criteria, *Journal of Engineering Education*, USA, January 2003, pp 7-25.

[22] R. M. Felder & R. Brent (2004), The intellectual development of science and engineering students Part 1: Models and challenges, *Journal of Engineering Education*, USA, 93 (4), pp 269-277.

[23] Sanjay Goel (2003), Activity based flexible credit definition, *Tomorrow's Professor, Stanford University, USA.*

[24] Sanjay Goel (2009), A proposal for a Tutorial on Enriching the Culture of Software Engineering Education through Theories of Knowledge and Learning, *Proceedings of 22nd Conference on Software Engineering Education and Training,* IEEE, pp 279-281.

[25] Roger C. Schank and Chip Cleary (1995), *Engines for Education*, pp 27-31, LEA Publishers, USA.

[26] E.G. Arias, H. Eden, G. Fischer,, &, E. Schraff (2000), "Transcending the Individual Human Mind – Creating Shared Understanding through Collaborative Design", *ACM Transactions on Computer Human-Interaction,* 7(1) pp. 84-113.

[27] L. Festinger, (1957), *A theory of cognitive dissonance*, Stanford University Press, Stanford, USA.

[28] Thomas F. Kamradt and Elizabeth J. Kamradt (1999), Structure Design for attitudal instruction, Charles M. Reigeluth (ed.)., *Instructional-design Theories and Models: A new paradigm of instructional theory Vol. II*, LEA, USA, pp 263-290.

[29] Barry Kort and Rob Reilly (2002), Theories for Deep Change in Affect sensitive Cognitive Machines: A Constructivist Model, *Journal of Educational Technology & Society,* Vol. 5 Issue 4, International Forum of Educational Technology & Society, USA, *pp 56-63.*

[30] Bruner, J. S. (1966). *Toward a theory of instruction*. Cambridge Massachusetts: Belknap Press.

[31] Said Hadjerrouit (2005), Constructivism as guiding philosophy for software engineering education, ACM SIGCSE Bulletin**,** Vol. 37, Issue 4, December, ACM, pp 45-49.

[32] Sanjay Goel (2006), Do Engineering Faculty Know What's Broken?, *The National Teaching & Learning Forum,* James Rhem & Associates, USA, Vol. 15 Number 2, pp 1-10.

[33] Ritu Arora and Sanjay Goel (2009), Software Engineering approach for teaching development of Scalable Enterprise Applications, proceedings of 22nd Conference on Software Engineering Education and Training, IEEE, pp 105-112.

[34] Spiro, R. J. & Jehng, J. (1990). Cognitive flexibility and hypertext: Theory and technology for the non-linear and multidimensional traversal of complex subject matter. D. Nix & R. Spiro (eds.), *Cognition, Education, and Multimedia*. Hillsdale, NJ: Erlbaum, pp 163-205.

[35] Watt S. Humphrey (2005), PSP: A self improvement process for software engineers, Addison-Wesley Professional, USA.

[36] Vikas Kumar and Sanjay Goel (2009), Software Bug Taxonomy for Effective Programming, *Journal of Computer Science Education*, Routledge, UK, Communicated.

[37] Dillenbourg P. (1999) What do you mean by collaborative learning? In P. Dillenbourg (Ed) *Collaborative-learning: Cognitive and Computational Approaches*. Pergamon Oxford: Elsevier pp.1-19.

[38] Lipponen, L. (2002). Exploring foundations for computer-supported collaborative learning, In G. Stahl (Ed.), *Computer Support for Collaborative Learning: Foundations for a CSCL community. Proceedings of the Computer-supported Collaborative Learning 2002 Conference* Hillsdale, NJ: Erlbaum, pp. 72-81.

[39] Sanjay Goel and Vanshi Kathuria (2008), A Novel approach for pair programming, Journal of Information Technology Education, Informing Science Institute, USA, Accepted for publication with revision.