CHAPTER 3: DISTINGUISHING FEATURES OF
SOFTWARE DEVELOPMENT AND
REQUISITE TAXONOMY OF CORE COMPETENCIES

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

Technology professionals are expected to serve human needs through designing, building, evaluating, maintaining, testing, and modifying systems, processes, and components. In the process, they engage in creation, operation, maintenance, application, and destruction of forms of matter or energy, and/or information. Software developers are concerned with information aspects of such endeavors. Software provides support to acquire, store, search, filter, transfer, transform, and/or destroy information. The transformation may involve content transformation, form transformation, composition, and/or decomposition. Software is not just enabling people to do their activities in newer ways.

It is also empowering them to do new activities to satisfy their needs at multiple levels of Maslow’s need hierarchy [141a], and also seek happiness through expanded levels of positive relatedness, and also enhanced levels of autonomy and competence. Software-enabled newer ways of self-expression are helping people to experience higher levels of self acceptence. Hence, software-based artifacts are also reshaping the cultural landscape of society.

It is not proper to call software engineering an extension to any single discipline. Like languages and mathematics, the problem domains and solution possibilities are unlimited, and hence, software engineers can find opportunities to integrate their disciplinary knowledge with any
other discipline, and also grow in that domain. The discipline of computing is inherently highly inter-disciplinary and is continuously expanding through collaborations with other disciplines of human understanding. Surely, concepts are shared back and forth, but software engineering has its own distinguishing features. It, without doubt, shares many common practices like project management, and making design tradeoffs with other engineering disciplines. However, some of the following distinctive aspects distinguish it from many other engineering disciplines.

Section 3.1: Programming as an Art to Software Engineering

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

As computer programs started handling more complex tasks of various applications, their size increased. Large programs need modifications and enhancements throughout the period of their active use. The development and maintenance of large programs was no more an individual effort, but a teamwork which requires disciplined engineering processes with systematic documentation of activities of analysis, design, coding, testing, deployment, and maintenance. This was a qualitative jump in the software development process, and by the mid 1980s various courses covering different aspects of Software Engineering became an integral part of computing education. Software development evolved out of a mathematical art to an engineering activity for handling complex tasks. Similar to the other engineering activities, problem solving and technical competence would be the key core competencies required for software engineering.

Engineering with a Difference

Software engineers do not manage high volume manufacturing, or mandatory repeated implementations. Unlike other artifacts, replication of software artifacts is easy, and often without any costs. A much larger number of skilled developers and testers have to collaborate to create and evolve many software artifacts. Therefore, only skilled workers are required for software engineering related activities, and group work is much more important. Software engineering is comparatively much younger to other engineering disciplines and the theories, best practices, and essential development tools are still evolving. Further, the obsolescence and
technological changes are very rapid. Consequently, *disciplined lifelong learning is a must, for software engineers.*

While all other engineering disciplines aim to create physical products by combining material and physical processes, software is created from ideas and existing software, and is largely independent of the material and physical processes. Hence, it has *much lesser dependence on physical sciences and constraints.* This gives them the freedom to *create imaginative virtual spaces and services limited only by human thinking.* Hence, there is considerable room for variant approaches to defining and solving problems and creative thinking.

Designs often involve a large number of layers of *discrete abstractions and complex interactions* among a very large number of components. The methods of dealing with this complexity are not mature and effective enough. Consequently, projects face higher uncertainty factor, and a design cycle often requires several iterations. According to estimates, *80% of software projects fail to meet their original objectives, schedules, and budgets* due to a failure to manage this complexity [142].

Because of the underlying digital phenomenon, noise, fluctuations, uncertainties, or errors can result in unpredictable outcomes and software crashes. Further, the *inherent invisibility makes it vulnerable to failures and unpredictable behavior.* This often causes the absence of indicators of failure before total catastrophic failure. Hence, the designer’s task to create and manage “soft failure modes” becomes more complex.

