CHAPTER 5: SOFTWARE DEVELOPERS’ EDUCATION FOR
DEVELOPMENT OF
COMPETENCY DRIVER-HABITS OF MIND

As discussed in Section 4.5, Costa and Kallick [203] suggested sixteen habits of mind of intelligent people to solve unfamiliar problems (Annexure AN6). Good professionals develop powerful habits of mind to use their intelligent thinking behavior for solving problems within their professional settings. These thinking habits distinguish them from the novices. *Thinking is the creation of a mental representation of what is not in the immediate environment* [209].

Thinking continuum spans from one extreme of automatic thinking to another extreme of controlled thinking. Pure association is the simplest form of automatic thinking. Automatic thinking occurs in situations where repetition encourages decision making based on previously learned responses. In controlled thought, in contrast, we deliberately hypothesize a class of objects and experiences, and then view our experiences in the light of these hypothetical possibilities. Formal thinking, visual imagination, scenario building, creation are forms of controlled thinking. According to Piaget, in formal thinking, the reality is viewed as secondary to possibilities.

Software development is more of a cognitive activity rather than a construction activity. With specific reference to debugging, which is one of the key challenging activities of software development, Metzger puts forward a systematic approach by integrating the thinking perspectives of six approaches: detective, mathematician, safety expert, psychologist, computer scientist, and engineer [157]. With respect to the multifaceted activities of software development, following the following three mental habits have been identified as the most important for software developers:

i. Attention to details
ii. Critical and reflective thinking
iii. Creativity and innovation
Those with a history of successful thinking efforts of some kind are much more willing to make more thinking effort of that kind. They know from past that they can productively engage in such thinking. Hence, it is imperative for software development education to nurture these habits among the students.

**Section 5.1: Software Developers’ Education for Development of Attention to Details**

Thoroughness and concern for different perspectives and aspects, including very small or routine matters, are very important for all software developers. They need to carefully examine the objects/ideas under consideration in terms of form, function, relationship, and perspective. Programming requires habit of long attention spans typically lasting several hours and often for several days on a single problem. Software designers need to work at varying levels of abstraction, and ensure consistency in terms of interpretation and implementation across these levels of abstraction. This requires a keen attention to details, and the ability to correlate them across the various levels of abstractions encountered.

**Importance of attention to details for software development work**

Further, given the inconspicuous nature of software, its visibility limited to the side-effects affected in the system's environment, it is imperative that the limited visibility is accurate and consistent with the desired objectives and behavior. This requires careful planning and execution, with particular attention to exacting details. Expert programmers have the habit of paying attention to minor details [194].

**In our 2009 survey** on required competencies for software developers, twenty software professionals assigned ‘attention to detail’ an average rating of 3.3 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, ‘attention to detail’ of software developers also relates to the following:

1. Good grasping power and attention to detail: breadth, depth, clarity, accuracy, preciseness, specificity, relevance, significance, completeness, consistency
2. Listening skill
3. Quality consciousness

Boehm quotes Winston Royce from his classical paper on the waterfall model written in 1970 [210], “In order to procure a $5 million hardware device, I would expect a 30-page specification would provide adequate detail to control the procurement. In order to procure $5 million worth of software, a 1500 page specification is about right in order to achieve comparable control.”

Over the last four decades, the behavior richness of software has exceeded the data or control richness. Agile software development methods recognize the limitations of the human mind in its capacity to see and freeze the details in advance. They view detailed acceptance tests not just as testing artifacts but also as executable requirements. They differ from the traditional waterfall model essentially by continuously evolving and detailing the specifications iteratively and incrementally creating just enough documentation for the situation at hand in a just-in-time manner. Empirical methods have become very popular in software engineering, hence, **ability to gather data and its systematic analysis** have become very important for software developers. After spending enough energy exactly detailing the specifications and/or acceptance tests, the algorithm/computation design process becomes a much simpler task.

