CHAPTER 9: SOME INTERVENTIONS FOR ENHANCING THE QUALITY OF SOFTWARE DEVELOPMENT EDUCATION

In the previous chapter, we developed a framework of pedagogic engagements. This framework can be used for designing a large variety of instructional interventions to immerse students in a four-dimensional hierarchy of active, integrative, reflective, and collaborative pedagogical engagements. In this framework, we included two core principles for facilitating deep learning: cognitive dissonance [327] and cognitive flexibility [206]. Cognitive dissonance is about the conditions that are necessary for learning and cognitive flexibility is about the mastery of some subject matter. Instructional interventions designed with the help of this framework can help the faculty to create conditions of cognitive dissonance and flexible learning of the subject. In this chapter, we discuss some such interventions developed by us. As stated in the previous chapter, development of the theoretical framework and these empirical interventions has been an intertwined process.

Learning primarily happens because of learners’ engagement in various activities relevant to the content, rather than mainly depending upon content’s transmission from external sources. Student-centric active learning 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. The author has applied some in-class active learning techniques such as think-pair-share, share your experiences with the project and assignments, design a small algorithm, and so on and also advocated a strategy of activity based flexible credit definition as one component of learner-centric education [358].

Baumgartner [359] has proposed a framework for viewing teaching as a designed activity, and has observed that teachers employ a diverse range of coaching and mentoring strategies like ‘teacher as guide, ‘teacher as project manager,’ and ‘teacher as troubleshooter’ in open-ended learner-centric classrooms to support students during their design process. It has been suggested
that an open-ended approach encourages a diversity of views and perspectives, and also makes a critical reflection on observations and experiences possible.

**Section 9.1: Increasing Cognitive Dissonance through Problem-centric Approach in Software Development Education**

The most natural way to create dissonance (Section 8.2.1) would be to lead learners through a problem-centric approach. We elaborate upon three types interventions based on problem-centric approach: Inquiry Teaching, Project-inclusive Teaching, and Reflective Teaching/Assignments. In all these interventions, we try to engage the students at higher levels of all the four-dimensions of our framework of pedagogical engagements: active, integrative, reflective, and collaborative (Section 8.3).

**Section 9.1.1: Inquiry Teaching in Software Development Education**

The lecture format in which abstraction precedes the instantiation and concretisation, helps students in developing skills in linear thinking and deductive reasoning, and also succeeds in creating a knowledge-base as an inventory of unutilised concepts. It however fails miserably to give direct and guided practice in inductive reasoning and lateral thinking. 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. 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. We have developed a new model, SERO, for designing inquiry teaching oriented lectures. This mode has been tried out in some courses, viz., Data Structures, Computer Graphics, Orientation to Engineering, etc.
Section 9.1.1.1: SERO Model for Inquiry Teaching in Software Development Education

The discourse in the lecture classroom can be viewed as a story telling artifact. The objective of this artifact is to create a meaningful learning experience and knowledge structures for every learner. The discourse in a large number of lectures is designed as a closed artifact that primarily sees the students as consumers. A fundamental challenge for designers in the new millennium is to design open systems and artifacts by inventing and designing a culture in which humans can express themselves and engage in personally meaningful activities [317]. Open systems and artifacts must evolve, they cannot be completely designed prior to use. They must evolve at the hands of the users, and they must be designed for evolution. The dichotomy of designer and user has to be eschewed. *Seeding, Evolutionary growth, and Reseeding (SER) has been proposed as a conceptual framework for designing sustainable, open, and evolutionary systems* [318] [360].

A seed is the initial state of a system that is intended to evolve. The evolutionary growth phase is one of unplanned evolution as the seed is used by the members of a community to do work. Reseeding is a deliberate effort to organize, formalize, and generalize knowledge created during the evolutionary growth phase. Courses as seeds have been proposed as a promising model to evolve and enrich courses by allowing students to act as active contributors, and not just as passive consumers [361].

The genesis of any story experience is Emotional Movement [362]. Users crave emotional engagement and stimulation. Situated inside the context of lecture classroom, every learner (user) is the author of his own personal meaning. Meaning is the product of interaction between the observer and the system, the content of which is in a state of flux, of endless change and transformation [363]. In Poetics, Aristotle suggested that a well constructed plot must be a whole having beginning, middle, and end [364]. Movement Oriented Design (MOD) views a story as an ensemble of ‘story units’ in which a ‘story unit’ has three parts, the Begin, Middle, and the End (BME) [365]. Begin lays the groundwork, hooks the user, imploring to find out more. Middle carries the main story message, conveys the core meaning. End terminates the story, concludes the current story, and/or links to the next.
As per the SERO model, every lecture is delivered as a series of SER blocks, and concluded with a learning Outcome. **Seed** is the fresh idea or question from a teacher which is generally not an obvious derivative of an earlier idea. **Evolution** has been used to label the active learning phase in the class involving individual thinking, group work, discussions (among student groups of varying size, and also between the students and teacher), and solving problems that require thinking in terms of analysis, synthesis, and/or evaluation. **Reseed** is being used to label the phase of formalizing the informal ideas generated during the evolution stage, and deriving another seed as a derivative of this evolution.

Students usually have greater motivation to learn in the context of solving a problem, than if the content is delivered out of context [366]. The seeding phase in SERO based lectures offers good opportunity to create context. Situated in this context, the content is developed during the evolution phase through problem solving activities.

The teacher makes a deliberate attempt not to deliver generalized content without the context or before problem solving. Instead, the generalisations are presented as a natural fallout of the theorising process through solution-unification during the reseeding phase to conclude the evolution phase. In this model, the teacher has to support the students during evolution phase individually or in smaller groups and only some time the entire class.

The teacher needs to be the centre of attention of the entire class only during the limited period of **seed** and **reseed** stages, and occasionally during the **evolution** stage, as and when the need arises. Sometimes the evolution phase may also become teacher-centric, as the teacher may occasionally decide to demonstrate the problem solving process with some specific case(s), rather than engaging the students in problem solving because of the lack of sufficient background with the students or time constraints. However, the problem solving characteristic of the evolution phase remains unchanged. At the end, the learning outcomes are summarized and an assignment is announced. This assignment forms the reseed for the next class.

Usually there are not many seeds in a lecture, only reseeds. Most of the time is used in evolution and active learning. This model has been tried out successfully in many courses, even with a
large number of students. Figure 9.1 shows pictures of one such class during the evolutionary growth of a concept through group exercise. Appendix A14 gives a summary of two such lectures, one each in computer graphics and data structures.

![Figure 9.1: Group exercise during the evolutionary phase of SERO style lecture](image)

**Experience**

SERO style lecture classes were found to be highly engaging and useful by motivated undergraduate students. However, many other students, who were mainly motivated by examination oriented study, did not find these classes very useful for them.

