Senge, an eminent thinker on system thinking posited that true learning should be transformation of spirit and mind, not merely an accumulation of information or knowledge. According to Bigg’s 3P model (involving three stages of Presage, Process, and Product) [239], students’ perceptions of their learning environment, in light of their motivations and expectations, determine how situational factors influence their approaches to learning and learning outcomes. Research indicates that students’ learning strategies, academic performance, understanding, and academic integration, are linked to their attitude and epistemological perspectives. These attitude and perspectives may either enhance or constrain the scope and nature of their learning [240-241]. These vary according to age, past performance, and contextual factors like home environment, pre-college schooling experience, college experiences, and educational level. Attitudes and perspectives also affect a professional’s motivation and ability to practice. Hence, it is most important to make efforts to help students to form enabling attitudes and perspectives.

The following list enumerates the recommended attitudes and perspectives especially with reference to the requirements of the profession of software development:

1. Curiosity
2. Decision making perspective
3. Systems-level perspective
4. Intrinsic motivation to create/improve artifacts

In the following sections we discuss the rationale as well as explore theoretical and empirical grounds of these traits in multiple disciplines.
Section 6.1: Software Developers’ Education for Development of Curiosity

Importance of Curiosity for Software Developers

In our 2009 survey on required competencies for software developers, twenty software professionals assigned ‘curiosity’ an average rating of 3.15 on a scale of 0-4. An overwhelming majority of 90% of these respondents recommended it to be a critical or very important competency with respect to the requirements of software developers' multi-faceted professional activities.

With reference to Appendices A2 and A3, curiosity of software developers also relates to the following:

1. Interest in ‘how things work’ and ‘how to create things that work,’ interest in the power of technology,
2. Ability to see things as they are, observation skills
3. Broader understanding and interests
4. Respect for the classic authors of the great books
5. Openness to constructive criticism
7. Active listening skills
8. Ability to develop a very good understanding of domain specific vocabulary, its semantics, and established thinking patterns
9. Experimentation skills
10. Knowledge of contemporary issues and business practices
11. Knowledge of physical and natural world. Intercultural knowledge
12. Mentoring
13. Research skills
14. Self-awareness
15. Inclination for verification and validation, respect for facts and data

Curiosity is a highly important trait for software developers, and they need to proactively remain ready and engaged in lifelong learning because of following reasons:
The applications domains are highly diverse and continuously evolving, consequently software developers have to continuously learn more about these domains, mostly through self-learning, and work experience often without any long term formal education in the concerned domains.

Various technological innovations and changing social trends are continuously and rapidly reshaping user expectations, understanding these continuously evolving expectations is very crucial for software developers.

The development technology and platforms keep changing constantly often without proper documentation and examples, hence, the developers need to explore the useful enhancements and changes themselves again usually without much formal training.

The developers usually have to understand other developers’ work in order to extend, debug, maintain, integrate and/or re-engineer it.

Creation of “simple and idiot-proof system interfaces” requires them to be curious about how an average person approaches technology, and

There can be unintended consequences and risks of creating software inappropriate or at odds to its real purposes.

Metzger recommends exercising curiosity during the debugging process to locate other defects of the same root cause, and also defects of other kind [157].

In olden days the software developers main focus was on learning how computing systems work so that they can be efficiently utilized for meeting well known and understood computing needs. Fast and reliable internet access, multimedia rich client, and mobile computing have opened new possibilities for exploring hitherto unknown computing needs. Hence, today’s software developers need to be deeply interested in learning not only about the power of information and software technology, but also needs and even possibilities of human beings.

Expanding user expectations, changing user processes, evolving domain knowledge and understanding of users’ needs, growing power of information technology, and rapid transformation of development platforms make software development a highly iterative and
evolutionary process. In order to properly respond to these factors, the developers need to have open mindedness. Their work needs good observation skills and strong ability to see things as they are. Hence, software developers also need to have broader understanding and interests. Only highly curious software developers are able to develop very good understanding of domain specific vocabulary, its semantics, procedures, and established thinking patterns.

Active listening is crucial for requirement analysis and all other form of knowledge sharing with various stakeholders in the process of software development. Only a curious mind can be an active listener.

**Some Theoretical Perspectives on Curiosity**

**What is Curiosity?**

David Hume explained curiosity in his *Treatise of Human Nature* as “that love of truth, which is the source of all our enquiries.” Brand interpreted Hume’s work on curiosity [242]. Benedict [243] viewed curiosity as a sign of the rejection of the known as inadequate. Further, she posits that curious people seek and manifest new realities and reshape their own identities and their products. Reio and Callahan view curiosity as a state of emotional arousal, induced by a conceptual conflict or uncertainty that induces information seeking or exploratory behaviors to relieve the uncertainty. It results in the restructuring of knowledge structures or learning [244]. Annexure AN8 gives some more important theoretical perspectives on curiosity: Arnone [245] and Peterson et al [246].

While curiosity is a state commonly experienced by all people, it is also a trait which is much more typical of some people than others [247]. With reference to the importance of curiosity, Einstein said, “I have no special talents. I am only passionately curious.” Curiosity increases learners’ attention. Curious people can challenge their views of self, others, and the world with an inevitable stretching of information, knowledge, and skills. Curiosity is closely associated with love for learning which is necessary for systematically mastering new skills and bodies of knowledge through formal education or self-learning.
Curiosity is a fundamental motivational component for all openness facets including openness to experiences and open mindedness. Open mindedness involves multi-perspective thinking and suspension of judgment. Only in the state of open-mindedness one is able to recognize one’s misconceptions and the limitations of one’s knowledge. Curiosity gives one the ability to weight all evidence with fairness, and if required, change one’s mind in the light of new evidence. It is recognized as a source of critical thinking and also creativity.

Curiosity stimulates an inquiry within the existing framework that leads to acquisition of more information. A higher level of curiosity can also stimulate an inquiry about the framework itself, and results in evolution of perspective. At such level, a curious mind can get engaged in evolving a larger meaning in life beyond the immediate and short term interests of the self. Research suggests that curiosity is an important process for psychological well-being [248].

Diversity of Curiosity
A reinterpretation of Anderson and Krathwohl’s taxonomy of knowledge types [134], Carson’s taxonomy of knowledge types [249], and also Gardner’s theory of ‘multiple intelligence’ [155] help us to understand the variations of the curiosity of different persons. Depending upon their interest and abilities, persons may have their strengths or weaknesses with respect to the categories of all these classification systems. From the perspective of Anderson and Krathwohl’s taxonomy of knowledge types, persons may differ with respect to their (i) factual curiosity: inquisitiveness about factual knowledge, (ii) conceptual curiosity: inquisitiveness about conceptual knowledge, (iii) procedural curiosity: inquisitiveness about procedural knowledge (mental and psychomotor), and also (iv) meta-cognitive curiosity: inquisitiveness about meta-cognitive knowledge. Software developers need to have curiosity of all these types. In addition, in order to develop useful software, they also need to have a high level of contextual curiosity: inquisitiveness about the evolving context and expanding context of software technology.

Using Carson’s taxonomy [249] as the lens to differentiate between different types of curiosities, we can see the categories of (i) empirical curiosity: inquisitiveness about the environment and experiences, (ii) rational curiosity: inquisitiveness about abstractions, relations, and quantities, (iii) conventional curiosity: inquisitiveness about manmade conventions e.g. language, notation,
protocol, rule, law, standard, guidelines, procedures, etc., (iv) conceptual curiosity: inquisitiveness about concepts, theories, patterns, design, (v) cognitive curiosity: inquisitiveness about mental procedures, algorithms, heuristics, (vi) psychomotor curiosity: inquisitiveness about body control, (vii) affective curiosity: inquisitiveness about emotional and aesthetic aspects, (viii) narrative curiosity: inquisitiveness about understanding human condition with human perspective, and (ix) spiritual curiosity: inquisitiveness about the spiritual (not to be confused as religious) side of human experience and life. All these curiosities, except the last one, are beyond any doubt highly relevant to software developers’ work. It can also be argued that spiritual curiosity helps in overall growth of any person and helps them to understand larger purpose and meaning of life, which helps them to deal with work related dilemmas and issues of responsibility.

Levels of Curiosity
Epistemological beliefs of the learner about ‘what is knowledge’ and ‘what are the roles of a learner, teacher, and peers in the learning process’ influence their curiosity as well as learning process. In 1970’s, Perry [250] proposed a nine stage model of cognitive and moral development. The initial five stages are purely cognitive, whereas ethical aspects also get integrated in the later four stages. These nine micro level stages are also broadly grouped into four macro level stages. At the level of ‘dualism,’ people believe things are right or wrong and have faith and commitment to truth and knowledge as stated by genuine authorities. At the second macro level stage of ‘multiplicity,’ the diversity in thinking is recognized, but the person does not feel the need to commit to any specific belief or mode of thinking. The third macro-stage is ‘relativism.’ At this stage, the person sees the context sensitivity of knowledge. The final macro-stage is ‘commitment,’ at which the learners feel the need to take positions and commit to them.