Further, software developers need to follow and comply with policies, procedures, checklists, safety and security measures, and standards. In their thinking and expressions, software developers need to show attentive considerations for context, scope, boundaries, interfaces, assumptions, scalability, and constraints. Often serious oversights occur during the systems analysis phase, resulting in wrong, inconsistent, or incomplete requirements, poor usability, and poor test planning. **Software bugs, often requiring costly rework, are also usually caused due to oversight over seemingly minor details.**

Metzger has catalogued various types of **skill based errors** in software that occur either because of ‘inattention’ or ‘over-attention’ by the developers [157]. As per Metzger, the **inattention**
Failures category includes psychological error subcategories of ‘interrupt sequence, start another sequence,’ ‘interrupt sequence, omit step,’ ‘interrupt sequence, repeat step,’ ‘sequence interrupted, loss of control,’ ‘multiple matches, incorrect choice,’ ‘multiple sequence active, step mixed,’ and ‘sensory input interferes with active sequence.’

The over-attention failures arise because of human memory failure and manifest themselves as omission, repetition, and reversal. The errors in this category include psychological error subcategories of ‘forgetting the goal,’ ‘order memory error,’ ‘spacing memory error,’ ‘coordination memory error,’ ‘remembering incorrectly,’ and ‘not remembering.’

Further, Metzger recommends that like a detective, debugging requires the developers to focus on facts, pay attention to unusual details, gather facts before hypothesizing, use a system for organizing facts, state facts to someone else, start by observing, avoid guessing and following emotionally comfortable hypotheses, keep a log of observations, assumptions, hypothesis, and experiments, and follow look-once-and-look-well strategy.

Good software needs to have mechanisms for ensuring data consistency, fault tolerance, and graceful handling of exceptions.

Code analysis, performance tuning, quality assurance, standard and regulatory compliance, program comprehension, code archaeology, large teamwork, geographical distribution, legacy systems, contractual constraints, risk engineering, data or technology migration, and disaster recovery require very careful attention to minute details. Procedures of version control, configuration management, requirement tracing, defect tracing, and document tracing also require an attentive eye for details. Hence, software developers need to have the habit of repeated verification and careful monitoring. For ensuring traceability, they need to regularly organize and maintain records of their work. They need to develop the habit of seeking and bringing clarity, precision, accuracy, completeness, and consistency in work, its documentation as well as record.
Some Theoretical Perspectives on Attention

Cognitive psychologists have been studying the phenomenon of attention for several decades. Galotti gives an excellent account of their findings on this aspect [196]. We give a brief summary in Annexure AN6. In the 1980’s, Anne Treisman showed that perceiving individual features takes little effort or attention, whereas gluing features together into a coherent object requires more. Software development is basically about gluing a large number of abstractions related to application domain as well as programming environment into a coherent system. Hence, it requires a significantly higher level of attention.

As per classical Indian philosophy, the Raja Yoga system deals with attention and concentration. Yogi Ramacharaka (real name William Walker Atkinson) wrote a commentary on this system [211]. In this commentary, he wrote that the word ‘attention’ is derived from the Latin words "ad tendere," meaning "to stretch toward." It involves focusing of mind’s entire energy upon the object, observing every detail, dissecting, analyzing, and drawing every possible bit of information about the object. Attention is a prerequisite of good memory, and it also affords the powers of association. It enables one to combine, associate, classify, etc., and thus create new knowledge and expressions. It sharpens all other mental faculties.

According to Raja Yoga [211], after voluntary attention is firmly fixed, and held upon an object, the mind will "do the rest." Voluntary attention is a very good substitute for genius, and unlike genius, it can be sharpened through practice and perseverance. Attention requires thinking of, and doing, one thing at a time. This habit is learnt through practice. In order to excel, one has to “immerse oneself" in the work. In order to discover more details about an object, one needs to engage in several iterations of (re)examinations and evolutionary expressions. Critique of the work products of earlier iterations in the light of the re-examination of the object (problems), progressively reveals newer details and affords new opportunities for richer descriptions and other expressions.

Given the knowledge intensive nature of software development, and software development being viewed as a continuous learning task [120] [148], it is no surprise that the evolutionary methods of software development are manifestations of this approach.
**Pedagogic Perspective**

In our recently concluded survey, “Software developers - (How) Did your college help you in your development?” (Table A10.2 (ii), Appendix A10), a large fraction of 71% felt that as compared to all other kind of academic engagements, their student projects did much better to develop their attention to details. This was followed by research literature survey and mentoring juniors (37% each), laboratory work (35%), laboratory work, and industrial training (33%). Discussions with others and traditional knowledge transmission oriented lectures were found to be least effective in this regard by the respondents.