**Challenges for Inquiry Teaching in Software Development Education**

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 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 in Software Development Education through Puzzle Solving

Solving a puzzle 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, we redesigned 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. In this course, at two different campuses of JIIT, over eight hundred first year engineering students were distributed in six lecture sections and twenty tutorial sections. More than twenty faculty members were involved in delivering lectures and running weekly tutorial classes. All the concerned faculty members (Prakash Kumar, Alok Agarwal, Vikas Saxena, Shikha Mehta, Anshul Gakhar, and Chetna Debas) agreed to the proposal that instead of teaching programming or computer basics, we should start solving puzzles. For over a month, various kinds of puzzles were discussed in the lecture and tutorial classes. The puzzles were collected and chosen by the concerned faculty members. As per the feedback from the faculty, these classes were highly active and collaborative. Even in the post-lunch sessions, students very enthusiastically came to these classes.

Faculty members felt that puzzle solving activity improved students’ logical thinking ability, which is at the core for designing computer programs. A large number of students have reported multi-dimensional benefits in terms of enhancements in logical, creative, multi-perspective, and out-of-box thinking, attention, focus, concentration, patience, comprehension, urge for creation, etc.

The faculty members expressed that these were the most active and collaborative classes they had ever attended or conducted. It showed them the benefits of active and collaborative inquiry oriented classes. Encouraged by the positive results of this trial, we have now infused puzzled solving in two more courses in the current semester. In ‘data structures’ (2nd semester) and ‘fundamentals of algorithms’ (4th semester) courses, all teachers have happily dedicated the first
one to two weeks solely to puzzle solving. A more structured research in needed on infusing puzzle solving in software development education.

**Section 9.1.2: Project-inclusive Teaching in Software Development Education**

According to various surveys discussed in chapters four to seven, we found that projects were the most effective teaching methods with respect to enhancing various competencies relevant to software development. Semester-long project experience helps in developing multidimensional competencies in all the dimensions. Hence, semester-long projects have the potential to facilitate deeper learning in many significant ways. However, projects are usually conceived as a culmination activity of learning something. It is assumed that only after completion of conceptual learning and acquiring practical skills, some project can be attempted. *Usually in Indian universities, semester-long project work is not included as part of the regular computing courses. This limits the effect of the courses in terms of developing their competencies.*

**Project-Inclusive Regular Courses**

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*. It also opens many new challenges for the faculty. They have to guide the students in formulating and completing their projects. Simultaneously, they also have to manage the learning process. Hence, project-inclusive teaching also offers a higher level of creative opportunity for the faculty as well.

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*. We have tried to enhance the quality of several undergraduate computing courses by project inclusion. This attempt has given us the confidence that that it is usually possible to plan and deliver the courses with a central focus on the semester-long project work of the students. *Two different models, viz., project-centric evolutionary instruction and project-oriented instruction, have been proposed* for achieving this goal.
**Project-centric Evolutionary Teaching in Software Development Education**

Project centric evolutionary teaching offers *active, integrative, reflective as well as collaborative engagements* as per our frame work discussed in Section 8.3.

During the course of this research, project-centric evolutionary teaching was evolved for Enterprise Application Development [367]. Recently, it has been further expanded to many other courses like object oriented programming, database management, web application engineering, software engineering, and information systems.

In project-centric teaching, *we reverse the traditional teaching methodology* in which conceptual learning is followed by practice assignments, and only sometimes project work. In our scheme, at the beginning of the course, the teachers first help and guide the students to formulate the initial project problem. Examples and templates are used to complete this task. Since it is not possible for the teacher to discuss every project in the large class, the instructor then selects some of these projects to forward the subsequent classroom discussions. They try to define the initial and simplistic project scoping and specification for one or two projects through classroom discussion. The students follow a similar process to complete these tasks for their projects. Teacher guide the students to *incrementally enhance their project scope later* in the semester, essentially to create the context for the forthcoming concepts and topics of the subject matter. *They refine the project scope before introducing any new topic.*

The teacher has to bring in the concepts after setting the context. Conceptually, this model has some similarity to zero inventory manufacturing practice. The learners are not given a large inventory of unused concepts. The concepts are introduced only after creating the need for its use with reference to students’ semester-long project.

We have developed the conceptual schema for defining the main characteristics of student projects’ evolution in project-centric evolutionary teaching of object-oriented programming, software engineering, database management systems, web application engineering, enterprise software development and information systems. All these schemas have also been tried in real courses by concerned faculty members. Appendix A15 gives the stages of evolution of the defining characteristics of student projects in different computing courses.
Progressive evolution of the subject matter in the evolving context of a project is the hallmark of this approach. Appropriate concepts, theories, technologies, procedures, and tools are introduced as per the needs of each stage of project evolution. These stages are flexible enough to accommodate new concepts, theories, technologies, procedures, and tools at each stage. Metaphorically, it looks at content delivery as a natural process of a small but complete bud blossoming into a complete flower, rather than like the traditional but unnatural compartmentalized additive advancement. As per this model, most of the projects have high similarity with respect to technological issues. The application domain becomes the main differentiating factor for different student projects. Hence, observation and review of other students’ projects also gives an opportunity to expose the students to a variety of application domains.

This model is very suitable when the student projects can be planned to use most of the concepts of the subject matter. This model is not suitable if the objective is to have the students to carry out their projects in different areas of the subject matter, and the projects are required to be differentiated based on their technological aspects, rather than application domains.

**Project-Oriented Teaching**

While engagement in semester-long projects is highly beneficial for ensuring deep learning in courses, project-centred teaching has its own difficulties as well as limitations. In many courses, it is very difficult, and perhaps not even desirable, to plan technically similar semester-long student projects encompassing most of the topics of the particular subject matter. In such cases, it is better to plan projects on different topics. In order to leverage the advantage of peer learning, care has to be taken to evenly distribute the students’ projects over all the main topics. However, this scheme imposes some challenges regarding synchronising the project activity with the content delivery in the class. Either most students are not able to start their projects early in the semester or they have to start the project without any instructional support on the project topic. 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 Compiler 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. During this phase, the major issues, relevance, and typical project possibilities with respect to all topics are presented to the students. The objective of this phase is to help students broadly understand the subject matter, see the inter-connections between different topics, and also identify their project topics as well as formulate their project problems. Thereafter, the students start working on their projects.

In the second phase, the topics are sequentially picked up for in-depth classroom discussion.

If students’ projects are evenly distributed over all the main topics, topic related projects can be easily leveraged to provide the context and enrichment for detailed discussion on the topic, and also give a partial flavour of project-centric teaching for every topic.