As per Perry’s model, the movement through this stage is not automatic and progressive. One can undergo a long term pause at some position, or escape the progression by developing competence in some specific field, or even regress to lower position without one’s awareness. Felder and Breta [251], as well as West [252], provide a comparison between Perry’s model and

We have re-interpreted Perry’s model of intellectual and ethical development as a nine stage model of development of curiosity. Our re-interpretation is given in Table 6.1.

Table 6.1: Re-interpreting Perry’s nine stage model of intellectual development as nine stage model of curiosity development

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dualism</strong></td>
<td></td>
</tr>
<tr>
<td>1. Basic Dualism</td>
<td>Persons at this cognitive level believe that right solutions (knowledge and also values) to all problems are already known to ‘genuine’ authorities. Their curiosity is limited to learning right and specific solutions (facts and formulas) from authorities.</td>
</tr>
<tr>
<td>2. Full Dualism</td>
<td>Persons believe that solutions to all problems are already known to authorities, but some ‘genuine’ authorities may differ. Their curiosity is even more strongly focused on learning only the right and specific solutions from authorities by ignoring all other perspectives.</td>
</tr>
<tr>
<td><strong>Multiplicity</strong></td>
<td></td>
</tr>
<tr>
<td>3. Early Multiplicity</td>
<td>Persons believe that all problems are solvable. Further they think that even if ‘genuine’ authorities do not know the solutions to all problems, they know the right ways to find the correct solutions. Their curiosity expands to learn the right concepts and specific procedures, and ways of finding the correct solutions from authorities.</td>
</tr>
<tr>
<td>4. Late Multiplicity</td>
<td>Persons believe that some problems are unsolvable. Their curiosity is expanded to know what different experts say about such problem. However, they believe that one can choose any solutions for such problems as per one’s choice because there are no non-arbitrary bases to determine what is right.</td>
</tr>
<tr>
<td><strong>Relativism</strong></td>
<td></td>
</tr>
<tr>
<td>5. Contextual Relativism</td>
<td>Persons believe that all solutions must be evaluated in context and relative to their support by real evidence and logic. Their curiosity expands to learn to differentiate between weak and strong evidence, and to learn the analytic methods to evaluate solutions in the light of context, logic, and evidence.</td>
</tr>
<tr>
<td>6. Pre-Commitment</td>
<td>Persons start to see the need of integrating intellect with ethics for finding solutions in a contextual relativistic world. Their curiosity expands to learn to explore alternatives in open-ended problem solving, to make judgments based on personal and articulated standards, and be open to changing circumstances. However, persons at this level do not yet well consider or feel committed to their standards.</td>
</tr>
<tr>
<td><strong>Commitment</strong></td>
<td></td>
</tr>
<tr>
<td>7. Initial Commitment</td>
<td>Person makes actual commitments in personal directions and values as standards for open-ended problem solving and decision making.</td>
</tr>
<tr>
<td>8. Challenge to commitment</td>
<td>Persons experience the implications of their chosen commitments and standards, and also explore the issues of responsibility. Their curiosity expands to learn to evaluate the consequences and implications of their commitments, and to resolve conflicts.</td>
</tr>
<tr>
<td>9. Developing commitments</td>
<td>Persons develop a sense of self in both commitments and style and realize that commitment is an evolving activity. Their curiosity expands to learn to evolve and unfold their commitments in an ongoing manner.</td>
</tr>
</tbody>
</table>

**Does Education Arouse Curiosity?**

Interpreting Hume’s view on curiosity, Brand concludes [242] that *education is not so much about imparting the content of an inquiry, but has more to do with inquiry, process, activity, and finally a sense of pride that comes from ownership*. He also observed that unlike content,
curiosity is elicited rather than imparted. He quotes Hume, “What is easy and obvious is never valued; and even what is in itself difficult, if we come to the knowledge of it without difficulty, and without any stretch of thought or judgment, is but little regarded.”

Commenting on the inadequacy of modern education methods to promote curiosity, Einstein said, “It is, in fact, nothing short of a miracle that the modern methods of instruction have not entirely strangled the holy curiosity of inquiry.” Studies show that most under-graduates enter college at a Perry level 3, and graduate at a level 4 showing an average advancement only by 1/3 of a unit on a nine-point scale in four years [253]. Longitudinal studies have also shown that in the first three years, there is not much forward movement in the engineering students’ level as per Perry’s nine stage mode [254-255]. To understand the reasons of this phenomenon, in 2005, we carried out an empirical study through a detailed questionnaire on nature of questioning in the class. Twenty-nine undergraduate students of computer science and engineering and information technology gave their responses. A summary of their responses is given in Appendix A9.

Appendix A9 suggests that usually the classroom teaching is not oriented towards arousing or raising the level of curiosity. Consequently, we posit that higher education must motivate students to raise the levels of their curiosity on this hierarchy. Hence, Perry’s model (Table 6.1) is included in our proposed framework of pedagogical engagements in software development education (ref: Section 6.5 and Table 8.2, second column).

Enabling and Inhibiting Factors
Peterson et al [246] give an overview of the research on enabling and inhibiting factors that influence curiosity. Novelty, complexity, uncertainty, and conflict may work both ways depending upon the person’s appetite. Arnone [245] identified six instructional elements that can arouse curiosity: incongruity, contradictions, novelty, surprise, complexity, and uncertainty. Arnone [245] also suggested some instructional strategies for fostering curiosity among students.

Peterson et al posited that perceived probability that the knowledge is attainable, and perceived probability that personal resources can be expanded by integrating new knowledge, determine
the level of curiosity. The fueling factors also include increased knowledge and awareness of knowledge gaps in areas that are personally meaningful and engaging [246]. Impediments include anxiety, overconfidence, excessive self-focused attention, dogmatism, low cognitive resources, internal pressures like guilt and fear, external pressures like threat, punishment, and tangible rewards or pathological conditions. These suggestions are embedded in our proposed framework (Section 8.2.1).

**Pedagogical Implications**

In our recently concluded survey, “Software developers - (How) Did your college help you in your development?” (Table A10.2 (iii), Appendix A10), a large fraction of 66% felt that as compared to all other kind of academic engagements, their student projects did much better to develop their ‘curiosity.’ This was followed by research literature survey (62%), thinking oriented lectures (42%), laboratory work (38%), discussions with faculty, discussion with peers, and industrial training (36% each), and mentoring juniors (32%). Written examinations and discussion with others were found to be least effective in this regard by these respondents.

Given the nature of the problems, software developers need to solve, computing students need to be repeatedly engaged in asking questions like: (i) What (else) can be technology enabled? (ii) How can we do this using available and forthcoming technology? (iv) What resources are needed? (v) Is this approach efficient, effective, and/or appropriate? (vi) What is the scalability and sustainability of this approach? (vi) What are unintended consequences and risks? Further, negatively phrased question (why not?...) are also equally important.

Hence, in order to arouse curiosity of various kinds and at various levels, the education process has to be necessarily made student-centric where they learn to ask variety of questions as per Anderson and Krathwohl’s taxonomy, Carson’s taxonomy, and also Perry’s levels. Repeated and continued engagement in challenging questions, open-ended problem solving and projects, collaboration, community work, mentoring, etc., offer such opportunities. Finally, in order to continuous develop and evolve students’ intellect, and develop their zeal for excellence and elegance, their curiosity in great works of literature and arts should also be developed.
Section 6.2: Software Developers’ Education for Development of Decision Making Perspective

Importance of Decision Making

Decision making is about *choosing intelligently among less than perfect possibilities.* Professional decisions are broadly classified in three categories according to their scope: (i) **strategic decisions** concern general direction, long term goals, philosophies, and values; least structured and most imaginative; most risky and with most uncertain outcome, (ii) **tactical decisions** support strategic decisions; tend to have medium range, medium significance, with moderate consequences, (iii) **operational decisions** support tactical decisions; are structured and often made with little thought; impact is immediate, short term, short range, and usually low cost; can be preprogrammed, pre-made, or set out clearly in policy manuals.

With respect to software development, the developers broadly need to take decisions on two issues:

(i) What is to be (to visualize the product) and

(ii) How to deliver what is certain to be in the product.

These can be viewed as **product decisions, project decision, and process decisions.** In this context, it is very important that operational decisions in all these categories are consistent with tactical and strategic decisions.