In our proposed framework of pedagogies of engagements in software development education, active (critique) as well as reflective engagements in Sections 8.3.1 and 8.3.3 respectively contribute to the development of this competence.

**Section 5.2: Software Developers’ Education for Development of Critical and Reflective Thinking**

**What is Critical Thinking?**

John Dewey (1909), considered by many as the father of modern critical thinking, posited that critical thinking involves active, persistent, and careful consideration of a belief or supposed form of knowledge in the light of the grounds which support it and the further conclusion to which it tends.

Tama [212] saw it as “a way of reasoning that demands adequate support for one's beliefs and an unwillingness to be persuaded unless support is forthcoming.”

American Philosophical Association posited that critical thinking is purposeful, self-regulatory judgment that results in interpretation, analysis, evaluation, and inference, as well as explanation of the evidential, conceptual, methodological, criteriological, or contextual considerations upon which that judgment is based [213].

**What is Reflective Thinking?**

Kotkamp [219] defined reflection as “a cycle of paying deliberate attention to one’s own action in relation to intention... for purpose of expanding one’s opinion and making decisions about improved ways of acting in the future, or in the midst of the action itself.”
Importance of Critical and Reflective Thinking for Software Development

In a study [191], almost unanimously, i.e., 99.3% of 1023 experts rated ‘abstract thinking and reasoning’ as an important element of human intelligence. In our 2009 survey on required competencies for software developers, twenty software professionals assigned critical and reflective thinking an average rating of 2.6 on a scale of 0-4. A majority of 55% 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, ‘critical and reflective thinking’ of software developers relates to the following competencies:

1. Reasoning: quantitative and verbal, and critical thinking: ability to question, validate, and correct the purpose, problem, assumptions, perspectives, methods, evidence, inference, reliability, relevance, criteria, and consequences
2. Analytical skills
3. Listening skills
4. Quality consciousness and pursuit of excellence
5. Constructive criticism
6. Research skills: methods of mathematical research, engineering research, design research, and social science research
7. Reflection and transition between ladders of reflection. Meta-cognition
8. 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
9. Sensitivity towards global, societal, environmental, moral, and ethical issues, and sustainability
10. Entrepreneurship, sense of mission, perseverance, commitment, self motivation, dedication. Adaptability, flexibility, open-mindedness, and ability to multi-task
Like all professionals, software developers also examine and interpret situations as per some established paradigms of their profession. Software development is well recognized as a knowledge-based activity. Hence, the behavior and performance is controlled by conscious logical and analytical reasoning [157]. Such reasoning is invoked by beginners who start performing a task, or by experienced persons who face a novel situation. In either case, working at this level, we make two kinds of errors either because of resource limitation of the conscious mind, or because of incomplete/incorrect knowledge. These errors are dominated by extrinsic factors, and are difficult to detect and correct.

With specific reference to software debugging, Metzger cites research based findings regarding errors in logical and analytical reasoning. Table 5.1 gives a summary of this collation. This table is included to support our proposed framework of pedagogical engagements (Table 8.5).

**Table 5.1:** Some common errors in logical and analytical reasoning

<table>
<thead>
<tr>
<th>No.</th>
<th>Error Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><strong>Misdirected focus</strong> – tendency to focus on interesting rather than logically important aspects of the problem.</td>
</tr>
<tr>
<td>2</td>
<td><strong>Storage limitation</strong> – storage capacity of conscious mind is extremely limited; hence, the presentation of problem can have a great impact on the ability to store all the relevant information as conscious mind reasons through a problem.</td>
</tr>
<tr>
<td>3</td>
<td><strong>Information availability</strong> – people give too much weight to facts that readily come to their mind, and have a tendency to ignore information that is not readily accessible.</td>
</tr>
<tr>
<td>4</td>
<td><strong>Hypothesis persistence</strong> – preliminary hypothesis formed on the basis of incomplete data early, in the problem solving process are retained in the face of additional, more complete data available later.</td>
</tr>
<tr>
<td>5</td>
<td><strong>Selective support</strong> – people are often overconfident of their information. They justify their plans by focusing on their information and often ignore information that does not support their plan.</td>
</tr>
<tr>
<td>6</td>
<td><strong>Limited reviewing</strong> – people do not consider all the aspects during review. Even when they do, they fail to see the aspects as an integrated whole.</td>
</tr>
<tr>
<td>7</td>
<td><strong>Inadequate data</strong> – people are very likely to draw conclusions from inadequate data.</td>
</tr>
<tr>
<td>8</td>
<td><strong>Multiple variables</strong> – people tend to predict extreme values for partially related variables.</td>
</tr>
<tr>
<td>9</td>
<td><strong>Misplaced causality</strong> – people are likely to judge causality based on their perception of the similarity between a potential cause and its effect.</td>
</tr>
<tr>
<td>10</td>
<td><strong>Dealing with complexity</strong> – people have trouble thinking about complex processes that occur over time, and prefer to deal with a single moment. They also have difficulty in dealing with nonlinearity and multiple side effects.</td>
</tr>
<tr>
<td>11</td>
<td><strong>Decision and probability</strong> – people don’t make good decisions in circumstances that require assessing probabilities.</td>
</tr>
</tbody>
</table>