Section 9.1.3: Creating Conditions for Reflective Engagements in Software Development Education

Reflection is not an automatic activity. Students do not usually automatically reflect well upon their actions and tasks in various assignments. This limits not only the quality of their assignments, but their overall learning as well. A small post-assignment, reflective activity can amplify their learning from the same assignments. Borton’s framework for reflective thinking [325] discussed in Section 5.2 and Section 8.3.3, includes three questions: ‘what, so what, and now what?’

We have successfully deployed this framework to enhance the learning value of many assignments. For example, for the past three years, we have been asking the students to maintain a log (PSP style), of their time and programming errors in software laboratories. More details about infusion of PSP (time estimation as well as bug) are discussed in Sections 9.2.1. Many students were finding it to be a wasteful activity. We realized that because they never referred to their logs, they saw no benefit of creating such logs. Hence, recently we introduced an element of mid-semester reflective exercise, where a group of few students jointly review their logs and
write a reflective report using Borton’s framework. This exercise helped the students to draw and see the benefits of maintaining a log. They saw what kind of errors they were commonly making, and how they compared with other peers. This helped them in improving their programming skills.

For the last several years, we have been asking the final-year students to submit a reflection report on their final year project. After completing the project, they are required to give an additional report answering the questions given in Table A16.1 (Appendix A16). Further, in the main project report, we have added elements that require reflective thinking. These include project specific reflective review of quality assurance procedures, debugging, risk recovery, and error and exception handling techniques.

In 2009-10, we have specifically tried to inculcate reflective thinking through reflective engagements in several courses. For example, in three elective courses for the 8th semester undergraduate students, ‘software documentation,’ ‘software construction,’ and ‘software risk engineering,’ delivered at a fast pace in three weeks, reflection has been used very strongly. In each of these three courses, the students were required to write a report, reflecting on their 7th semester project in the light of subject knowledge of the respective courses. They were required to suggest strategies to improve their project’s specific aspects that were related to the subject matter of these specific subjects. Table A16.2 (Appendix A16), gives the problem statement of the assignment (25% credit) in these three courses. Further, even in the final exam, question(s) were asked to make them reflect upon this work. These were finally designed by the concerned faculty members in consultation with the author of this thesis. In future, we plan to create more templates for infusion of critical thinking and reflection in many computing courses.

In an ongoing elective course, ‘software arteology,’ reflection is being infused in all assignments. The assignments require them to essentially reflect upon published literature, professional’s experiences, peer’s experiences, or their own experience. Further, at the end of every assignment, they are also required to write a small report specifically addressing the issues as per 2nd and 3rd questions in Borton’s model. Table A16.3 gives two sample assignments in this course. The students are required to submit their assignments, only after a peer review. This also brings some
elements of collaboration and reflection on others’ work. The second-last and last sub-questions in each assignment are based on the 2nd and 3rd question respectively as per Borton’s framework. The responses of the last sub-question, in each assignment are particularly very interesting, where students are expressing what they learnt by doing the specific assignment, and what they plan to change in future. Table A16.4 (Appendix A16) gives a few sample responses.

We conclude that reflective engagements are highly effective for creating deeper learning. More work is required to reflective assignments in all courses and tasks.

**Section 9.2: Increasing Cognitive Flexibility through a Multifaceted Integrated Approach in Software Development Education**

In order to engage the students at higher levels of integrative engagements, the second dimension of our four-dimensional engagement taxonomy discussed in Section 8.3, and to impart cognitive flexibility with special reference to software development, an integrated approach to software development education is necessary. In order to achieve this objective, we have visualized and administered *three types of interventions for instructional reform*, viz., *Multilevel Infusion* of key technologies (web, multimedia, mobile, and security) and professional practices (systems design, estimation, open source, and debugging), *Integrative Courses* and *Group and Community Learning*.

**Section 9.2.1: Multilevel Infusion for Continuous Integration in Software Development Education**

As mentioned in Section 7.4.1, the details of multi-level infusion of various technologies and professional practices are discussed below. This kind of ‘*multi-level infusion*’ offers ‘*Inter-subject intra-disciplinary relational*’ engagement to the students as per Table 8.8. In the following discussion, we also refer to the feedback received from mentors (Appendix A17). The details of our intervention to engage senior level students as cross-level mentors are discussed later in section 9.2.3.2. It may be noted that every mentor was mentoring only one host course.
Multi-level Infusion of Web Technology
Web technology is integrated in several introductory courses. The first programming laboratory courses, introduces HTML, before any practice with programming. In the data structures course, they do some programming assignments around HTML files. JDBC is introduced in database course. Some web search and page ranking algorithms are included in the fundamental of algorithms course. Information systems course focuses on building web-enabled information systems. Computer network courses starts from the topmost layer of the protocol stack, leveraging students’ familiarity with the web, and goes deeper to lower layers. Table A17.1 (Appendix A17) gives a summary of the feedback received from the mentors regarding infusion of web technology in different junior level courses.

Multi-level Infusion of Multimedia Technology
Multimedia technologies are infused in many introductory courses. In the first programming course, students learn to use basic graphics and sound functions. In the data structures course, deep practice of recursion is given with the help of several examples of graphical fractals. They also learn some basic data structures for simple geometric objects. The data structures for building simple games like snake-and-ladders, and ludo, etc., are also discussed. Database systems course insists on creating databases with multimedia objects. Graphics API’s are used in object-oriented programming as well. In the algorithms course, students are required to write programs for algorithms visualization and also implement simple games, using decision trees. In the web application engineering course, hypermedia design patterns are introduced. Table A17.2 (Appendix A17) gives a summary of the feedback received from the mentors regarding infusion of the multimedia technology in different junior level courses.

Multi-level Infusion of Mobile Technology
Aspects of mobile computing have also been infused in some introductory courses. For example, in the operating systems course, an overview of mobile operating systems is given. J2ME is included in database systems and web application engineering. The course on information systems includes Android. The courses on computer organization, microprocessors, and computer architecture also bring some discussion about mobile platforms. Table A17.3
(Appendix A17) gives a summary of the feedback received from the mentors regarding infusion of the mobile technology in different junior level courses.

**Multi-level Infusion of Security Aspects**

Recently, some attempts [368-370] have been made to incorporate security aspects in computing courses. We give an elaborate model of infusing security related aspects in every semester of the first three years. We have chosen traditional computing courses for infusing the selected security aspects without overloading the students or compromising on the main topics of the core course. Appendix A18 gives the details of this model [370a]. Some of the topics indicated for each course, can be easily integrated by interested faculty.

Some features of this model have already been tested in our courses. Infusion of security aspects in our courses, including some laboratory assignments has been highly appreciated by final year students who are mentoring the laboratories of first three year courses. Table A17.4 (Appendix A17) gives a summary of the feedback received from the mentors regarding infusion of security aspects in different junior level courses.