**In our 2009 survey** on required competencies for software developers, twenty software professionals assigned decision making skills an average rating of 2.85 on a scale of 0-4. A large majority of 75% of these respondents recommended decision making to be a critical or very important competency with respect to the requirements of software developers' multi-faceted professional activities. Recently, we also concluded a poll among software professionals that was conducted for over one year. The respondents were asked to identify the weakest area addressed by engineering education in computing related disciplines. The offered choices were conceptual knowledge, decision making ability, learning ability, procedural knowledge, and thinking ability.
With reference to Appendices A2 and A3, decision making competence of software developers also relates to the following:

1. Perseverance and commitment
2. Integrity and authenticity
3. Accountability and responsibility
4. Project planning and management, project scoping, estimation, process planning and management
5. Entrepreneurship
6. Persuasion and negotiation skill
7. Sense of urgency and stress management

Fifty-eight professionals responded to our poll. Around 30% of the respondents’ age was above 35 years, and around 50% were in the age group of 25 to 34. 65% of the respondents worked for large or enterprise size organizations and remaining worked for small or medium size organizations. Responsibility allocation among the respondents varies as 64% in engineering, 12% in consulting, 8% in academics, and 4% each in creative, marketing, and operations. The distribution of their choices is as follows:

- Decision making ability: 41%
- Thinking ability: 24%
- Procedural knowledge: 15%
- Conceptual Knowledge: 15%
- Learning ability: 3%

A very large fraction of the responding software professionals consider that the weakest contribution of engineering education in computing related disciplines is in the area of developing students’ decision making ability.

Decision making requires a decision making perspective which is complementary but independent of intrinsic motivation to create/improve artifacts, curiosity, and systems level perspective. It is done in the light of one’s personal as well as organization’s values, and is highly affected by one’s sensitivity and awareness of socio-economic and other broader concerns. Decision making perspective requires taking decisions based on information and
evaluation of alternatives against objectives. In order to strengthen students’ decision making ability, software development education has to sensitize its students to multi-dimensional aspects and also some well known techniques of decision making.

The decision making process requires software teams to blend short term as well as long term perspectives. Long term perspective focuses on sustainability that includes concerns for stability, efficiency, and scalability. Often senior management is found expressing their concern about new software developers’ tendency to rush the problem by making a solution that addresses the operational problems of the customer without looking for strategic solutions.

**Decision Deficiencies**

Salas and Klein [256] have identified *five forms of decision deficiencies*: (i) *aim deficiency* occurs when a decision fails to meet a decision makers explicitly stated aim, (ii) *need deficiency* occurs when a decision maker fails to meet the actual need(s) in a given situation, (iii) *aggregate outcome deficiency* occurs when, collectively, all the outcomes of a decision (even beyond aim and need) leave the decision maker worse off than some effective reference, (iv) *competitor deficiency* occurs when, in aggregate, a decision is inferior to some competing alternative, and (v) *process cost deficiency* occurs when the cost of arriving at a decision is very high. *This model is included in proposed framework of engagements (Table 8.6b).*

**Some Theoretical Models about Decision Making**

Elaborating upon the social and creative dimension of decision making, Allwood and Selart [257] have emphasized on the importance of *restructuring the decision task through many iterations* of problem redefinition as well as goals, and also reformulating the problem space, seeing the broader context. *Their suggestion about iteration is integrated in our proposed framework (Table 8.5).* They also recommend viewing decision making as a synthesis rather than analysis activity, and insist on building a more holistic relational mental model. To generate alternatives, they recommend generating a large pool of alternatives by focusing attention on more unusual aspects of problem situation. As per them the alternatives need to be evaluated by integrating intuition and insight with logic and analysis.
Ullman posits that in real-life there are no right decisions but only satisfactory decisions. *Decision making is about finding the best possible satisfactory decisions* [258]. He posits that it is not an event or an action, but a process of repeatedly finding out what-to-do next. He has defined robust decision as the best possible choice, one found by eliminating all the uncertainty possible within available resources, and then choosing with known and acceptable levels of satisfaction and risk. He also posits that decision management is determining what-to-do-next with the available information in order to make most robust decision as part of standard work processes, and documenting the results for distribution and reuse. He has attributed information uncertainty to factors like knowledge limitations, incompleteness, approximations, viewpoint differences, terminology imprecision, inconsistency, and information’s evolving nature. He suggests that *decision making requires effort for uncertainty management* to make the best possible use of the uncertainty that cannot be eliminated.

Further, he has identified some *decision making challenges*. These are conflicting interpretations, conflicting priorities, incomplete understanding of the criteria of evaluation and risks of each alternative, and absence of good decision making strategy. In order to develop decision making competence, students need to be given practice in decision making through such challenging situations. *Hence, we include these challenges in our framework of pedagogical engagements (Table 8.6b).*

Seyedjavadein and Fahimi have recommended the use of TRIZ principles, cited in section 5.3, to generate alternatives during decision making [259]. As operational level decisions focus is on simplification and efficiency, the decision maker should seek the most *positive alternatives* which would add to the value of the system. For strategic level decision making, they recommend seeking positive *alternatives* which would add to the value of the system, while avoiding the *threats* to the system. For safety level decisions, seeking potentially negative alternatives which would damage the system is important in order to prevent them. For security level decisions, one needs to seek the *most negative alternatives* which would damage the system seriously in order to prevent them at any cost.
**Taxonomy of decision making**

Rowe and Boulgarides [260] have designed a two-dimensional taxonomy of decision making styles with respect to management education. They have identified four different styles of decision making. The four styles differ from each other mainly in two dimensions: (i) need for high structured-ness vs tolerance for ambiguity, and (ii) focus on technical aspects vs focus on people and their needs. The summary of these four styles is given in Table 6.2. *With reference to decision making in software development, we posit that the software developers need to integrate these styles. This perspective is included in our proposed framework of pedagogical engagements in software development education* (Ref: Section 6.5 and Table 8.5).

<table>
<thead>
<tr>
<th>Table 6.2: Four decision styles proposed by Rowe and Boulgarides</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>i. Directive style:</strong> This style is characterized by a low tolerance for ambiguity and rational way of thinking. It uses limited data and considers limited alternatives. This style is good for such technical issues that require lower cognitive complexity and have short range impact. It is especially suitable for implementing operational objectives by using rules and procedures in a systematic, efficient, and satisfactory way. It is more suitable for seeking acceptance and avoiding conflicts.</td>
</tr>
<tr>
<td><strong>ii. Behavioral style:</strong> This style is characterized by a low tolerance for ambiguity and intuitional way of thinking. Like directive style, it also uses limited data and considers limited alternatives. This style is good for such people related issues that require lower cognitive complexity and have short range impact.</td>
</tr>
<tr>
<td><strong>iii. Analytic style:</strong> This style is characterized by a high tolerance for ambiguity and rational way of thinking. It involves consideration of large amount of data from multiple sources, and evaluation of multiple alternatives. This style is suitable for such challenging technical issues that require focus on long range, and creativity. Analytics decision making is particularly useful for situations that require significant effort of analysis, planning, and forecasting.</td>
</tr>
<tr>
<td><strong>iv. Conceptual style:</strong> This style is characterized by a high tolerance for ambiguity and intuitional way of thinking. Like analytic decision style, it also involves consideration of large amount of data from multiple sources, and evaluation of multiple alternatives. This style is suitable for such challenging people related issues that require focus on long range, and creativity. Conceptual decision making is particularly useful for situations that require exploring new options, initiating new ideas, forming new strategies, being creative, taking risks, people oriented-ness, and ethical considerations.</td>
</tr>
</tbody>
</table>

Becker and Connor [261] have found that immediate gratification values are significantly related to the tendency to use a directive decision-making style. *Delayed gratification* values are related to a preference for a conceptual style. *Competence values* are related to a directive style, while *conscience values* are related to a behavioral decision-making style. *Self-constriction values* (self-controlled, responsible, logical, and obedience) are related to a behavioral style, while *self-expansion values* (broad-minded, cheerful, and imaginative) are related to a conceptual decision-making style.
In a laboratory experiment, software project managers with directed and analytics decision making style were found to respond better to performance measure criteria of time to complete project plan, completeness of initial project plan, and variances in a project plan, and scope change in a project plan [262]. Behavioral style decision making responded better to change of end date in a project plan.

The traditional engineering education model strengthens directed and analytics styles which are apt for taking decision regarding how to deliver what is certain to be in a product. However, with respect to taking decisions regarding visualizing and defining the product to be, conceptual style has to be strengthened. The evolutionary approaches to software development share many similarities like people orientation, openness, trust, and shared goals with conceptual style of decision making.