Only critical thinking can help in controlling such errors in logical and analytical reasoning at various stages of software development. Further, Metzger recommends that like a detective, debugging requires the developers to reason based on facts, validate assumptions, eliminate
alternative hypotheses, reason inductively as well as deductively, and consider all interpretations of the fact that seem to be relevant.

Hazzan and Tomakyo [124] highlight the importance of ability of reflection for software developers. Evolutionary software development approaches including agile methods draw their strength from the possibility of continuous reflection. Reflection helps in building new perspective. The highest level of SEI's Capability Maturity Model (CMM) level 5 (Optimized level) is characterized by focus is on continually improving process performance through both incremental and innovative technological changes/improvements. Such improvements can only be facilitated by reflective thinking. Similarly, the highest (5th) level of People CMM (P-CMM optimizing level) also focuses on continuous improvement of workforce competence through reflection on the quantitative management activities established at maturity levels 4.

Some Theoretical Perspectives on Critical Thinking

Minger’s Framework for Critical Thinking

In 2000, Minger [216] proposed a framework for critical thinking with special reference to management education. Because there are many subjective aspects related to software development, we find it relevant for the purpose of software developers as well. The four levels of this framework are as follows:

i. Critique of rhetoric: argument analysis by checking for logical fallacy, soundness, and validity.

ii. Critique of tradition: critical attitude towards actions in organizations, cultures, traditions, and assumptions that underpin these beliefs.

iii. Critique of authority: being skeptical of one dominant view.

iv. Critique of objectivity: being skeptical of information and knowledge, recognition that information and knowledge is never value free, and are continuously reshaped by the structures of power within a situation. Implies the meta-cognitive process in critical thinking.

These levels are included in proposed framework of pedagogical engagements in software development education (Table 8.5).
Paul’s Model of Critical Thinking

Paul sees it as a mode of thinking - about any subject, content, or problem - in which the thinker improves the quality of his/her thinking by skillfully analyzing, assessing, and reconstructing it. Paul proposed a taxonomy of Socratic questioning to facilitate critical thinking [214]. It included six categories of questions: (i) questions of clarification, (ii) questions that probe assumptions, (iii) questions that probe reasons and evidence, (iv) questions about viewpoints or perspectives, (v) questions that probe implications and consequences, and (vi) questions about the question. This model has been extended and also applied to engineering reasoning [215].

As per this model, the elements for critical thinking are: purpose, question at issue/problem to be solved, concepts, information, assumptions, inferences, interpretations, points of view, implications, and consequences. We also add the elements of context, criteria, and method to this list. This model also lists some standards for critical examination of the elements. These include clarity, specificity, relevance, logical, significance, consistence, breadth, depth, accuracy, precision, fairness, and completeness. Critical thinking involves the processes of identifying, analyzing, synthesizing, evaluating, reviewing, and considering the elements in the light of the abovementioned standards. As per Paul, repeated engagements in these processes result in the development of the intellectual traits required for critical thinking.

Paul’s extended model is included in our proposed framework of pedagogical engagements in software development education as a checklist for guiding critical thinking during various stages of software development (Section 8.3.1).