**Multi-level Infusion of Systems Design Aspects**

In order to lay an emphasis on systems design, some improvisations like necessity of design diagramming, evolutionary project scoping (in many courses), and necessity of following design guidelines and standards (in some courses), have been visualized and administered. For helping design diagramming habits for analysis and design of systems, a new graphic notation, *Concept Map*, for representing software systems has been developed and administered in some courses. The details of this graphic notation are given in Appendix A19.

Faculty and students have found this concept mapping technique to be very useful. It has been used several times in the data Structures course. Recently, it has also been deployed in some other advanced level courses to compliment the standard notation of UML. Table A17.5 (Appendix A17) gives a summary of the feedback received from the mentors regarding infusion of the some of the system design related aspects in many aspects in different junior level courses.
Multi-level Infusion of Estimation Tools

In order to develop estimation skills, the programming process related data, as adapted from the Personal Software Process (PSP) [371], has been administered in many computing laboratory courses of the first to third year. Initially, the students show a lot of unwillingness to record the PSP logs of their progress. It is perceived as an unnecessary burden that has nothing to do with their programming tasks. However, a reflection after some practice makes many of them self-realize the benefits of using it with respect to their programming practice. In April 2009, the students were asked to write their comments on the use of the PSP in their computing laboratory. This was an open-ended feedback. A 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. Table 9.1 gives a summary of the feedback received from students regarding the perceived benefits of maintaining PSP logs in their computing laboratory courses.

Table 9.1: Benefits of PSP as perceived by Students

<table>
<thead>
<tr>
<th>Benefits of PSP perceived by Students</th>
<th>2nd Semester students, 109 responses</th>
<th>4th Semester students, 91 responses</th>
<th>6th Semester students, 75 responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming Efficiency Enhancement</td>
<td>57%</td>
<td>59%</td>
<td>76%</td>
</tr>
<tr>
<td>Defect Rate Reduction</td>
<td>37%</td>
<td>37%</td>
<td>59%</td>
</tr>
<tr>
<td>Activity Record and Reflection</td>
<td>32%</td>
<td>39%</td>
<td>59%</td>
</tr>
<tr>
<td>Estimations and Planning</td>
<td>25%</td>
<td>27%</td>
<td>31%</td>
</tr>
</tbody>
</table>

Based on this feedback, we realized that Humphrey’s format of PSP logs is not good enough for enhancing the estimation skills of undergraduate students. Hence, we have modified it for achieving higher gains in estimation as well. In our new PSP format, students are required to write their estimated time for completing their programming assignments. Students are also required to revise their estimates after every stage of the software development: analysis, design, and implementation. Finally, they also record the actual time for completing their assignments. With this continuous engagement with estimation, they become sensitive to its importance. Further, every revision in estimates makes them more careful while making future estimates. Table A17.6 (Appendix A17) gives a summary of the feedback received from the mentors regarding infusion of the estimation related tools and techniques in different junior level courses.
Multi-level Infusion of Open Source

Open source has been infused in many ways. Students are encouraged to search, select, and include/modify open source code in their projects: mini projects in all courses, minor projects, and final year project. Since 2008, all final year students are required to give an additional report on how they used and/or modified the open source code in their project. Open source is regarded as published literature, and all literature survey oriented assignments allow and often insist on inclusion of open source survey, e.g., in advanced data structure (M.Tech. course), all students were required to survey open source in their chosen application domain, and catalogue the data structures used in chosen code. In the first programming course, the 1st semester undergraduate students start their programming laboratories with introductory Python, even before they use C. Core Java is taught in the object-oriented programming course. MySql and JDBC are used in the database course. Linux is used in the Unix laboratory course. The web application engineering course includes PHP, Java Script, XML, J2ME, etc. Linux is used in Operating systems course.

In 2008, for their 5 credit minor project-I, all 5th semester computing students were required to enhance an open source project in the area of software engineering (any phase). In 2009, all 5th semester computing students built database driven websites using open source add-ons like crawlers, security API, J2ME, etc. Since 2007, program comprehension and re-engineering have been included in the 5th semester software engineering course. In 2008, all 6th semester B.Tech (IT) students used Wonderland for creating database integrated virtual worlds in their second minor project. J2EE, Android SDK, XMS API, Ajax, Open XLS, and Drupal are being used in the information systems course. NS2 simulator is used in the computer networks lab. The elective course, computer graphics, emphasized the use and/or extension of open source game engine, e.g., Box2D, Box3D, J-monkey, Ogre-M, etc. Table A17.7 (Appendix A17) gives a summary of the feedback received from the mentors regarding infusion of the open source in different junior level courses.

Multi-level Infusion of Debugging

Debugging is generally thought of as an implicit activity for software development. Students are expected to detect and fix their buggy code. Due to this expectation, students typically have limited experience in bug detection. Not only this, the debugging experience they get is purely...
by chance and not by design. As part of assignments, students can be asked to identify and correct code section containing buggy code. The computing curriculum recommendations by professional societies [1] [50] [52] [113] also include debugging. To address this issue, assignments for bug detection and removal should be given. However, we feel that the typical delivery of computing courses does not enforce students to have debugging experience in a systematic manner. To bridge this gap between suggested curriculum and its actual implementation, we propose some guidelines for assignments. Bug detection with and without use of debugging tools should be inculcated among students.

In order to infuse debugging experience, we have prepared a taxonomy of software bugs with an objective of designing debugging related assignments in various computing course. This taxonomy has been summarized in Appendix A6. We take a view that software bugs are results of misconceptions about specific topics in specific courses. Debugging assignments can be given during delivery of courses based on topics included in our taxonomy of bugs. The taxonomy can be used as an input for generating these assignments. Bugs related to a particular course, as mentioned in our taxonomy, should be experienced by students during the course delivery. This can be done through assignments such as bug detection, bug generation, comparative study of debugging tools, and program comprehension of existing debugging tools, creating simple debugging tools for specific bugs and enhancing existing debugging tools. Additionally, students can be asked to maintain a bug log for every programming assignment as prescribed by Humphrey [371].

The log format suggested by Humphrey has the following parameters: (i) date of bug detection, (ii) sequential numbering of bug, (iii) bug category, (iv) phase of SDLC in which the bug was injected, (v) phase of SDLC in which the bug was removed, (vi) time spent in finding and fixing the current bug, (vii) bug number for the bug whose fix resulted in current bug, (viii) brief description of the bug, mentioning its reason. We propose that following parameters should also be incorporated in the bug log: (i) behavioural manifestation: the symptoms of running system that helped in finding bug, and (ii) techniques and tools that helped in fixing bug.
Reflecting on the data from the bugs log will help the students in systematizing their art of debugging. In our experience of introducing this log in laboratory work of various computing courses like introduction to programming, data structures, object-oriented programming, database management systems, algorithm, software engineering, information systems, and compiler design has been encouraging. A large number of students have felt that it helps them to improving their programming and debugging skills. However, a good number of students, especially at the first year level have found this to be too time consuming. While a two-third majority of second and third year students appreciated the benefit of maintaining this log, only one-third of the first year students found it to be useful. Table 9.1 summarizes the benefits, perceived by the students. Table A17.8 (Appendix A17) provides a summary of mentors’ feedback on PSP logs in some computing laboratories. Appendix A20 gives some more proposed interventions in this regard.