With reference to decision making in software development, we posit that the software developers need to be able to integrate the four decision making styles identified by Rowe and Boulgarides, Table 6.2. Hence, we include this in our proposed framework of engagements (Table 8.5)

**PrOACT and PROACTIVE approaches**

Hammond et al have created a framework for effective decision making, PrOACT (Problem, Objectives, Alternatives, Consequences, and Trade-offs) [263]. As per PrOACT, decision making consists of eight elements – formulating the problem in terms of its context and essential elements, clarifying key objectives with priority to serve as decision criteria, creating alternatives using creative thinking, identifying consequences with accuracy and completeness, clarifying trade-offs, uncertainty, risk tolerance, and linked decisions. The last three elements are not necessarily involved in all situations, but help clarify the decisions in volatile and evolving situations. Since, software development is a highly evolving situation, these aspects become very important for software developers’ decision making. Hammond et al view decision making as a multidimensional task with analytical, psychological, social, cultural, and intuitive processes.
Hunink has extended PrOACT model for medical decision making in the face of uncertainty and resource constraints [264]. As per this extension, PROACTIVE approach includes: defining the Problem, Reframing the problem from multiple perspectives, focusing on the Objectives, expanding the Alternatives, considering the Consequences and associated chances for each alternatives, identifying the Trade-offs involved, Integrating the evidence and values, optimizing the Value of interest, and Exploring uncertainty. We consider it propose that it can be helpful software development related decision making. We strengthen our proposed framework of pedagogical engagements with this model (Ref: Table 8.6b).

**Decision Oriented Model of Software Processes**

Toffolon and Dakhi [265] have proposed a decision oriented model of software processes. As per this model, the software development decision making is taking decisions with respect to four subspaces related to software projects: (i) problem space, (ii) solution space, (iii) construction space, and (iv) operation space. The decisions in these four spaces are driven by two broad categories of purposes:

(i) decisions to manage complexity and risk, the two essential characteristics of software, and

(ii) decisions to reduce the negative impacts of two kinds of accidental characteristics of software, i.e., uncertainty and complications.

This model is included in proposed framework of engagements (Table 8.6b).

**Decision Making for Risk Management in Software Projects**

Risk has been viewed as the probability of suffering losses while pursuing goals due to factors that are unpredictable or beyond [266]. Risks can be internal or external. Internal risks arise because of inadequacies in process capability (including core and support functions), and organizational structure. External risks are caused by uncertainties in external conditions. Risk management requires a systematic approach of reducing the harms due to risks, making the project less vulnerable and product more robust. It is very important aspect of decision making.

Boehm [267], one of the pioneers of software risk management field described it as comprising of two functions (i) risk assessment: identification, analysis, and prioritization; (ii) risk control:
management planning, resolution, and monitoring. SEI’s has also identified six elements of software risk management: identify, analyze, plan, mitigate, track and communicate.

**Importance of Risk Management in Software Development Education**

In one of our recently concluded survey among software professionals, fifty-seven professionals responded. The respondents professional experience distribution is as follows: (i) around 15% with more than 15 year experience, around 10% with between ten to fifteen years of experience, around 40% have five to ten years of experience, and the remaining with less than five years experience. A good fraction of 36% respondents recommended that risk planning and mitigation must be included in the main goals for new curriculum for the future generation of software developers.

**Software Risk Categorization Schemes**

Boehm identified the top ten risks items. The top four in this list were personnel shortfall, unrealistic schedule and budget, wrong function and properties, and wrong user interface. According to Brian A Will, the top most risks include creeping software requirements, requirement gold plating, low quality of released software, and unachievable schedule.

Keil et al provided categorization framework for software project risks [268]. They categorized these risks into four quadrants. The first quadrant risks relate to customers and users. These risks have a high level of perceived importance but a low level of control possibility. Hence, mitigation is essentially done by increasing users’ participation and commitment to the software project. The second quadrant risks relate to ambiguities and uncertainties about scope and requirements. These risks have high perceived importance as well as a high level of control potential for project managers. The third quadrant risks relate to execution that has moderate perceived importance but high level of control is possible. The last quadrant risks relate to environment and have moderate perceived importance as well as low control possibility for project managers.

Wallace and Keil have further classified fifty software risks into these four categories [269]. They also analyzed the effect of these risks on process and product outcome. They have
concluded that for project managers, minimizing and managing the execution, scope and requirement related risks are critical from both perspectives. Further, they observed that in situations where product outcomes are more important than time and budget, the risks related to users and customers also become very critical.

SEI has proposed two taxonomies. First, they catalogued and classified one hundred and ninety-four risks into the three broad level categories [270] of product engineering, development environment, and program constraints. Later, SEI proposed another taxonomy for software development risks for high-performance computing scientific/engineering applications. This taxonomy classifies the sources of software development risks into the three broad categories of development cycle risks, development environment risks, and programmatic risks.

Pandian has given a distribution of software development risks. As per his analysis, 70% risks are internal and only 30% are external. Project risks account for 30%, product risks for 30%, and process risks for 40%. In the process risks, the most vulnerable areas are related to human resource and requirement issues, and least vulnerability is found to exist in coding and testing [266]. Georgieva et al have provided a survey of software risk assessment methods [272].

We include risk assessment for identifying, analyzing, and prioritizing project, product, and process risks in our proposed framework (Table 8.6a). We also recommend the use of SEI taxonomies as checklists.

**Ethical Decision Making**

Professional decision making ability is not only related to technical competence only. Instead of just technical competence, intelligence, or creativity, it is related to professional wisdom. Sternberg [229] has defined wisdom as the application of tacit as well as explicit knowledge, as mediated by values, towards the achievement of a common good through a balance among (a) intrapersonal, (b) interpersonal, and (c) extra-personal interests over the (a) short term and (b) long term to achieve a balance among (a) adaptation to existing environments, (b) shaping of existing environments, and (c) selection of new environments.
Boyle [273] has proposed a six-stage process of ethical decision-making for computing professionals. The first stage is about moral perception and personal knowledge of the moral good, which depends upon the ability to recognize that an ethical problem exists and that a person has some personal responsibility to respond. The second stage is of the moral discernment and personal ability to think logically, which enables a person to state the ethical problem clearly. The third stage is of moral resolution and personal ability to analyze complexities of the stated problem, in order to arrive at an individual position which is justifiable to one’s self conscience. The fourth stage is of moral assessment and personal ability to assess one’s freedom. According to Boyle, computing professionals must be aware of new developments, particularly in the context of the history of technology in the computing field, in order to handle the new freedoms properly. The fifth stage deals with moral decision and personal knowledge of one’s duties. The last stage is of moral action and personal willingness to follow one’s intellect.

IEEE-ACM code of ethics for software engineers provides directions and guidelines for all these stages, except the second and last stage. The second and third stage depends upon the critical thinking ability, and the last stage depends upon one’s value system. Boyle sees the entire process as circular, such that the moral actions of one cycle shape the moral perception for next cycle. We strengthen our proposed framework of pedagogical engagements with this model (Table 8.6b).

Pedagogical Perspective
In our recently concluded survey, “Software developers - (How) Did your college help you in your development?” (Table A10.2 (iii), Appendix A10), a large fraction of 77% felt that as compared to all other kind of academic engagements, their student projects did much better to develop their decision making skills. This was followed by laboratory work (38%), mentoring juniors and industrial training (35% each), thinking oriented lectures (31%), and discussions with other students (31%). Traditional knowledge delivery oriented lectures and written examinations were found to be least effective in this regard by the respondents. Further, 90% and 71% respondents respectively felt that as compared all other engagements, student projects and industrial training did much better to enhance their project planning and management skills.
All other engagements were felt to be ineffective in this regard. We can conclude traditional form of engineering education, misses the opportunity to develop students decision making thinking and perspective.

Students need to learn to take decisions related to product, process, and project. Such ability can be developed by developing their decision thinking perspective. Development of such perspective requires exposure to wider contexts, and also reflection on senior professionals’ decisions taken in tricky situations.

In addition, the education programs must also engage students in professional decision making in real-life like situations. Typical academic engagements like traditional lectures, short assignments, written examinations, and textbook oriented exercises do not create such engagement. Student-centric learning engagements like semester long group projects offer a great potential to give them opportunities to take and improvise their decisions.