A lot of available software is highly vulnerable to failures. *With exponentially expanding size and complexity of software, its reliability is becoming an even more challenging issue.* Fortunately, most software artifacts are not potentially life threatening or a source of human injury. This, however, has unfortunately contributed to insufficient sensitivity towards risks and reliability among developers. In the last two decades, more attention is being given to developing and following good engineering practices to minimize software risks. This issue is further elaborate in Section 6.2.
In this section, we discussed how the software development process evolved from an individual’s art to a large team’s engineering effort. However, some characteristics like knowledge intensive nature of the work, much lesser dependence on physical sciences and constraints, inherent invisibility, discrete abstractions, and complex interactions, etc., make it a significantly different from all other forms of engineering. Hence, we posit that while many elements of traditional model of engineering education are not in alignment with the requirements of software development.

**Section 3.2: Debugging as a Core Activity in Software Development**

A software bug is an anomaly in behavior of running program [142a]. Detection and fixing of bugs is an ongoing process in software development and evolution. Every phase of testing in Software Development Life Cycle (SDLC) i.e. unit testing, integration testing, system testing and customer testing is typically followed by bug fixing. According to Humphrey [371], more than half a typical software organization’s effort is devoted to finding and fixing defects. Indeed, such is the emphasis on debugging that many software companies test their prospective employee with buggy code. Proficiency in debugging is integral to deep understanding of software development. Though the possibility of bug introduction lies in every phase of Software Development Life Cycle, but a majority of them can be traced to either design or implementation issues. A significant amount of research has been and continues to carried out on debugging. In March 2010, a word search on “debugging” in ACM Guide showed approximately 23,000 papers. Out of these, approximately 11,000 papers have been published since 2005. In no other developmental activity including engineering, an exactly analogous activity does not play such a central role.

IEEE Standard Classification for Software Anomalies [142a] provides comprehensive methodology for classifying bugs in each phase of bug life cycle. It presents customizable framework for software organizations which serves two objectives. First, it facilitates effective bug tracking by enforcing bug classification in each phase of bug life cycle i.e. Recognition, Investigation, Action and Disposition. Second, it provides with data of bug classification which can be analyzed to identify problematic areas responsible for common bugs. Additionally, a
reflective analysis of bug related data of project and/or release, can help in measuring impact of a process change followed for a particular release or project.

**Section 3.3: Process Centric System Development and Maintenance in Software Engineering**

The term ‘engineering’ in the context of software development refers to having a *systems approach* to problem solving, and also following a disciplined *process-centric/oriented approach* for assuring the quality of the deliverable software system, and their *maintenance and evolution throughout their life cycle*. The traditional approach of engineering education does not pay much attention to user interaction and evolution, and hence, is not completely suitable for software.

The various aspects of the system development process mainly deals with human processes and engineering management processes. Due to the absence of physical material, the software maintenance activity is not about managing wear and tear. Instead, it is focused on *learning about the misunderstood and changing requirement, removing development errors, and continued development*. Consequently, it is imperative for software developers to develop the *process-centric system development approach as well as the competencies for software maintenance and continuous evolution*.

**Section 3.4: Software as an Integral Part of Business, and Need for Comprehension for Software Maintenance**

The engineering approach to software development, coupled with continuous exponential advancement in computer hardware technology, brought a higher level confidence among its users, and they started to look at this integrated field as ‘information technology.’ Software has now become an integral part of business processes of a large number of organizations. Today, any development of business/application software must presume that the *developed software would go through repeated changes*. To satisfy their clients by the quality of their service, *software professionals must be capable of comprehending the existing client software*, and then performing the required modifications/enhancements as per their evolving requirements.
Often a good amount of very old program and/or open source code is re-used, and is blended with new code in the enhanced and newer versions of software. A very large number of developers are engaged in maintaining and evolving the work of other developers. Reuse-based development methodologies are becoming more popular. Hence, a good familiarity with existing components, open source, and the ability to comprehend programs are very important. This is presently not at all addressed by computing educational curricula. Therefore, *software developers need to develop the ability to comprehend the software developed by others, and also write software that can be easily comprehended by other developers.* Increasing dependence on large amounts of Free and Open Source Software (FOSS) makes it even more crucial.