**Some more aspects being considered for multi-level infusion in software development education**

In collaboration with various faculty members, we continue to strengthen the infusion of the abovementioned eight elements: web technology, multimedia technology, mobile technology, security aspects, design aspects, estimation aspects, open source, and debugging in various introductory computing courses. Now, we have also started working towards designing appropriate models for infusion of selected elements of the following important issues related to software development:

1. Software documentation
2. Software quality
3. Software risks managements
4. Advanced level programming techniques
5. Formal methods in software engineering

Some elements of these have already been infused in the final year project deliverables. More work is needed to offer courses using the approach of *multi-level inter-disciplinary infusion.*
**Section 9.2.2: Integrative Capstone Courses in Software Development Education**

As discussed in Section 7.4.1, integrative capstone courses can help in strengthening nonlinear, integrative and systems thinking, and flexible learning.

**Courses for Intra-disciplinary Integration of Diversified Computing Topics**

We have made some attempts to design some advanced level computing courses like ‘multi-dimensional data structures,’ ‘systems programming,’ and ‘graph algorithms and applications’ to bring integration of otherwise widely spread computing concepts. These integrative courses offer ‘Inter-subject intra-disciplinary relational’ engagement level as per Table 8.8.

Data structures design is a pervasive computing concept, and design of application specific data structures is a crucial software development activity. The basic course on data structures helps in creating a general purpose foundation for most of the later computing courses. In this basic course, usually some common linear and non-linear data structures are introduced in multiple application contexts. Later, the courses on algorithms try to further build up this generic understanding. Many of the specialized computing courses can be modeled as data structures, algorithms, and methods. Limited space in the curriculum does not give the opportunity to take many such specialized courses. An integrative course on ‘has been created to deepen problem solving ability with a special focus on various domains involving n-dimensional, multimedia, and spatio-temporal data. This course leverages the advantages of basic foundation data structures course and multilevel infusion. It offers students an opportunity to learn about some important computation issues related to various areas of computing, e.g., computer graphics, image processing, multimedia, GIS, robotics, data mining, mobile computing, bio-informatics, VLSI Layout, etc., in a single course. More importantly, they also understand and explore the reusability of many multi-dimensional data structures across application domains. This approach enables flexible learning.

Similarly, the course on ‘graph algorithms and applications,’ attempts to contextualize the graph based algorithms in a variety of application area. The course on ‘systems programming’ leverages the learning in microprocessors, operating systems, compiler design and computer networks.
Courses for Inter-disciplinary Integration with Selected Elements of Human Sciences

Over the years disciplines have evolved such that they have been separated not only in terms of underlying factual, conceptual, and procedural knowledge, in theoretical as well as empirical space, but also in terms of research questions, perspectives, meta-cognition, and methodologies. In modern times, most of the interesting developments are taking place at the edges of the disciplines. The disciplines are getting integrated not only in terms of content but also perspectives and methodologies. A trans-disciplinary approach is required for solving most large real-life problems. Hence, the integration of seemingly disconnected disciplines of human knowledge offers very exciting learning opportunities. Here we elaborate upon some experiences in designing and delivering some courses that try to contextualize 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 core competencies. We have made an attempt to create a fourth orbit inter-disciplinary integration (Table 8.8) with some traces of trans-disciplinarity in these courses.

Theory of Knowledge, Learning, and Research

As discussed in Section 3.7, Armour [120] [148] 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 course of ‘theory of knowledge, learning, and research’ attempts to addresses this requirement. The students are exposed to a spectrum of theories related to human learning and thinking. All these theories are also used for reflecting about learning in general, and also with specific reference to software development. It enhances the 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. They are motivated to view software development as a learning and critical thinking activity.
Human Aspects for Information Technology

Another multi-disciplinary integrative course is ‘human aspects for information technology.’ It aims to explore the humanistic grounds for information technology and software development. Students analyze the required competencies for specific activities of software development. The students evaluate the information technology, and also the activity of software development with respect to multi-dimensional 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. They are engaged in collecting and analyzing professional dilemmas of practicing software developers in the light of theories of moral reasoning and human development. They learn about technological disasters and failures of software systems.

Various codes of professional ethics for engineers are also analyzed in this course. All these experiences help the students to understand the meaning and importance of professional responsibility. In this backdrop, the models of critical thinking are used for analyzing the ethical issues with respect to ongoing developments in information technology.

Finally, a module on creative thinking and inventive problem solving is integrated with this background (as per Table 8.6b). Selected models of creative thinking and inventive problem solving are used for designing ethically sensitive technological solutions and services. Further, there is strong tradition of formally teaching ‘Research Methodology’ in many non engineering disciplines. Such content is not usually offered in engineering disciplines. However, some programs of information systems offer such courses. The research in the field of software development combines the research methods of engineering as well as social sciences. In this course, various qualitative as well as quantitative research methods are discussed with the help of illustrative examples from the published research literature in software development. It also helps in enhancing the critical and integrative thinking, analytical skills, and also self-learning.

Section 9.2.3: Group and Community Oriented Engagements in Software Development Education

In the following two sub-sections, we report our experiments with two different models of group and community learning. Collaborative pair and quadruple programming prepare the students
for teamwork and benevolence, while cross-level peer mentoring is aimed for preparing them for larger organizational concerns, universalism, and responsible citizenship of larger communities. The students reported several benefits of collaborative pair programming 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. Both forms of group and community oriented engagements help in enhancing students’ sense of accountability and responsibility, ability to accommodate themselves to others, to see themselves as bound to all humans with ties of recognition and concern, as well as multi-perspective and creative thinking.

Section 9.2.3.1: Collaborative Pair and Quadruple Programming

Using our framework, we have transformed the popular concept of pair programming, to make sure that both the students in a pair necessarily collaborate, build upon each other’s work, and also do an equal amount of similar work. Our adapted implementation of ‘collaborative pair programming’ is based on facilitating higher levels of collaboration using Dillenbourg’s four conditions of collaborative learning as discussed earlier (Table 8.10). Table 9.2 shows Dillenbourg's four requirements for maximizing collaborative learning, and how we implemented each in our study.