Some Theoretical Perspectives on Reflective Thinking

Critical thinking about ideas, object and world is not sufficient for creating meaningful systems. In his classic book, Barnett [217] describes his notion of ‘critical being’ as including thinking, self-reflection and action: “critical persons are more than just critical thinkers. They are able critically to engage with the world and with themselves as well as with knowledge.” He identified three domains of criticality: knowledge and ideas (critical reason), the experience of self (critical reflection) and the action in the world (critical action).
Moon [216] cites Ford (2005) who differentiated between the levels of pre-criticality, criticality with the agenda of others without much challenge to the given frameworks, and criticality to one’s own agenda. This calls for reflective thinking. As per the multiple intelligence theory of Gardner, reflection is associated with the intra-personal intelligence. Costa and Kallick [203] also emphasized on the ability to reflect to evaluate the productiveness of our own thinking.

In 1979, Bateson proposed a model of logical categories of learning. He viewed that progressively deeper levels of learning require change of action, assumptions, or context and commitment. The first level of learning is about making minor fixes or adjustments in action. The second level of learning requires reflection to challenge one’s beliefs and assumptions. This facilitates new insights for changing the rules and making major changes. The third level of learning requires even deeper reflection to bring about a shift in understanding our context, values, point of view, and commitments.

Schön [218] defined reflective practice as the practice by which professionals become aware of their implicit knowledge base and learn from their experience. He introduced the following three notions:

1. **Reflection in action**: reflect on behavior as it happens, so as to optimize the immediately following action.
2. **Reflection on action**: reflecting after the event, to review, analyze, and evaluate the situation, so as to gain insight for improved practice in future.
3. **Ladders of reflections**: action, and reflection on action make a ladder. Every action is followed by reflection and every reflection is followed by action in a recursive manner. In this ladder, the products of reflections also become the objects for further reflections. This is included in our proposed framework of pedagogical engagements in software development education (Table 8.9).

Further, Schön [220] posited that the mental habit of reflection and ability to move across the ladders of reflections is central to professionals’ approach to their work. He saw ‘design’ as ‘reflection in action’ in which changing a given situation takes precedence over the interest of understanding it. He also observed that for a designer, the phenomena/situation continues to
change during their work. Table 5.2 summarizes some of the key observation of Schön, in this regard.

Table 5.2: Some key aspects of Schön’s perspectives on ‘design’ as ‘reflective action’

<table>
<thead>
<tr>
<th>Designers begin with situations that are at least partially uncertain, ill-defined, complex, and incoherent. Designers construct and impose a coherence of their own. Subsequently they discover consequences and implications of their constructions – some unintended – which they appreciate and evaluate, sometimes leading to reconstruction of initial coherence – a reflective conversation with material of a situation. They spin out a web of moves, consequences, implications, appreciations, and further moves. Each move is a local experiment that contributes to the global experiment of reframing the problem. Moves create new problems to be described and solved. Moves have expected/or unexpected consequences in many design domains and implication bindings on later moves. In this process, designer reflect in three dimensions:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The domains of languages in which the designers describe and appreciate the consequences of their moves, e.g., use, technology, form, cost, scale, character, representations, quality, standards…</td>
</tr>
<tr>
<td>2. The implications they discover and follow. Designers evaluate their moves in terms of:</td>
</tr>
<tr>
<td>a. Desirability of their consequences.</td>
</tr>
<tr>
<td>b. Conformity to/violation of implications of earlier moves.</td>
</tr>
<tr>
<td>c. Their appreciation of new problems or potentials they have created.</td>
</tr>
<tr>
<td>3. Their changing stance towards the situation with which they converse: Can/might, should/must, what if, unit/total, moves/appreciation of outcomes, and tentative adaption of strategy/commitment.</td>
</tr>
</tbody>
</table>

Relating software engineering to Schön’s work on reflective thinking and professions (1987), Hazzan and Tomakyo [124] posit that mental habit of reflection and ability to move across the ladders of reflections are closely associated with software engineering processes. They also give examples of such ladders of reflection for soft engineering tasks. Further, one of the key principle in the agile manifesto is, “at regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.”

Proposing a model for reflective design, Phoebe Sengers et al [221] recommend that designers should use reflection to uncover and alter the limitations of design practice, to re-understand their own role in the technology design process, and to support users in reflecting on their work and lives. Stones [222], Ginsburg [223], and Lasley [224] have identified the following elements of reflection: (i) practical experience, (ii) meaningful knowledge base of subject, context, and users, (iii) interaction with others, (iv) philosophical awareness and understanding of what constitutes good practice, and (v) strong problem solving skills. These elements are embedded in various dimensions of our proposed framework of pedagogical engagements in software development education (Section 8.3).
Reflection is not an automatic activity. It requires controlled thinking. 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.