**Section 3.5: Role of Empathy and Social Sensitivity in Software Development**

Software development in many ways is all about people: users, customers, developers, and managers [143]. All other engineering disciplines attempt to boost human performance by building tools, programs, and systems for supporting physical processes, but on the contrast the software engineers do so by supporting their cognitive processes.

**User Empathy**

Many artifacts and services created by software engineers offer a much higher level of cognitive, social, and emotional engagement opportunities to users. Unlike all other engineers, software developers do not spend much time building their systems, instead they spend more time trying to figure out what the systems should do. The activity of software development is like ‘writing,’ where the real task is actually knowledge acquisition, construction, structuring, and representation.

Software design solutions and approaches are often manifestations of the designer’s thinking process, rather than the expressions of a physical phenomenon. Hence, an understanding of people’s cognitive processes like how they think, how they plan, how they assess situations, how they represent and structure information, how they decide, and other related mental processes is not only helpful but often an essential requirement of their work. This makes their task of
requirement elicitation much more complex. The analysts and designers often have to understand
the difference between what clients and users ask, want, and actually need. Designs that are
suitable for a context are not necessarily as suited for other contexts. Software developers need to
understand users’ requirements from multiple perspectives. Software projects usually require a
significantly higher level of ‘communication’ for customer interaction and support. Therefore the
analysis, design and architectural part of the software calls for an integrative balance of
structured as well as unstructured thinking. The implementation part is easier and mostly
depends upon structured thinking.

The main concern of software developers has been gradually shifting from making ‘inexpensive’
software to ‘quality’ software to ‘appropriate’ software. In order to create such software,
developers have to acquire the mental adaptability to clearly understand the nuances of
processes, and identify the automation possibilities in various application domains. Consequently, they need to have not only an interest in the work of other human beings, but also
an ability to understand their experiences as well as beliefs.

Because of the people-centric nature of the activity, the software developers have to deal with
professional challenges related to intellectual property, security, privacy, anonymity, offensive
content, and cyber regulation, etc. It is crucial for software engineers to respect cultural
diversity, and appreciate the conflicts and complexities of the human mind. User empathy: seeing
things from users’ perspective, and aptitude for ‘narrative reasoning’ are essential for those
software engineers who analyze and design the software requirements specifications.

**Group work**

As compared to the other kinds of engineering industries, the software industry places a much
deeper level emphasis on group work. Large multi-locational, multi-cultural global teams
concurrently work in different parts of the world to meet the requirements of clients of varied
cultural backgrounds. The majority of an engineer’s time in the software industry is spent
working with other programmers. The nature of group work among software engineers is not
limited to process-centered coordination. Whitehead [144] observes that software engineering
projects require many software engineers to collaboratively create a large number of artefacts
incorporating code, requirement specifications, architecture descriptions, design models, test plans, etc. In the software industry, many tools have become popular for facilitating process-centered coordination, ensuring mutual awareness, traceability and consistency, and also collaborative creation of software development related artifacts [145].

Shared development environment, engagement with the team, constant feedback, reviews, and continuous testing and integration are some of the hallmarks of software development methods [146]. All this require a significant amount of group work. eXtreme Programming strongly relies upon practices like daily meetings and pair programming. It uses the practice of pairing not just for code development, but also for design, refactoring, as well as testing [147].

Thus, empathy and social sensitivity and are the core competencies required for development of appropriate and quality software. Development of social sensitivity is not to be taken at the periphery, but at the core of the software developers’ education program. Good exposure to ‘human and social sciences’ as well as arts, particularly literature, can help in this regard. Engagement (comprehension, analysis, and so on) with well constructed literary narratives, requires the person to imagine characters’ position and experiences. Hence, it can be a great help in nurturing these abilities. These aspects are discussed further in Sections 4.4 and 6.3.