Table 9.2: Application of Dillenbourg's principles

<table>
<thead>
<tr>
<th>Dillenbourg's requirement</th>
<th>Our implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Set up the initial conditions.</td>
<td>Pairs of students without any programming experience were formed by faculty in the beginning of the semester.</td>
</tr>
<tr>
<td>2. Over-specify the collaboration contract with a scenario based on roles.</td>
<td>In each laboratory session, the members of each pair were first required to individually complete two different programming tasks. On completion of both their individual tasks, they worked together to solve a more complex problem that was designed as an extension of both their individual problems (ref: Table A21.1, Appendix A21).</td>
</tr>
<tr>
<td>3. Scaffold productive interactions by encompassing interaction in the medium.</td>
<td>The pair members were not allowed to interact for completing their individual tasks. However, if one of the pair member completed his/her task much in advance, and the other member felt the need of peer’s support even for completing his/her individual task, the laboratory instructor allowed them to do so by assigning a small penalty of marks to the second member.</td>
</tr>
<tr>
<td>4. Monitor and regulate the interactions.</td>
<td>For every group of thirty students, at least two faculty members and one teaching assistant were available for clearance of doubts and monitoring.</td>
</tr>
</tbody>
</table>
Each exercise fulfilled the purpose of making the students work both individually and in a team. Table A21.1 (Appendix A21) shows a few sample exercises [389]. The concerned teaching faculty validated these assignments before administering the same to students. Appendix A21 gives some more details about the setting up of experiment.

For all the laboratory instructors and teaching assistants, the most common observations were to find paired programmers brainstorming far more than individual programmers, suggesting alternate implementations during evaluations, approaching instructors for doubts lesser than individual programmers, and having more details like null checks and memory checks in their programs. It led the students to check their thinking and reason their decisions, they examined and discussed their ideas with others, and evaluated other’s statements and solutions. They modified their own programs to fit in their code in the new but similar situation presented by the combined task question, and at the same time also acted as evaluators for their partner’s programs. We believe that this experience trained them for reading and building upon others’ code in future. The instructors also felt that student pairing also helped in improving the effectiveness of teacher student interaction in the labs.

Based on the results and the feedback from the students and instructors, some of the evident advantages of collaborative programming that we could bring out effectively in our course were: **efficiency, trust and teamwork, problem solving skills, and quality.** By the end of the semester, inexperienced-paired programmers reduced the relative performance gap from 40% to 10%, and performed at the same level as the experienced-solo programmers during the final examinations.

In 2009, we administered this form of collaborative programming in the laboratories of the object-oriented programming course for more than 350 students of the third semester of B.Tech. (CSE/IT). These students have already had two semesters of programming experience, but had not experienced pair programming in their earlier courses. Based on the same model, the assignments were designed by five colleagues teaching this course. In the laboratories of this course, we also had approximately thirty final year students as regular visitors, who act like mentors of third semester students. The details of mentoring program are discussed in the next
sub-section. Approximately 70% of these mentors have felt this form of collaborative programming to be extremely valuable for students’ long term as well as short term gains. Another 20% have also found it to be valuable, and also felt that this instructional intervention is worth the extra effort by students. These mentors have felt that it exposes students to observe different ways of programming, improve their style, and also encourages weak programmers to learn to program.

Collaborative Quadruple Programming

Our approach combines all the levels of collaboration proposed by Salmon (Table 8.10). The regular two-stage fixed-partner pair programming model has been further enriched by occasional extension into a three-stage semi-fixed partner quadruple programming model. For the purpose of occasional extension into a three-stage model, the laboratory class of thirty students is divided into four categories A, B, C, and D. Like the collaborative pair programming mode, the students first complete their different individual tasks. One student of category A and one student of category B then make pairs, and collaborate to modify, adapt, and integrate their individual work to complete a larger and more complicated task AB. Students of categories C and D also pair to complete another larger and complicated task CD. On completion of their individual tasks, the pair partners test each other’s work. If needed, the faster students can also help their partners after completing their own individual tasks. The members of these pairs are fixed for the entire semester, and they are advised to progressively evolve and follow their own coding guidelines through the semester. Finally, in each laboratory session, one AB pair collaborates with a CD pair to complete the final complex task that requires adaptation, modification, reuse, and integration of the work done for their individual and/or pair tasks. These partnerships between pairs are not fixed for the semester. On completion of their combined AB task, the fastest AB pair partners with the fastest CD pair after they have also completed theirs. Gradually, other pairs are also grouped into quadruples. The pairs that are not able to complete their pair tasks are not allowed to carry out the next level of quadruple task. Appendix A22 gives one such assignment for ‘J2EE,’ based on this model.

Our approach of collaborative pair programming has resulted in 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. We have also observed that paired laboratory experience is especially advantageous to inexperienced programmers. Another advantage that was evident from the students’ responses to the feedback sessions was that paired programmers were motivated to work collaboratively even outside the class, although this was not demanded or suggested by us. Consequently, we conclude that this form of collaborative pair programming positively influences all the dimensions of our competency taxonomy, and also does not suffer from the disadvantage of developing reluctance developed for solo programming, as was reported by some practitioners of regular form of pair programming.

Section 9.2.3.2: Cross-level Peer Mentoring in Software Development Education

As per our framework, Table 8.5, mentoring experiences gives the highest levels of active engagement. It also gives an opportunity to the mentors to review the work of others, giving experience of third level of collaborative engagement (Table 8.10). Further, it also creates conditions for integration (Table 8.8) and reflective engagements (Table 8.9) for mentors. Hence, in our view, mentoring offers a wholesome learning opportunity to the mentors.

During 2005 to 2008, a total of one hundred and sixty-four final year undergraduate students were engaged in mentoring their junior students’ laboratories as part of their formal assignment in ‘learning sciences’ or ‘theory of knowledge, learning, and research.’ They also correlated their real mentoring experience with various learning theories and proposed designs for e-learning systems for specific modules of host courses. In 2008, through the facilitation of software engineering course, a total of two hundred and five third-year students were engaged as project mentors for junior students’ mini projects.

In 2007-08, selected forty students of another fourth year elective course, software engineering management, were engaged to mentor juniors’ second year combined project in object-oriented programming and database management systems as part of their own activity: project management practice. In 2008-09, all two hundred students of this course group-mentored the third-year five credit minor projects. These final-year students mentored the juniors’ projects to build tools in diverse areas of software engineering. They submitted weekly mentoring reports.
As per the feedback received from the faculty of these courses, nearly 65-70% mentors provided good help to mentees. The three faculty members of the facilitating course software engineering management felt that mentoring assignment provided their students a better understanding of the role of human factors in software engineering, improved their project management, team management, leadership skills and also helped them to improve their problem understanding and problem solving abilities.