Pedagogic Perspective

Many studies have showed that multi-paradigm disciplines like humanities, social sciences, and psychology had a positive influence on students’ self reported growth in critical thinking skills. However, Li et al [226] have found that self perceived gains of students’ critical thinking skills most significantly depended upon the degree of their integration into the academic and social community of the university rather than their discipline of study. The other significant influencing factors were found to be the quality of lower division courses. Gender and quality of advising were found to be insignificant factors in this regard. The quality of teaching was found to be a very significant factor for influencing their academic integration. The quality of curriculum was found to be the most significant factor for influencing their social integration.

In our recently concluded survey, “Software developers - (How) Did your college help you in your development?” (Table A10.2 (ii), Appendix A10), half of them felt that as compared to all other kind of academic engagements, their student projects did much better to develop their critical and reflective thinking. Critical and reflective thinking was also felt to improve through engagements of thinking oriented lectures (48%), discussions with faculty and other students (44% each), and research literature survey (42%).

Engagements in homework, knowledge transmission oriented lectures, written examinations, and industrial training were felt to be least effective with respect to development of critical and
reflective thinking. Adding an element of reflection after all their engagements can enhance the perceived effectiveness of many low rated engagements as well. In our proposed framework of pedagogical engagement, reflective engagements seek to achieve this goal. We have also discussed some such instructional interventions discussed in Section 9.1.3.

Section 5.3: Software Developers’ Education for Development of Creativity and Innovation

What is creativity?
In a study [191], a large fraction of 59.6% among 1023 experts rated ‘creativity’ as an important element of human intelligence. It was rated much above some other elements like ‘goal directedness’ or ‘achievement orientation.’ Costa and Kallick [203] have included ‘Creating, Imagining, and Innovating,’ as one of sixteen mental habits that characterizes intelligent people when they are confronted with problems, the resolution to which are not immediately apparent. As per Sternberg’s theory of tri-archic intelligence, creative ability along with practical and analytical abilities together define human intelligence. He explained creativity as the ability to apply problem solving processes to novel and unfamiliar problems.

Divergent thinking, i.e., the ability to generate new ideas is at the core of creative ability.

Creativity is generating new thoughts, invention is transforming the creative thoughts into novel tangible ideas, and innovation is the first novel application of those ideas in a specific context. Osche [227] sees creativity as bringing something into being that is original (new, novel, unusual, unexpected) and valuable. She posited that the most important criterion was the willingness of creative people to work hard and put in the extra time necessary to turn out a quality product in a given domain. Albert Rothenberg associated creativity with, ‘Janusian thinking,’ i.e., the ability to conceive and hold two or more contradictory or opposite thoughts simultaneously. He also posited that the creative process is a matter of continually separating and bringing together, bringing together and separating, in many dimensions: affective, conceptual, perceptual, volitional, and physical.
While, mystical view of creativity attributes it to divine inspirations, pragmatic view believes that some techniques can stimulate creativity. Psycho-dynamic perspective posits that it arises from the tension between *conscious reality* and *unconscious drives*. Social-personality attributes creativity to personality variables, motivational variables, and the socio-cultural environment. Evolutionary approaches suggest that like the process of evolution, blind generation of a large number of ideas should be followed by selective retention. Confluence approach seeks to integrate various perspectives.

**Importance of Creativity for Software Development**

In our survey of fifty-seven software professionals (Table 4.2), 66% respondents included ‘innovation and research’ 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. In another survey conducted by us in 2009 on required competencies for software developers, twenty software professionals assigned creativity and innovation an average rating of 3.0 on a scale of 0-4. A large majority of 80% of these respondents recommended ‘creativity and innovation’ 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, creativity and innovation of software developers also relates to the following:

1. Design skills
2. Imagination: storyboarding, extrapolation, visualization, cognitive flexibly: ability to transfer and models of solutions of one situation/field to another, multi-perspective thinking, lateral thinking, inductive thinking, out-of-box thinking, unstructured thinking
3. Complex problem solving
4. Research skills: use and integrate methods of mathematical research, engineering research, design research, and social science research
5. Experimentation skills
6. Entrepreneurship
Usually, software design projects require more than a synthesis of previously learned knowledge. Design is primarily an inductive process. This process of reasoning is non-deductive: there is no closed pattern of reasoning to connect the needs, requirements, and intentions with a form of software. To succeed as software designers, computing students need to be well trained in inductive reasoning.