Section 3.6: Project Scoping and Estimation for Software Contract

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

Section 3.7: Learning New Domain and Knowledge Structuring in Software Development

The software development processes essentially try to map the application domain requirements to programming constructs. Application domain training and even certifications
have become a common part of continuous training programs of software developers. Software projects require a higher emphasis on abstraction, reflection, modeling, and information organization of various distinct application domains. Therefore, the opportunity of interdisciplinary work is higher, and is further increasing. A deep understanding of various functions in specific application domains is highly valued in the software engineering industry. The software development education program needs to expose the students to diverse types of domains, and also nurture the ability to learn nuances of newer domains.

Software as a Medium to Store Knowledge
Armour [148] argued that software is not a product, but the fifth medium to store knowledge after DNA, brain, hardware (artifacts), and books. For storage, it represents the knowledge in space and expresses the stored knowledge in space and time. He posited that all kind of human knowledge is now being transcribed into software, because software has a wider range of valuable storage and structuring characteristics as compared to all the previous medium: quite persistent like books, update frequency is only slower than that of brain, intentionality (our ability to change it deliberately) is higher than the hardware and books, ability to self-modify is higher than DNA, hardware, and, books, and activeness (ability to affect the outside world) is relatively unlimited.

He took the position that the difficult and time consuming part in software development is not transcribing the already acquired knowledge into an active form, but acquiring and structuring the knowledge with concern of completeness, consistency, and usability. He viewed software development as a learning activity, rather than a production activity, and advocated that software developers need more training in learning, and knowledge structuring mechanisms rather than in software itself. Hence, exposure to disciplines like cognitive psychology that try to understand human understanding and learning becomes much more relevant for software developers.

Section 3.8: Software Development Process for Ill-defined Problems
A very large number of software developers face new problems every day. As application domain requirements are embedded in real experiences, it is usually very difficult to map and concisely describe it as a software problem. Therefore, real-life software problems are mostly ill-
defined problems. Often multiple iterations and representations are required to define the problems. Projects usually require at least an incremental innovation, and significant amount of development. Further, usually there are no unique best solutions, only multiple acceptable solutions to partially solve the real-life problems. Software developers need to explore new opportunities, identify hidden requirements, generate new concepts, incorporate novel elements, innovate new user interfaces, functional and architectural designs, reuse components, uncover hidden faults, derive new use, and do many other such tasks. Accordingly, there is higher focus and challenge on integration, continued evolution, reuse, creativity, and flexibility. In Section 4.5, we shall further elaborate upon ill-defined problem solving.

This characteristic of software problems, and the development process, also contributes to make it a multi-dimensional activity. Agile methods are increasingly being accepted to develop software primarily to address this aspect of software problems. Reflective thinking, multi-perspective thinking, critical thinking, creative thinking, and innovative problem solving significantly contribute in transforming complex ill-defined problems into simpler well-defined problems.

This requires them to be not just skilled, but also reflective and creative. Familiarity with a variety of creative works can help in enhancing individuals’ creativity. Hence, reflective engagements with the discipline of aesthetics, arts, literature, and design can play a significant role in nurturing reflective thinking and creativity of software engineers. Reflective thinking is discussed later in Section 5.2

Problem-centric learning methods have been found to be more effective for developing the competence to solve ill-defined problems [156]. Further, software problems are not only ill defined, they are also socio-technical soft problems (discussed in Section 6.3) that require a soft systems approach. Consequently a systems-level perspective becomes very important for software developers.
Section 3.9: Empirical and Qualitative approaches in Software Development

Research

Software engineering in many ways is all about people: users, customers, developers, and managers [143]. Unlike many other disciplines of engineering, for their regular day-to-day practice, often software engineers have to collect and analyze qualitative data. They collect such data through brainstorming, interviews, conceptual modeling, and observation for activities like requirement engineering, project planning, use case and task analysis, etc. Usage logs, documentation, static and dynamic analysis, bug tracks, etc. are used for activities of program comprehension, testing, reverse engineering, etc., [149].