Based on our earlier positive experiences, very encouraging feedback from industry, and consultation with faculty members of the Department of CSE and IT, in 2009, more than three-hundred final year B.Tech (CSE) and B.Tech (IT) students were compulsorily engaged to mentor approximately fourteen hundred juniors’ laboratories and projects at any of the three lower years. Mentoring was considered as an integral part of their day-to-day work for mentors’ own year-long final year capstone project that is assigned more than 10% credit of the entire B.Tech. program. Nearly forty faculty members, who are also the project supervisors of these final year projects, agreed to keep 10 marks (out of the supervisor’s quota of 35 marks) earmarked for day-to-day work of the first semester of the final year.

**Multiple Benefits of Cross-level Peer Mentoring**

The feedback received from host faculty, facilitating faculty, mentee students, and mentor students during different stages of this scheme’s implementation has been positive. In 2007, a survey was conducted among the CSE and IT department’s faculty members. Twenty-six faculty members responded. More than 40% faculty members felt that this model of cross-level curricular peer mentoring significantly helped many students. Another 26% felt that it marginally helped many students, and the remaining were of the view that it was marginally helpful for few students. Most of them felt that it provided benefits to mentees as well as mentors.

In their opinion, mentees got benefits like increased level of instructional and doubt clearing help, increased opportunities for one-to-one out of the class help, improved programming skills, improvement in problem solving approach, and increased comfort level. The other benefits in their view included healthier cross-level relationships between cross-level students, and also
increased confidence of the mentors. Few faculty members also expressed their concern about the risk of increased spoon feeding of the juniors and discipline. Except for one, all other faculty members expressed their desire to continue the scheme.

In 2007, a feedback survey was jointly conducted through facilitating and host faculty among the second year students of a host course. Two hundred and seventeen students gave an average rating of 3.3 to more than forty final-year mentors based on the extent of help provided by them on a scale of 0 to 4. While the juniors felt the benefit of more easily accessible and friendly guidance, their mentors also reported several learning outcomes from this engagement: increased pride, and hence, enhanced motivation for more challenging work in their final year project, insights for leadership and project management issues, exposure to people related aspects in software engineering, handling quality and late delivery, and enhanced interpersonal skills.

As the seniors guide the juniors, and 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: experiencing 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. In the second semesters of 2007-08 and 2008-09, when the mentoring facilitating courses were not operational, many students of the final year, and also the third year, volunteered to mentor the juniors’ laboratories without any credit.

Mentors provide support in various ways. The mentors of “Introduction to Computer Programming” have reported to help their mentees in removal of syntactical errors, problem understanding, programming logic development, mapping it to programming language
constructs, debugging, providing study resources, helping, project formulation, etc. Some of them have attempted to work at a deeper level by trying to help their mentees to develop a better approach towards programming problems. More than 70% of these mentors have claim that in order to mentor, they have revised the old content of the host subject, and also learnt the new content that has been added for the juniors through self study. For example, nearly all mentors of the introduction to programming course revised their C language skills, and also learnt Python that has been recently introduced in this course. Every week, before meeting the mentees in the scheduled laboratory time, they prepare themselves well with mentee’s specific programming assignments. More than 70% responding mentors claimed to provide regular support to few of their mentees even outside the scheduled contact time. Some motivated mentors have taken some special initiatives like creating online communities of their mentees, regularly holding discussion with their mentees after the scheduled hours. Mentors are also discussing their mentee’s problems with other mentors. Some of their comments regarding their own learning gains through mentoring of juniors are given in Appendix A24.

Some of these students have felt that mentoring does well to productively engage their mind better than many other conventional education experiences like lectures, tutorials, and even written examinations. In their view, mentoring is specifically effective for engaging their mind in the following types of thinking:

Thinking required paying attention to minute details.

1. Thinking required learning application of some theory, concept, model, tool, procedure, or method.
2. Thinking required critiquing something, and also designing the criteria for the same.
3. Reflection upon personal and others’ experience/work/ideas to evaluate/improve it or to identify some pattern/model.

Reflections of Former Cross-level Mentors (Alumni)

To understand the learning gains of mentoring experience, the alumni of Jaypee Institute of Information Technology has been approached to give their feedback on their mentoring experiences of juniors’ laboratories and projects. This survey conducted by us 2009, is discussed in Appendix A22. The results of their feedback show that, in terms of its effect on all
competencies in our taxonomy, an overwhelming majority of responding alumni members who had got involved in mentoring during their undergraduate program perceived mentoring to be more/most effective as compared to other academic experiences. Many of them found that in comparison all other academic experiences, it was the most effective experience in terms of its effect on development of several competencies. The respondents felt its most significant effect on development of competencies like: accountability and responsibility, communication skills, and ability to accommodate self to others. Its positive effect on several other competencies is also significantly higher than several other academic experiences. These competencies include: curiosity with humility, attention to details, critical and reflective thinking, decision making skills, problem solving, creativity and innovation, and analytical/design/debugging skills. More than half of these respondents also mentioned that they are still in touch with their own erstwhile mentors.

**Reflections of Final year Cross-level Mentors**

*In another survey, conducted in 2009, among the more than three hundred final year mentors, an overwhelming majority of nearly 95% respondents have felt that mentoring juniors is resulting in their own multi-dimensional learning of various kinds that will be useful for their future career. Only 15% mentors did not find their mentoring experience to be useful with respect to their final year project. Around 70% of them considered mentoring experiences to be extremely, mostly, or many times useful in terms of its direct or indirect contribution of knowledge, skill, mindset, thinking, habits, problem solving methodology, etc., for their final year project. The mentors of second and third year level host courses considered it to be directly useful for their final year project. They have reported learning benefits like revision of the subject, sharpening of skills, and improvement of project planning and people related skills like understanding of multiple perspectives, listening skills, group work, leadership, etc. They also feel that it has increased their patience, empathy, sense of responsibility, etc. Some felt that this experience will help them in competitive examinations, placement interviews, or getting teaching assistantship during higher studies. Interestingly, some of them are very excited to discover their hidden teaching talent and interest.*
Why Does Cross-level Mentoring Benefit the Mentors?

As per the cognitive flexibility theory [206] revisiting a subject with different issue questions makes the learnt matter more easily transferrable to unfamiliar problem situations. Mentoring gives senior students an opportunity to revisit an earlier course from a different objective, higher level of maturity, and richer background of various other related courses. Mentoring juniors for their laboratories and projects gives a wholesome experience to the mentors. It engages them in rehearsal as well as elaboration of the host subject’s concepts, technical skills, and applications. The act of explaining the subject to juniors requires the mentors to create novel examples, analogies, and expressions. In addition to advising their mentees on doing their assigned problems, many motivated mentors often also design additional problems for them. The act of guiding them in project formulation, scoping, and design helps them to validate their own project experience in various courses. Mentors also often help the juniors in debugging, and some time marginally even in implementation. Many of them have felt that in terms of SOLO taxonomy, earlier they had usually approached the subject from a quantitative perspective with limited focus on inter-linkages between different concepts. The mentoring experience facilitated them to review the same subject from a qualitative perspective at relational level focusing on integrating varied concepts.