Decision thinking is not automatic, but controlled thinking [274]. Students’ decision making ability can only be developed by developing their decision making perspective. In this section we have discussed some important models and tools that can help in developing their decision perspective. The decision oriented models suggested by (i) Toffolon and Dakhi as well as (ii) Boyle (with respect to IEEE-ACM code of ethics) are can be very useful for students. Students should be exposed to the decision deficiencies identified by Salas and Klein and decision challenges identified by Ullman. As suggested by Allwood and Selart, they must be required to iterate over the decision tasks. The student engagements should require them to integrate the four decision styles suggested by Row and Boulgerides. They must also be engaged in risk management of product, project, and process risks. SEI taxonomies can be used as checklists. We include these in our proposed framework of pedagogical engagements (Tables 8.5 and 8.6b).

Further, literature on decision making offers some excellent general purpose techniques that have been used in various professions. Some of these are: Pareto analysis, paired comparison, T-Chart, decision matrix, grid analysis, PMI (Plus, Minus, and Interesting), decision Tree, six thinking hats, star-bursting, step-ladder, and Delphi. All these techniques are essentially
manifestations of the core idea of decision making as a process of making choices. These involve generation of alternatives and evaluating their consequences. These and other such techniques can also be very effectively used by software developers for taking effective decisions during various activities of software development. Hence, computing students should be well exposed to these techniques through their curriculum. These techniques are used to support our proposed framework of pedagogic engagements/ software development education is on our future agenda (Ref: Table 8.6b).

Finally, as per our exploratory study of students’ software projects, we have found that normally student projects do not expose them with many typical risks in software projects. The most common risks in student projects are due to lack of their proficiency with development tools and/or open source, and unrealistic estimates. They do not get exposed to other typical software project risks discussed above. We hypothesize that the student projects also need to be viewed and administered from an additional perspective of exposing them to common real-life software project risks. Typical projects designed in protected academic setting often do not achieve this goal. In future extension of our work, we plan to carry forward this idea and propose an appropriate model of administering and designing student projects.

Section 6.3: Software Developers’ Education for Development of Systems-level Perspective

Merriam Webster's Collegiate Dictionary, tenth edition, defines ‘system’ as “a regularly interacting or interdependent group of items forming a unified whole.” This, however, is only a partial view of systems. Thinkers of various disciplines like engineering, management, science, economics, sociology, political science, etc., have contributed significantly to understanding systems as well as systems thinking. According to Meadows, a system is more than the sum of its parts, its part are simultaneously interconnected in multiple directions, it has a purpose, it produces its own behavior over time and its response to external triggers and/or forces is a characteristic of itself. In real-life these responses are usually very complex [275]. Checkland [275a] identifies the four classes of systems: natural systems, designed physical systems, designed abstract systems, and human activity systems.
“Systems thinking” is seeing wholeness, seeing interrelationships rather than individual things. Isolated knowledge by a group of specialists generated in a narrow field has no value in itself, only its synthesis with the rest of the existing knowledge gives it a meaning [277]. Solovey [279] found the eleven laws of system thinking proposed by Senge [277] to be applicable to software development. Annexure AN9 gives these laws.

**Importance of Systems Thinking for Software Development**

*In our 2009 survey on required competencies for software developers, twenty software professionals assigned ‘systems-level perspective’ an average rating of 2.95 on a scale of 0-4. A large majority of 85% of these respondents recommended it to be a critical or very important competency with respect to the requirements of software developers’ multi-faceted professional activities.*

Usually software programs are complex systems. They are executed on computing environments that are examples of complex systems. Software is usually a critical subsystem of a larger technical and/or organizational/social system. Further, the development life cycle of software is another example of a very complex social system. In the context of software development, holistic understanding of the problem and solution paves the way for a robust implementation. Metzger recommends the inclusion of gestalt understanding for debugging tasks [157]. As software-based systems have grown larger, more complex, and need inter-disciplinary inputs, the capacity of systems thinking has become crucial for software developers. Concerns like user-centeredness, reuse, integration with existing subsystems, legacy systems, quality, cost, security, availability, and maintainability make it imperative to develop systems thinking.

With reference to Appendices A2 and A3, systems-level perspective of software developers also relates to the following:

1. Ability to accommodate himself to others, empathy, “be the customer” mentality - genuine interest in understanding what other people are trying to accomplish and based on this understanding think about creating technical solutions to help them reach their goals. Genuine interest in understanding “why to create software” and the broader context of
software systems. Cognitive task analysis. Appreciation of unstated requirement and ability
to identify these. *Listening skills*, approachable, and respect for people. *Ability to work in
homogeneous, multi-disciplinary, multi-locational and multicultural teams*. Ability to work
under supervision and constraints, Understanding of the impact of personal character and
behaviors on others.

2 Ability to see the self as bound to all humans with ties of recognition and concern. Seek help
from other, *Ability to help and assist others, mentoring, commitment to others’ success.*
*Sensitivity towards global, societal, environmental, moral, ethical and professional issues,
and sustainability*. Respect for the intellectual property of others. Work ethics.

3 Organizational skills.

4 Quality, *cost*, and security consciousness, pursuit of excellence, intellectual *accountability
and responsibility*, intellectual *integrity*, intellectual courage, strength of conviction: assertive
without being aggressive. Commitment to systematic documentation of the work. Recognize
and act upon the need to consult other experts, especially in matters outside their area of
competence and experience. Commitment to the fulfillment of needs of all users and persons
who get affected by the technological solutions. Eagerness and inclination to understand the
unintended consequences of creating software inappropriate or at odds to its real purposes.
Commitment to health, safety, dignity, and welfare of the users and also the people who will
be affected by their systems. Sensitivity towards constraints like economic disadvantage and
physical disabilities that may limit software accessibility.

5 Self-acceptance, self-regulation, self-awareness, self-improvement: strength to resist instant
gratification in order to achieve better results tomorrow. Being honest and forthright about
one’s own limitations of competence. Tendency to avoid false, speculative, vacuous,
deceptive, misleading, or doubtful claims. Faith in reason and review, inclination for
verification and validation, respect for facts and data. Awareness and regulation of automatic
thoughts.

6 ‘Big picture’ view, holistic and multi-perspective thinking, knowledge integration,
consideration for multilateral viewpoint, and user-centeredness. Process and rule-oriented
mindset. Tolerance to ambiguity and risk. Ability to understand and also build upon other’s
work. Ability to work such that others can easily understand and build upon.

7 Perseverance and commitment.
8 Complex problem solving skills
9 Analytical thinking
10 Design skills

**Systems Engineering Perspective**

Frank and Waks [280] have given a multifunctional comprehensive definition and explanation of engineering systems thinking. Further they have also given the characteristics of engineers who are able to demonstrate such thinking. This definition links multiple options of seven facets. The definition is given in Table 6.3. We use this to strengthen our framework of pedagogical engagements (Table 8.6a).

<table>
<thead>
<tr>
<th><strong>Table 6.3:</strong> Multifaceted definition of engineering systems thinking by (Frank and Waks, 2001)</th>
</tr>
</thead>
<tbody>
<tr>
<td>The engineering systems thinking of a/an [facet A – specialization field] engineer who deals with a system of a [facet B - complexity level] level of complexity involves the ability to understand [facet C – systems aspects and implications] and the [facet D - interrelationship] [facet E- interconnections] and to [facet F – functional domain] without [facet G – constraints].</td>
</tr>
<tr>
<td>Options for [facet A – specialization field]: (i) electrical, (ii) electronics, (iii) computers, (iv) software, and (v) others</td>
</tr>
<tr>
<td>Options for [facet B- complexity level]: (i) very low, (ii) low, (iii) intermediate, (iv) high, and (v) very high</td>
</tr>
<tr>
<td>Options for [facet C – systems aspects and implications]: (i) understanding the whole system, (ii) understanding the synergy of the system, (iii) understanding the contribution of components of the system, (iv) understanding the system from multiple perspectives, (v) understanding the implications of modification to the system, and (vi) understanding a new system immediately upon presentation</td>
</tr>
<tr>
<td>Options for [facet D - interrelationship]: (i) interaction between, (ii) hierarchy of</td>
</tr>
<tr>
<td>Options for [facet E- interconnections]: (i) internal subsystems, (ii) external neighboring systems</td>
</tr>
<tr>
<td>Options for [facet F – functional domain]: (i) locate system failures, (ii) outline failure solution, (iii) analyze/dismantle system to individual components, and (iv) synthesize/design subsystems linkages to a whole</td>
</tr>
<tr>
<td>Options for [facet G – constraints]: (i) need to understand details for understanding the whole, (ii) refraining from multitasking, and (iii) “getting lost” when dealing with system issues or acting in a non-familiar professional environment</td>
</tr>
</tbody>
</table>

**Levels of systems thinking**

Sanford observed that our upbringing, and particularly our education, has trained our thought patterns to follow a segmented and reductionist path. The new capability to see and to think in terms of systems thinking also starts with being able to “envision” relationships and structural components of nested whole ways of thinking. There are *five levels of systems thinking: closed, cybernetic, complex adaptive, developmental, and evolutionary* [281]. We use the lower levels of Boulding’s levels (Annexure AN9, Table AN9.4) to extend this hierarchy as depicted in Table 6.4. We also merge the levels of closed and cybernetic systems into a single category. *This
modified ladder is integrated in our proposed framework of pedagogical engagements in software development education (Table 8.2, fourth column).