Further, researchers in software engineering often investigate extremely complex processes in software developments involving a large number of professionals who use highly complex skills. The questions related to cognitive, behavioral, and social aspects of developers and other stakeholders, are also of immense importance and interest to software engineering researchers. The analytical and quantitative research paradigm, which is otherwise well accepted in other disciplines of engineering, is not sufficient for investigating real-life issues involving humans as well as their interactions within themselves and also with technology [150].

Many a times, experimentation is not even possible, and qualitative data is the main, and sometimes, the only source of information. Hence, the researchers in many computing areas, e.g., information systems, software engineering, human computer interaction, and entertainment computing are increasingly relying upon the empirical and qualitative research methods like case studies, action research, survey, etc. These research methods are already very popular in fields like business, social work, psychology, sociology, political sciences, education, information systems, urban planning, architecture, and so on. Like information systems, urban planning, and architecture researchers, the software engineering research community mostly uses these methods with a pragmatic and result-oriented view, rather than from a philosophical stand. Consequently, an understanding of these methods is becoming increasingly important for software engineering researchers, as well as practitioners.
Acceptable data sources and research methods characterize different disciplines. The differences in the two cultures of natural science and humanistic research were highlighted by Snow in 1959 [151]. Qualitative data and methods are not used in science and engineering disciplines. They confine themselves to analytical and quantitative approaches. Qualitative approaches are used in social sciences. However, software engineering practitioners as well as researchers use analytical, quantitative, and also qualitative approaches, and give the opportunity to integrate these two cultures. A heavy dependence on qualitative data and increasing currency of qualitative methods among software engineering practitioners as well as researchers is a distinction that further distinguishes the field from other disciplines of engineering. This brings it relatively closer to disciplines like architecture and information systems. We include the quantitative as well as qualitative data analysis techniques in our proposed framework of pedagogical engagements in software development education (Table 8.6).

**Section 3.10: Software Development: Whole-Brain Activity**

Diverse activities of software development are not confined to any single type of thinking style. Diverse types of left- as well as right-brain thinking skills are integrated to create good software. Abstraction, logic, reduction, critical thinking, etc., are considered as left-brain activities whereas concretization, intuition, creativity, holistic thinking, etc., constitute right-brain thinking. Only interesting persons can develop interesting software. 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. Hence, like all long-term system development activities, software development also requires several cycles of left- and right-brain activities: abstraction and concretization, logic and intuition, critical thinking and creativity, reflection and experimentation, micro-scoping and macro-scoping, as well as reduction and holistic thinking respectively. Software development is a whole-brain activity.

The whole process requires an integration of the ability of abstract conceptualization, and an active experimentation with concrete experiences and reflective observation [152]. Software development leverages developers’ strengths in varied types of intelligences as identified in
Sternberg’s *Theory of Triarchic Intelligence* [153], Herrmann’s *Four Quadrant Model of the Brain* [154] and also Gardner’s *Multiple Intelligence Theory* [155].

In addition, to the ability to solve ill-defined problems, it is imperative to *develop diverse types of thinking skills, comprising whole-brain activity, among software professionals*. Our proposed *framework of pedagogical engagements in software development education, discussed in Section 8.3, aims to offer such whole brain engagement*.

**Section 3.11: Revised Taxonomy of Core Competencies for Software Developers**

Integrating all our earlier theoretical and empirical studies about core competencies, recommendations of various organizations as well as researchers, the first version of our earlier taxonomy, and our latest understanding of the distinguishing features of the software development activity, we have further revised our list of desired core competencies for software developers. With reference to the specific context of software development, *we created a comprehensive set of thirty-three competencies (Appendix A2)*. It *subsumed and expanded the thirty-five competencies of Table A3.2 (Appendix A3)*. Some additional very important competencies were also included in this comprehensive set of Appendix A2. These included - *curiosity, domain competence, abstraction, algorithmic thinking, knowledge of physical and natural world and intercultural knowledge, reflection, self acceptance and self regulation, and workload management*.