Software has grown much beyond the simple interfaces to information. Much of software is increasingly becoming concerned about user’s experience.

In the last two decades, there has been an increasing attention on user interface design with a focus on user experience. Software companies need creative people in order to do the high-level design of new innovative software products. They need creative minds to design the new procedures and tools that make the development of new, ever-more-complicated software applications easier. A reductionist and linear thinking give evolutionary incremental advancements; revolutionary advancements come from non-linear and holistic thinking, and intuition. Software companies require both kinds of mindset. Non-linear thinking is necessary for generating the ideas to break current boundaries. However, linear mindset is necessary for executing these ideas.

With reference to the complex problem solving discussed in section 4.5, the aspects of problem solving or decision making process in which creativity can be applied are the following: (i) restructuring the problem/decision task, (ii) generating alternatives, and (iii) selecting decision criteria and strategy, and evaluating alternatives [230]. These three are included in our proposed framework of pedagogic engagement (Table 8.5).

Restructuring the problem/decision task requires holistic perception of problem/issue and involves several iterations of redefining the problem and/or goals. Hence, systems thinking discussed in Section 6.3 and reflective thinking discussed in Section 5.2 play a very crucial role in finding creative solutions. One of the software engineers, we interacted with, commented, “a problem is not a problem until it is revised again and again.” Another expressed that “problem
definition is a thing which people generally don’t take interest in. People start running for the different solutions without even having a mere idea of the problem.” An entrepreneur reflected, “problem formulation does not end, at least till a solution is achieved, and sometimes it just goes on and on by improving upon the found solution.”

**Sternberg’s propulsion theory of creativity**

As per Sternberg’s propulsion theory of creativity [229], creative contributions are attempts to propel a field from wherever it is to wherever the creator believes the field should go. He proposed following four-level taxonomy of creative contributions.

1. The lowest level consists of *paradigm preserving contributions* that leave the field where it is through replication.
2. The next creative level is of *paradigm forwarding contributions* that move the field forward in the direction it already is going. This movement may be forward incrementation or advance forward incrementation.
3. A higher level of creative contributions is *paradigm rejecting*. Such creations move the field in a new direction from an existing or preexisting point. It involves redirection or reconstruction.
4. The highest levels of creative contributions are also paradigm rejecting. This rejection is not to redirect the field from an existing old point, but to restart the field in a new place, and move in a new direction from there. It requires *re-initiation and/or integration*. Inter-disciplinary approaches stimulate such thinking.

These levels are included in our proposed framework of pedagogical engagements in software development education (Table 8.5).

**Pedagogic Perspective**

Several techniques have been developed for stimulating the mind for generating alternative ideas. These include Osborne’s checklist, SCAMPER (Substitute, Combine, Adapt, Modify or Magnify, Put-to-another-use, Eliminate, Rearrange or Reverse), and Edward de Bono’s concept of lateral thinking and ‘po’ (provocative operation) emphasizing on suppose, possible, hypothesize, and poetry, etc., [231]. *Brainstorming* also helps a great deal creative thinking.
Altshuller [232] studied hundreds of thousands of patents, and proposed a powerful Theory of Inventive Problem Solving (TRIZ/TIPS). This theory identified 40 recurring principles that were repeatedly being applied by the inventors in different fields. Table 5.3 gives a brief list of these 40 principles. Subsequently, this technique has become very popular among a large number of researchers. Since 1996, ‘The TRIZ journal’ is being published every month at triz-journal.com. TRIZ principles have also been found to be metaphorically manifested in software design [233-234]. Researchers have also attempted to extend these principles by adding some more principles that are especially relevant for software, especially because of its material-less nature [235]. Some of these additional principles include metaphor, scope, evolution, privacy, usability, synchrony, etc.