Mentoring very frequently creates cognitive dissonances [327] for the mentors. In the process of resolving these dissonances, mentors get engaged in reflection about the subject matter and also about their own thinking habits, attitudes, beliefs, and even values. This reflection created opportunities for deeper learning and transformation. Many mentors have reported that mentoring became their turning point. 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, and also leadership and decision making skills. Some faculty members have observed that sometimes even those students, who had not performed well in their course as regular students, in the later semester, take their mentoring task in the same course very seriously and do an excellent job.
Hence, we conclude that cross-level curricular peer mentoring has multi-dimensional effect on mentees as well as mentors. Instead of viewing it as a strategy to partially overcome faculty shortage for junior level courses, it should be viewed as a necessary educational experience for seniors that help them in enhancing several of their own competencies.

**Section 9.3: Reflective Workshop on Pedagogy for Engineering Faculty**

The author has also conducted some workshops for engineering faculty on effective teaching process. The experiences of one such workshop, ‘effective lecture,’ are briefly discussed here. It was conducted in 2004, for the faculty members of three engineering institutes. The session was attended by faculty members of varied experience, and diverse departments of science, engineering, and humanities. At the beginning the workshop, the faculty members were asked to fill up a form to rate the importance (most important/important/not important) of 16 attributes of a lecture. After this few anecdotes collected earlier were shared with them. Then, they were requested to recall and briefly write their own anecdotes about the two most effective formal lecture classes attended by them a student. Then they were also required to recall their own most effective lecture classes as faculty members. They were required to mutually share their anecdotes within pairs. Subsequently, they were required to publically share some of these anecdotes. Faculty members showed a great enthusiasm to share their anecdotes. Finally, they were required to re-rate the same sixteen attributes. Fifty-four faculty members coming from different institutes, departments, qualification level and experience level exercised their option to give their responses to the author. Table 9.3 summarizes these responses.
Table 9.3: Comparison of pre- and post-workshop consolidated ratings by faculty

<table>
<thead>
<tr>
<th>Lecture Format attribute</th>
<th>Fraction of respondents who rated the attribute as most important at the beginning of the workshop (A)</th>
<th>Fraction of respondents who rated the attribute as most important towards the end of the workshop (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a. careful listening</td>
<td>20.37%</td>
<td>15.09%</td>
</tr>
<tr>
<td>b. explain textbook</td>
<td>1.85%</td>
<td>3.77%</td>
</tr>
<tr>
<td>c. seek on-the-spot clarifications</td>
<td>42.59%</td>
<td>60.38%</td>
</tr>
<tr>
<td>d. seek clarifications</td>
<td>18.52%</td>
<td>18.87%</td>
</tr>
<tr>
<td>e. problem solving</td>
<td>38.89%</td>
<td>60.38%</td>
</tr>
<tr>
<td>f. creative thinking</td>
<td>66.67%</td>
<td>83.02%</td>
</tr>
<tr>
<td>g. in-class-group-work</td>
<td>22.22%</td>
<td>60.38%</td>
</tr>
<tr>
<td>h. create conceptual designs</td>
<td>31.48%</td>
<td>69.81%</td>
</tr>
<tr>
<td>i. analyze presented information</td>
<td>64.81%</td>
<td>67.92%</td>
</tr>
<tr>
<td>j. communicate your creations to neighbor students</td>
<td>14.81%</td>
<td>30.19%</td>
</tr>
<tr>
<td>k. communicate your creations to the entire class</td>
<td>29.63%</td>
<td>41.51%</td>
</tr>
<tr>
<td>l. critique</td>
<td>12.96%</td>
<td>24.53%</td>
</tr>
<tr>
<td>m. evaluate</td>
<td>33.33%</td>
<td>39.62%</td>
</tr>
<tr>
<td>n. discover</td>
<td>57.41%</td>
<td>66.04%</td>
</tr>
<tr>
<td>o. real-life example</td>
<td>72.22%</td>
<td>73.58%</td>
</tr>
<tr>
<td>p. contemporary issues</td>
<td>31.48%</td>
<td>41.51%</td>
</tr>
</tbody>
</table>

The difference in the two ratings, collected at the beginning and end of this 90 minute session, are very significant. While at the beginning of the session, only 22% respondents considered in-class-group-work as the most important attribute of lectures, 60% respondents rated this attribute as most important towards the end of workshop. Similarly, the fraction of the respondents who rated in-class conceptual design as one of the most important attributes also increased from 31% to 70%. Significant enhancement in favor of other attributes of problem solving, creative thinking, on-the-spot seeking the clarifications, communicate with the neighbor, communicate to entire class, critique, discover, and contemporary issues can also be seen. No theories of education, pedagogy, or communication were discussed in this very short duration workshop of 90 minutes. With the help of this reflective workshop, a significant change in faculty’s thinking was measured. This experiment shows that properly designed reflective engagements can be highly effective for changing the attitude, beliefs, and/or values.
Section 9.4: Chapter Summary

In this chapter, we have discussed several instructional interventions tried by us. All these interventions were administered in a chosen set of existing computing courses. Some new courses have also been developed in the process. Inquiry teaching has been tried out in some core courses. It was found 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 the 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. In collaboration with various faculty members, we continue to strengthen the infusion of eight elements: web technology, multimedia technology, mobile technology, security aspects, systems design aspects, estimation aspects, open source, and debugging in various introductory core computing courses. 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. A new form of collaborative learning have been proposed and tried out. A novel approach of collaborative pair and quadruple programming has been proposed. A novel form of collaborative learning, cross-level peer mentoring, has been evolved, tested, and scaled up. The results of sample tests were found to very encouraging.
We discussed our experience in conducting reflective workshops on pedagogy for engineering faculty. A significant shift in faculty’s beliefs about the active and collaborative learning was noticed. More work needs to be done in designing teachers’ training programs on pedagogy. We intend to use our framework to design many such workshops to motivate the teachers to use aspects of our framework in their teaching.

We also discussed the impact of many of these interventions in terms of feedback from students and alumni, and our experience in conducting a faculty development program. All these interventions are manifestations of some aspect(s) of the framework proposed by us in previous chapter. It may be noted that most of these instructional interventions were developed, refined, and administered during the course of this study before the development of the final framework proposed in the previous chapter. *Our experiences with all these interventions have helped a great deal in formulating the thought process for development of the framework.*