*Through further grouping based on logical closeness, hierarchy, and/or dependency, we have further reduced the set from thirty-three to twelve core competencies* (Table 3.1) *each with extended meanings*. The competencies in the comprehensive set are *subsumed within one or more of the revised set of twelve competencies*. We have dropped ‘wealth creation skills’ from our revised core set, as this was categorized in the lowest ranked category of competencies based on the rank given by our respondents, and it also does not integrate well within our final twelve competencies.
Table 3.1: Core competencies for software developers

<table>
<thead>
<tr>
<th>Twelve core competencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Technical competence</td>
</tr>
<tr>
<td>2. Computational thinking competence</td>
</tr>
<tr>
<td>3. Domain competence</td>
</tr>
<tr>
<td>4. Communication competence</td>
</tr>
<tr>
<td>5. Complex problem solving competence</td>
</tr>
<tr>
<td>6. Attention to details</td>
</tr>
<tr>
<td>7. Critical and reflective thinking</td>
</tr>
<tr>
<td>8. Creativity and innovation</td>
</tr>
<tr>
<td>9. Curiosity</td>
</tr>
<tr>
<td>10. Decision making perspective</td>
</tr>
<tr>
<td>11. Systems-level perspective</td>
</tr>
<tr>
<td>12. Intrinsic motivation to create/improve artifacts</td>
</tr>
</tbody>
</table>

Annexure A4 gives the mapping of thirty-five competencies of Table A3.2 (Appendix A3) with this reduced inclusive set of Table 3.1. The popular beliefs about the competencies, and also curriculum, place maximum importance on technical, communication competencies, and/or problem solving competencies. This table very strongly brings out the utmost importance of the development of a systems-level perspective among software developers. Seventeen out of the thirty-five of our earlier identified competencies relate to it. Ten out of thirty-five competencies relate to reflective thinking and nine relate to critical thinking. Eight out of thirty-five competencies relate to decision making perspective. Seven competencies relate to communication competence and also intrinsic motivation to create/improve artifacts. Six competencies relate to computational thinking, curiosity, and domain competence. Attention to detail and Creativity and innovation relate with five competencies. All these are emerging as important goals for education programs for future software developers.

These results are significantly different from all earlier results, including our own. Surely, like other engineers, problem solving skills are important for software developers (ref: Table 2.1). However the nature of the problems they are required to solve are significantly different from other engineering disciplines. Hence, their problem solving skills needs to be driven and conditioned by a systems-level perspective, critical and reflective thinking, decision making perspective, curiosity, intrinsic motivation to create/improve artifacts, curiosity, attention to detail, and creativity and innovation.
Based on several models of organizing the competencies and models related of learning and human development, we have also revised the structure of our taxonomy of core competencies. We posit that the three categories of our earlier taxonomy, given in Table 2.5, have interdependence and are not orthogonal. Consequently, in our revised taxonomy, we do not consider these three categories of competencies as orthogonal dimensions. As there is an interdependence of these categories, we model these competencies as a three-tier taxonomy of core competencies. We have also classified the revised set of twelve core competencies into three-tier taxonomy, as given in Table 3.2. It includes five basic competencies, three competency driver-habits of mind, and four competency conditioning attitudes and perspectives. This is also reproduced as Table 8.1. The arrows in this table indicate the direction of influence.