Table 6.4: Levels of systems thinking (derived from Boulding and Sanford)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Pre-structural thinking</strong>: seeks to understand, analyze, build, evaluate, and maintain the components</td>
</tr>
<tr>
<td>2</td>
<td><strong>Structural thinking</strong>: seeks to understand, analyze, build, evaluate, and maintain static structures and frameworks involving various components.</td>
</tr>
<tr>
<td>3</td>
<td><strong>Clockworks thinking</strong>: seeks to understand, analyze, build, evaluate, and maintain predetermined motion.</td>
</tr>
<tr>
<td>4</td>
<td><strong>Closed systems thinking</strong>: seeks to understand, analyze, build, evaluate, and maintain mechanisms, seek stabilization within tolerance and standards by allowing limited access and exchange with systems outside their boundaries in spite of richer interactions through a feedback based control.</td>
</tr>
<tr>
<td>5</td>
<td><strong>Complex adaptive systems thinking</strong>: seeks to understand, analyze, build, evaluate, and maintain effectiveness of open systems in the context of a continuously dynamic and evolving environment.</td>
</tr>
<tr>
<td>6</td>
<td><strong>Developmental systems thinking</strong>: seeks improvement by uncovering the full potential and expression of the unique essence of any entity or system, including the greater system of which we are a part. It involves re-conceptualization of the values by exploring the core value, core process, and core purpose interactively looking beyond themselves.</td>
</tr>
<tr>
<td>7</td>
<td><strong>Evolutionary systems thinking</strong>: this is generative field of evolving systems, it requires looking at the entire value chain and context and beyond what they serve.</td>
</tr>
</tbody>
</table>

**Shifting the Focus for Systems Thinking**

Capra [282] had proposed five criteria for systems thinking in natural sciences, as given in Table 6.5. The first two criteria refer to our view of nature’s complexity. In addition, the next three criteria refer to our epistemological beliefs and uncertainty.

Table 6.5: Shifting the focus for systems thinking (Capra’s criteria)

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Shift the focus from <strong>parts to whole</strong>, the properties of the parts can be understood only from the dynamics of the whole; part is merely a pattern in an inseparable web of relationships</td>
</tr>
<tr>
<td>2</td>
<td>Shift the focus from <strong>structures to process</strong>; every structure is a manifestation of an underlying process, the entire web of relationships is dynamic.</td>
</tr>
<tr>
<td>3</td>
<td>Shift from the <strong>objective science to epistemic science</strong>, the understanding of the process of knowledge has to be included explicitly in the description of natural phenomena.</td>
</tr>
<tr>
<td>4</td>
<td>Shift from <strong>building to network</strong> as metaphor of knowledge, there the material universe is seen as a dynamic web of interrelated events, there are no fundamental entities whatsoever: constants, laws, or equations, none of the properties of any part of this web is fundamental, they all follow from the properties of the other parts, and the overall consistency of their interrelations determines the structure of the entire web.</td>
</tr>
<tr>
<td>5</td>
<td>Shift from <strong>truth to approximations</strong>, all scientific concepts and theories are limited and approximate, scientists do not deal with truth but with limited and approximate description of reality.</td>
</tr>
</tbody>
</table>

**Blaauw’s Principles of System Architecture**

Blaauw [282a] has identified six more principles of good architecture. These include **consistency**, **orthogonality**, **propriety**, **parsimony**, **transparency**, **open ended-ness**, **generality**, and **completeness**. Deliberate usage of these principles for evaluation of software architectures can
help a great deal to improve them. *These are included to support our framework of pedagogic engagements (Table 8.6b).*

**Soft Systems Methodology for Solving Soft Problems**

The famous waterfall model of Structured Systems Analysis and Design Method (SSADM), is based on Checkland’s [284] Soft Systems Methodology (SSM) that was developed in late 1980’s for *solving soft problems*. *Soft problems are such problems that have multiple stakeholders with divergent values, beliefs, philosophies, interests, and also views about what the problem is.* This iterative approach consists of seven distinct stages given in Table 6.6. Recently, Jacobs [285] proposed an approach to applying systems thinking. This is also included in Table 6.6.

**Table 6.6:** Systems thinking approaches by Checkland and Jacobs

<table>
<thead>
<tr>
<th>Checkland’s Stages of Soft Systems Methodology for Solving Soft Problems</th>
<th>Jacobs’ approach for applying systems thinking</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Define and understand the problem situation (i.e., nature of the process, key stakeholders, etc.),</td>
<td>1 Explore the event/problem from multiple perspectives without jumping to solutions,</td>
</tr>
<tr>
<td>2 Express the problem situation through rich pictures,</td>
<td>2 Track the situation over a period of time and identify patterns and trends of behavior that go below the surface,</td>
</tr>
<tr>
<td>3 Select how to view the situation from various perspectives and produce root definitions,</td>
<td>3 Look for systemic structures such as interrelationships in the patterns and trends, balancing and reinforcing feedback, and delays, also understand the mental models that are driving these patterns, and</td>
</tr>
<tr>
<td>4 Build conceptual models of the system requirements to adequately address each of the root definitions,</td>
<td>4 Create new mental models to introduce change into the system, track and evaluate the effects of the changes, and identify unintended consequences and decide what needs modification.</td>
</tr>
<tr>
<td>5 Compare the conceptual models to the real world expression,</td>
<td></td>
</tr>
<tr>
<td>6 Identify feasible and desirable changes to improve the situation, and</td>
<td></td>
</tr>
<tr>
<td>7 Develop recommendations for taking action to improve the problem situation.</td>
<td></td>
</tr>
</tbody>
</table>

**Software as Socio-technical Systems**

The criteria identified by Capra and characteristics proposed by Sweeney and Meadows are highly relevant for software developers. In his system theory, Senge [286] argues that we often complicate the nature of the problem, because we tend to treat problems as if we are outsiders, rather than treating the problems and ourselves as one. He further posits that systems thinking also aims at integrating one’s insights into the inner systems and visions of the outer systems, and we need to transform the technical mode of working into the spiritual pursuit of work ethics. *As many software systems are socio-technical systems and the software development systems are*
essentially social systems, Senge’s perspective is even more relevant in the context of software development.

**In our 2009 survey** on required competencies for software developers, *twenty software professionals* assigned ‘accommodate oneself to others’ an average rating of 3.1 on a scale of 0-4. *A large majority of 80% of these respondents recommended it to be a critical or very important* competency with respect to the requirements of software developers' multi-faceted professional activities. In order to do cognitive task analyses, software developers need to have a genuine interest in understanding what other people are trying to accomplish, and based on this understanding think about creating technical solutions to help them reach their goals. In order to identify unstated requirements, they need to have a genuine interest in understanding ‘why to create software’ and also the broader context of software systems. Development of this kind of genuine interest requires the virtue of empathy as manifested in ‘be the customer/user mentality’ and tolerance to ambiguity. Often, software developers have to work in large teams of developers that are temporally and often geographically distributed and even culturally diverse. This requires the developers to have the ability to understand and also build upon other’s work and also the ability to work such that others can easily understand and build upon.

**In another survey** conducted us of fifty-seven software professionals (Table 4.1), *65% of our respondents included ‘group work, people management, and leadership’ as one of the most important activities that must be included in the main goals for new curriculum for the future generation of software developers. This ability also requires an attitude and ability to ‘accommodate oneself to others.’ Such ability also makes one more approachable.

**Ethical Aspects of Systems Thinking**

Hoffman saw empathy as the key to moral motivation [287]. **In our 2009 survey** on required competencies for software developers, *twenty software professionals assigned ‘see the self as bound to all humans with ties of recognition and concern’ an average rating of 2.65 on a scale of 0-4. A majority of 60% of these respondents recommended it to be a critical or very important competency with respect to the requirements of software developers' multi-faceted professional activities.
Commitment to the fulfillment of needs of all users avoiding unintended consequences on safety, dignity, health, and welfare of the users and also the people who will be affected by their systems requires them to have empathy and a commitment for accountability and responsibility. Deeper sense of responsibility comes from attitude and ability to ‘see the self as bound to all humans with ties of recognition and concern.’ This ability requires sensitivity towards global, societal, environmental, moral, ethical and professional issues and sustainability, as well as, respect for work ethics and intellectual property of others. The concern for sustainability requires sensitivity towards constraints like economic disadvantage and physical disabilities that may limit software accessibility. *Sternberg’s definition of wisdom, discussed under the theme of ethical decision making in section 6.2, is very relevant in this context.*

Such well grounded sense of responsibility can facilitate self regulation to resist instant gratification (in order to achieve better results) and also avoid false, speculative, vacuous, deceptive, misleading, or doubtful claims about their competence, products, and services. It strengthens intellectual integrity, intellectual courage, and gives a strength of conviction. This ability to see the self as bound to all humans when combined with the ability to accommodate oneself with other strengthen one’s ability to seek and provide help, participate as mentors/mentees/supervisor/supervise, have commitment to others’ success, and be assertive without being aggressive. All these are very important for successful group work.