Table 5.3: Principles of Theory of Inventive Problem Solving (TRIZ/TIPS)

| 3. Local quality | 13. The other way round | 23. Feedback |
| 7. Nested doll | 17. Another dimension | 27. Cheap short-lived objects |
| 34. Discarding and recovering | 35. Parameter change | 36. Phase transition |
| 37. Thermal expansion | 38. Strong oxidants | 39. Inert atmosphere |
| 40. Composite materials | 41. Discarding and replacing |

Kowalick identified seventeen secrets of inventing new products [236]. Some of these are (i) the real problem to be solved is rarely the same as the problem initially posed, (ii) technical systems often have many functions, some of which are useful, and others that are useless or even harmful, (iii) pruning a technical system is one of the highest forms of creativity, and (iv) solving technical design conflicts by making tradeoffs is not as useful as stating the objective in the form of a ‘contradiction,’ and meeting the contradictory requirements by design.

Metaphors and Analogies

Altshuller [232] also made the following observations: (i) problems and solutions were repeated across industries and sciences; (ii) patterns of technical evolution were repeated across industries and sciences, and (iii) innovations used scientific effects outside the field where they were developed. Moreover, since software serves multiple industry verticals, there is ample scope for
Cross-pollination of ideas and best practices. Cross-industry and multi-domain exposure fosters creative thinking, if one has an open mind. Using metaphors helps designers and developers to think around user goals and assumptions. Understanding customer expectations in terms of another common device or appliance that everyone uses may help them to design a better product interface that improves user adoption and reduces training time. However, metaphors and analogies need to be used with care as they come from a specific context, and hence, can sometime lead to serious misunderstandings in changed circumstances. The education program must encourage the development of metaphorical thinking. Arts and Literature related courses can make a huge contribution in developing such thinking.

In our recently concluded survey, “Software developers - (How) Did your college help you in your development?” (Table A10.2 (ii), Appendix A10), a large fraction of 82% felt that as compared to all other kind of academic engagements, their student projects did much better to develop their creativity and innovation. This was followed by thinking oriented lectures (53%), research literature survey and discussion with other students (45% each), laboratory work and discussion with faculty (39% each), and mentoring juniors (31% each). Written examinations, knowledge transmission oriented lectures, and homework were found to be least effective in this regard by the respondents.

Amoussou et al [237] have identified and collated the following activities for enhancing creativity and design in computing courses: (i) reflections on sources of inspiration including brainstorming techniques, (ii) reflections on bias that may affect creativity and design, (iii) identify and define the steps of the design process and provide design examples, (iv) identify and define criteria and constraints, (v) practice methods of evaluating options, (vi) reflect on norms of communication, and (vii) discuss ethics within the context of design.

Lassig [238] has proposed to adapt a balanced view about six environmental conditions to inculcate computing students’ creativity. These are: (i) a supportive and nurturing environment that also provides obstacles and challenges, (ii) some constraints are helpful for novel/unfamiliar tasks that are to be performed with limited knowledge/skills, (iii) evaluation generally inhibits creativity, when it must be done, the criteria should be clear, self evaluation
can also facilitate creativity, (iv) if the task is not too difficult, competition can stimulate a person who is initially not very motivated, however, if the task is difficult or the person is already motivated, competition can create anxiety and inhibit creativity, (v) enthusiastic cooperation does not automatically lead to more creative ideas, and (vi) role models are helpful for enhancing creativity, only when they encourage independent thinking.

We have included theories and techniques on creativity and innovative thinking in the course content of two undergraduate elective courses: (i) human aspects for information technology and (ii) software arteology. Many students have reported that it helped them to expand their creative thinking for their software projects. Later in Section 8.3.1., we support our proposed framework of pedagogical engagements in software development education with techniques of SCAMPER, lateral thinking, 40 TRIZ/TIPS principles and further extensions, as well as the activities collated by Aoussou et al and the environmental conditions suggested by Lassig, as discussed above.

Section 5.4: Chapter Conclusion

In this chapter we argued that the multifaceted basic competence for software developers can only be used and refined with the help of habits of mind that drive competence. These habits include attention to detail, critical and reflective thinking, and creativity and innovation. As per our studies discussed in this chapter, student-centric pedagogical activities, especially projects have been found to be most effective for development of these habits. The only way to inculcate these habits is by engaging them in such tasks that require them to use these habits. Only through repeated usage can these be enhanced. Development of these habits has to be put as a core learning outcome of all courses. We further discuss this issue in the seventh, eighth, and ninth chapters. In the next chapter, we discuss competency conditioning attitudes and perspectives that help in enhancements and meaningful application of these habits.