**Table 3.2:** Three-tier taxonomy of core competencies for software developers

<table>
<thead>
<tr>
<th>Basic Competencies</th>
<th>Competency Driver-Habits of Mind</th>
<th>Competency Conditioning Attitudes and Perspectives</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Computational thinking competence</td>
<td>7. Critical and reflective thinking</td>
<td>10. Decision making perspective</td>
</tr>
<tr>
<td>4. Communication competence</td>
<td></td>
<td>12. Intrinsic motivation to create/improve artifacts</td>
</tr>
<tr>
<td>5. Complex problem solving competence</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

With reference to our three-tier taxonomy given in Table 3.2, the basic competencies are necessary for software developers to contribute to the development of useful and quality software. The highly pervasive core competency driver-habits of mind are necessary to continuously develop, refine, and enhance the basic five competencies. These in turn also help in developing even more useful and high quality software. The most pervasive and enduring competency conditioning attitudes and perspectives create necessary conditions for creating meaningful software and wiser professional software developers. Thus, these competency conditioning attitudes and perspectives guide and regulate the application of competency driver-habits of mind. This in turn helps to create meaningful, appropriate, ethical, and very high quality software. The competency driver-habits of mind are also necessary to continuously develop, refine, and evolve the highest level competency conditioning attitudes and perspectives.
**Basic Competencies**

The basic competence for software developers includes skill, rules, and knowledge related to various technical activities of software development, computational thinking, application domains, communication, and general purpose complex ill-defined problem solving. These contribute to the development of useful and quality software.

**Technical competence** is manifested in the practical as well as intuitive understanding required for the executing various technical tasks related to software development. Computational thinking as an approach to problem solving, creating services, interfaces, and behaviors, and also understanding human behavior. Understanding layers of data and process abstraction forms the core for computational thinking.

Since, application domains of software include all kinds of domains, it is imperative for software developers to understand the concerns, focus, aim, knowledge structures, and thinking approaches of application domains. The communication competence for software developers is significantly different from the communication competence for sales professionals. It is essential for understanding the needs of the consumers, the difficulties of their clients and co-developers. It is required for knowledge acquisition as well as knowledge sharing.

Software development is not about finding answers to well defined problems but solving complex ill-defined problems. Performance on well-defined problems is not correlated with performance on ill-defined problems. Hence, education processes need to give special emphasis on this.

**Competency Driver-Habits of Mind**

Software development is more of a cognitive activity rather than a construction activity. With respect to the multifaceted activities of software development, three mental habits: attention to details, critical and reflective thinking, and creativity and innovation, have been identified as the most important for software developers. These habits contribute to continuously develop, refine, and enhance the basic competencies and create more useful and high quality software.
Inconspicuous nature of software and the necessity of thoroughness, long attention spans, consistency, etc., make ‘attention to detail,’ the most essential mental habit for soft developers. **Critical thinking** is necessary for controlling errors in logical and analytical reasoning at various stages of software development. Software development is essential an evolutionary activity that requires **continuous reflection** about the product as well as processes to uncover and alter the limitations of both. Much of software is increasingly becoming concerned about user’s experience.

**Creative** people are needed to design of new innovative software products for users and new procedures and tools for software developers, as well as management of software development. Development of these habits has to be put as a core learning outcome of all courses.

**Competency Conditioning Attitudes and Perspectives**
Attitudes and perspectives affect a professional’s motivation, expectation, and also ability to practice. Curiosity, decision making perspective, systems-level perspective, and intrinsic motivation to create/improve artifacts are especially important with reference to the requirements of the profession of software development. These attitudes and perspectives guide and regulate the application of competency driver-habits of mind to create meaningful, appropriate, ethical, and very high quality software.

**Curiosity** is recognized as a source of critical thinking and also creativity. Software developers are required to have a high level of curiosity to learn ‘how things work,’ ‘how to create things that work,’ and also find out ‘what may be consequences and risks.’ 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.

**Decision making** is about choosing intelligently among less than perfect possibilities. 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.
**Systems thinking** is seeing wholeness, seeing interrelationships rather than individual things. Software developers use systems to develop systems for supporting systems. Many software systems are socio-technical systems and the software development systems are essentially social systems.

Intrinsically motivated state has been found to more conducive to creativity. Hence, computing students’ intrinsic motivation for creativity needs to be enhanced for creating conditions for self actualization through creation.

In the next three chapters we discuss the meaning and relevance of these twelve competencies in the context of software development and the education of software developers.