**Cultivating systems thinking**

Senge [278] developed a toolbox for cultivating systems thinking. This are given in table 6.7.

<table>
<thead>
<tr>
<th></th>
<th>Learning how to draw systems maps, including</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>a. the interaction between cause and effect,</td>
</tr>
<tr>
<td></td>
<td>b. dynamic loop,</td>
</tr>
<tr>
<td></td>
<td>c. system feedback perspectives,</td>
</tr>
<tr>
<td></td>
<td>d. systems problems,</td>
</tr>
<tr>
<td>2</td>
<td>Learning how to describe reinforcing loops,</td>
</tr>
<tr>
<td>3</td>
<td>Learning how to describe balancing loops, and</td>
</tr>
<tr>
<td>4</td>
<td>Learning how to describe delays.</td>
</tr>
</tbody>
</table>
Software systems analysis and design techniques

Several semi-formal and formal techniques have been developed for software systems analysis and design. They offer powerful representation tools for data and behavior of software systems. Data representation techniques include conceptual data modeling techniques, knowledge representation techniques, ontologies, etc. Behavior representation techniques include FSM, State-chart, State Nets, Petri Nets, etc. We strongly recommend the frequent and repeated use of many of these semi-formal techniques in computing courses. We include these techniques to support our proposed framework (Table 8.6b).

Meta-Framework for Systems Engineering

Haskin [288] has proposed a meta-framework for systems engineering. The 6C’s in her framework are: Comprehension, Communication, Coordination, Cooperation, Collaboration, and Continuity.

Comprehension needs listening, empathy and broader general knowledge. She posits that these six C’s sit in the context of Code of ethics. Hence, we see that systems thinking require a higher maturity level of not only cognitive development, but also emotional and moral development.

Kohlberg [289] proposed a six stage model of human development based on their moral reasoning. Table 6.8 gives a summary of this model. This is part of our proposed framework of pedagogical engagements in software development education (Section 6.5 and Table 8.2, fifth column).

Table 6.8: Kohlberg’s six stage model of human development

| A. Pre-conventional level (Egocentric) (Self-centered) |
|---|---|
| 1. Obedience and punishment: the moral reasoning is motivated by avoiding anticipated punishment. |
| 2. Individualism and Reciprocity: the moral reasoning is motivated by self interest. |
| B. Conventional Level (Socio-centric) (Conservative) |
| 3. Interpersonal conformity: the moral reasoning is motivated by avoiding anticipated disapproval of others by ‘looking’ nice to them. |
| 4. Social systems and “Law and order”: the moral reasoning is motivated by avoiding anticipated dishonor or institutionalized blame and desire for maintaining social order. |
| C. Post-conventional (Onto-centric) (Progressive) |
| 6. Universal ethical principles: the moral reasoning is motivated by maintaining respect and dignity of all by emphasizing human values and rights. |
Moral development of a person is closely linked with the person’s value orientation. Spini [290] refers to Schwartz [291] who saw people’s values as their motivational constructs for deciding their actions. Schwartz value categories are discussed in Annexure AN9. Our exploratory survey of undergraduate computing students showed that, by and large, the responding students felt that most of their peers lacked the values of self-direction, benevolence, and universalism. However, a more systematic study is required on this aspect. Nevertheless, development of benevolence and universalism is crucial for developing the abilities to ‘accommodate oneself to others’ and ‘see the self as bound to all humans with ties of recognition and concern.’ Both these abilities are identified as key aspects of systems thinking. Further, development of self-direction is very important for arousing the intrinsic motivation to create/improve artifacts (discussed in the next section). Hence, the development of these values has to be addressed by software development education.

**Pedagogical Perspective**

In our recently concluded survey, “Software developers - (How) Did your college help you in your development?” (Table A10.2 (iii), Appendix A10), a large fraction of 68% felt that as compared to all other kind of academic engagements, their student projects did much better to develop their ‘systems-level perspective.’ This was followed by research literature survey (46%), laboratory work (34%), and industrial training (32%). Written examinations were found to be least effective in this regard by these respondents.

In this survey (Table A10.2 (iii), Appendix A10), a large fraction of 64% felt that as compared to all other kind of academic engagements, their student projects did much better to develop their ‘accommodate oneself to others.’ This was followed by mentoring juniors (56%), discussions with other students (46%) and industrial training (38%). Written examinations and lectures were found to be least effective in this regard by these respondents.

Further, in the same survey (Table A10.2 (iii), Appendix A10), a large fraction of 51% felt that as compared to all other kind of academic engagements, their student projects did much better to develop their attitude to ‘see the self as bound to all humans with ties of recognition and concern.’ This was followed by mentoring juniors (45%), industrial training (41%) and
discussions with faculty, students, and others (33%, 31%, and 29% respectively). Written examinations and traditional form of knowledge delivery oriented lectures were found to be least effective in this regard by these respondents.

In this section, we discussed several models and tools that can be used to develop systems-level perspective of computing students. We derived a new ladder of systems thinking based on Boulding and Sanford ladders. Capra’s suggestion for shifting the focus is very helpful in developing the mindset for systems thinking. Blaauw’s principles for systems Architecture are excellent guidelines for systems analysts and designers. Systems thinking approaches by Checkland and Jacobs, and Senge toolbox are also very helpful for inculcating the habit of systems thinking. Kohlberg’s six stages can act as ladders of moral development to take care of the moral aspects of systems thinking. Finally, as discussed in the previous section, use of the risk assessment techniques is also very helpful for developing systems thinking perspective. *We include all these models and tools in our proposed framework of pedagogical engagements (Table 8.6).*

**Section 6.4: Software Developers’ Education for Development of Intrinsic Motivation to Create/Improve Artifacts**

In our 2009 survey on required competencies for software developers, twenty software professionals assigned ‘intrinsic motivation to create/improve things’ an average rating of 2.9 on a scale of 0-4. A majority of 65% of these respondents recommended it to be a critical or very important competency with respect to the requirements of software developers' multi-faceted professional activities.

With reference to Appendices A2 and A3, ‘intrinsic motivation to create/improve artifacts’ of software developers also relates to the following:

1. Design skills
2. Creativity and idea initiation
3. Complex problem solving
4. Entrepreneurship, initiative taking, enjoys challenges, sense of mission, perseverance, result orientation, commitment, self motivation, dedication. Adaptability, flexibility, open-mindedness, and ability to multi-task
5 Research skills
6 Experimentation skills
7 Readiness for lifelong learning

Intrinsic Motivation
According to Webster's Dictionary, motivation is "the psychological feature that arouses an organism to action;" and "the reason for the action." Psychologists have carried out extensive research on various aspects of motivation. Motives influence one’s perception, cognition, emotion, and behavior [292]. Annexure AN10 summarizes the perspective of Aristotle, Descartes, James and McDougall, and Murray on this issue.

In 1943, Maslow proposed his famous theory of hierarchy of human needs [141a]. After later extension, his theory classifies human needs in a hierarchical structure of levels given in Table 6.9. He viewed that a person attempts to satisfy basic needs before directing behavior toward satisfying upper-level needs. According to him, people have a need to grow to move up the hierarchy of needs. The satisfied needs cease to motivate and unsatisfied needs can cause frustration, conflict, and stress. We view that higher education must motivate students to raise the levels of their needs on this hierarchy. This hierarchy is part of our proposed framework of pedagogical engagements in software development education (Ref: Section 6.5 and Table 8.2, third column).

<table>
<thead>
<tr>
<th>Table 6.9: Maslow’s Hierarchy of Human Needs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Biological and physiological needs,</td>
</tr>
<tr>
<td>2. Safety needs,</td>
</tr>
<tr>
<td>3. Belongingness and love needs,</td>
</tr>
<tr>
<td>4. Esteem needs,</td>
</tr>
<tr>
<td>5. Cognitive needs,</td>
</tr>
<tr>
<td>6. Aesthetic needs,</td>
</tr>
<tr>
<td>7. Self actualization needs,</td>
</tr>
<tr>
<td>8. Transcendence needs</td>
</tr>
</tbody>
</table>

Annexure AN10 includes some later perspectives by several researchers like Herzberg, Vroom, Alderfer (ERG theory), and Reis [294].

Ryff and Singer [245] have identified six factors for psychological wellbeing: self acceptance, positive relations with others, autonomy, environmental mastery, purpose in life, and personal...
growth. Hence, satisfaction of higher level needs as per Maslow’s model, motivator factors as per Herzberg’s theory, or growth needs as per ERG theory is necessary for wholesome experience and happiness in life. Enrichment and advancement of needs from low-level to higher level is not automatic. Satisfaction of lower level needs does not automatically facilitate upward movement of motivation factors. In 1980’s, Deci and Ryan proposed ‘self determination theory’ to suggest that humans have three innate psychological needs: autonomy, competence, and relatedness [296-298].

**Motivation for Creativity**

As per Sternberg, *motivation behind creativity is to go beyond what is known.* Sternberg [299] saw motivation at the centre of the processes that result in the development of expertise such that it affects meta-cognitive as well as knowledge acquisition activities. It also evolves as a result of learning and thinking. Ambile [300] proposed that the *intrinsically motivated state is conducive to creativity, whereas the extrinsically motivated state is detrimental.* Creativity research has found that *personal autonomy is a core characteristic of creative people.* Autonomous people consider their behavior as emerging from themselves, and may stay more deeply and creatively engaged in what they are doing. Self determined people may be more open to possible analogies or intuitions that are relevant to the problem with which they are concerned. They may also devote more conscious attention to problems that genuinely interest them [301].

Cognitive orientation theory [302] sees *motivation for creativity as a function of beliefs of four types (about goals, norms, oneself, and general beliefs about others and reality)* concerning themes identified as relevant for creativity. Their findings have shown that *there are attitudes and personality tendencies that promote creativity.* As per this study, the high and low creativity architecture students showed significant differences in the following themes:

i. feeling it is incumbent upon them to activate and use their talents and unique abilities
ii. interest and no discomfort in regard to views which differ or contradict their own
iii. daydreaming a lot
iv. demanding a lot from themselves
v. not in need of firm framework or strict regulations
vi. tendency to do original things
vii. tendency to delve deeply into what one deals with and examine it from all points of view
viii. thinking about things in one’s own way, and not necessarily as one has been taught
ix. thinking and doing one’s own thing even with no support from others
x. concern with the functionality of what one does
xi. ability and tendency to invest a lot of effort

Advancing this work, Caskin and Kreitle [303] have concluded that the belief system of highly creative students of architecture and engineering disciplines put a lot of emphasis on self - its uniqueness, development, and expression. The second major factor is maintaining openness to the environment without endangering inner directedness. They found that self beliefs as well as goal beliefs supporting creativity are higher in students of architecture and there are no significant differences with respect to their general and norm related beliefs. Architectural students scored higher than engineering students in the following groupings:

i. Self-development: investing in one-self and developing oneself; taking advantage of opportunities for promotion and learning; developing skills; not satisfied with any achievement but seeking more.

ii. Emphasis on the inner world: more interested in what takes place within oneself than in what occurs outside and in others; making efforts to learn about him/herself; feeling contradictions within the inner world; emphasizing the importance of fantasy

iii. Inner-directedness: making efforts to succeed in circumstances in which others tend to fail; lack of support from others does not affect self-confidence or self-esteem; clarity about one’s goals.

iv. Emphasizing one’s uniqueness: experiencing one’s uniqueness; feeling that he/she has unique talents; developing and highlighting one’s uniqueness; understanding things in his/her own way; being different from others; seeing things differently from others; making original things.

v. Functioning under conditions of uncertainty: liking ambiguity and uncertainty; liking to take risks; liking jobs in which not everything is clear; functioning even if he/she cannot control every detail of the process.
vi. Self-expression: expressing emotions outwardly; creating something personal; expressing thoughts, views, and skills; externalizing feelings; being loyal to own feelings and ideas; speaking with others about oneself.

vii. Non-functionality: able to work even if sees no immediate benefit; does not believe that every idea can be implemented in practice; compromising in regard to practicality and functionality; readiness to act even if functionality is not clearly stated, or not clearly requested from the start.

On the other hand, engineering students had higher scores in the following groupings:

i. Freedom in functioning: does not need a rigid framework of rules defining the situation and the conditions; unable to function according to the instructions of others; functions by intuition; need of freedom in thinking and acting; acting because he/she wishes and not because he/she ought to.

ii. Being receptive to the environment: extracts something from the environment even if it offers only a few stimuli (openness to the environment); absorbing from the environment as much as possible, not selectively; curious to learn a lot about every domain.

iii. Demanding from oneself: does not withdraw in the face of difficulties; striving for perfection, getting to the level of excellence one determines for oneself; high demands from oneself; Investing without limits; able to renounce comfort and pleasure.

Both the groups showed similar results with respect to their beliefs about contribution to the society: make something important and significant, even if it does not contribute to self-promotion; feeling that one can promote the general welfare; devoting time and effort to society; readiness to invest a lot to help people.

**Pedagogical Perspective**

In our recently concluded survey, “Software developers - (How) Did your college help you in your development?” (Table A10.2 (iii), Appendix A10), a large fraction of 74% felt that as compared to all other kind of academic engagements, their student projects did much better to develop their ‘urge to create/improve things’ and open mindedness. This was followed by research literature survey (58%), thinking oriented lectures (54%), discussions with students and faculty (50% each), mentoring juniors (44%), and laboratory work (42%). Written examinations,
traditional knowledge delivery oriented lectures, and homework were found to be least effective in this regard by the respondents. One of the main purposes of education is to sensitize and help its beneficiaries to enrich and nourish their intrinsic motivation towards growth oriented needs of cognition, aesthetics, self actualization, and transcendence needs.

Further, software development work requires significant design effort. In order to create interesting software, the developers need to first become intrinsically motivated interesting persons. Due to their nature, design problems cannot be solved by retrieving already existing solutions or by applying a routine process. Consequently, it is very important that software development education programs create such conditions that ignite intrinsic motivation among its students for creating/improving things.

Love for challenges, habit of perseverance, concentration, and initiative taking depend upon intrinsic motivation. Computing students also deserve to be self motivated to enjoy the pleasure of creative tasks for its own sake rather than for the associated extrinsic rewards. However, the above study clearly exposes a strong weakness of traditional engineering education in this aspect. It does not help the students much to evolve their attitudes and belief in support of creativity. Hence, if computing students’ intrinsic motivation for creativity needs to be enhanced for creating conditions for self actualization through creation, their education process needs to be significantly enriched, perhaps even by borrowing elements from architecture or design education that relatively more strongly encourages their students for seeking self-development, uniqueness and self-expression.

Intrinsic motivation for creativity is very difficult to develop through educational interventions. Repeated engagements in self reflection, collaboration, and a creativity supporting educational environment, as discussed in Section 5.3 is likely to help. Over-emphasis on external rewards like grades is detrimental to inculcating the intrinsic motivation. We have not been able to suggest any concrete pedagogical models in this regard. There is a need for more research to find suitable solutions for this goal.
Section 6.5: Chapter Conclusion

In this chapter, we have discussed the rationale of three traits that are classified as the most critical attitudes, perceptions, and values for software developers: motivation to create/improve things, curiosity, and systems perspective. We have examined some philosophies, models, theories, suggested procedures, and empirical results from multiple disciplines to understand the deeper meaning of these traits. In addition to the opening up the possibilities of upward movement along the professional development ladder proposed in Table 4.7, we also propose that development of the required attitudes, perceptions, and values for software developers must be kept on the top of the agenda of software development education programs. In order to achieve this goal, software development education programs must first aim to facilitate students’ movement to the higher levels of each of the following dimensions:

1. **Cognitive development:** Perry’s model, and others, discussed in this chapter suggest the levels of development along this dimension (Table 6.1).

2. **Personal need perception development:** Maslow’s model, and others, discussed in this chapter can be used as reference for understanding the levels of development along this dimension (Table 6.9).

3. **Levels of Systems Thinking:** Derived from Boulding and Sanford (Table 6.4).

4. **Moral development:** Kohlberg’s model opens up the possibilities of understanding the levels of development along this dimension (Table 6.8).

Table 8.2 juxtaposes these models. Further, with reference to decision making in software development, we posit that the software developers need to integrate the four decision making styles identified by Rowe and Boulgarides (Ref: Table 6.2 and Table 8.5).