CHAPTER

5 HUMAN ASPECTS OF SOFTWARE ENGINEERING

In a special issue of *IEEE Software*, the guest editors [deS09] make the following observation:

> Software engineering has an abundance of techniques, tools, and methods designed to improve both the software development process and the final product. However, software isn't simply a product of the appropriate technical solutions applied in appropriate technical ways. Software is developed by people, used by people, and supports interaction among people. As such, human characteristics, behavior, and cooperation are central to practical software development.
>
> Without a team of skilled and motivated people, success is unlikely.

KEY CONCEPTS

QUICK LOOK

What is it? At the end of the day, *people* build computer software. The human aspects of software engineering often have as much to do with the success of a project as the latest and greatest technology.

Who does it? Individuals and teams do software engineering work. In some cases, one person has much of the responsibility, but in most industry-grade software efforts, a team of people does the work.

Why is it important? A software team will be successful only if the dynamics of the team are right. It is essential for software engineers on a team to play well with their colleagues and with other product stakeholders.

What are the steps? First, you need to try to emulate personal characteristics of successful software engineers. Next, you should appreciate the complex psychology of software engineering work so that you can navigate your way through a project without peril. Then, you need to understand the structure and dynamics of software teams. Finally, you should appreciate the impact of social media, the cloud, and other collaborative tools.

What is the work product? Better insight into the people, the process, and the product.

How do I ensure that I've done it right? Spend the time to observe how successful software engineers do their work, and tune your approach to take advantage of the strengths they project.

5.1 Characteristics of a Software Engineer

So you want to be a software engineer? Obviously, you need to master the technical stuff, learn the skills required to understand the problem, design an effective solution, build the software, and test it in an effort to develop the highest-quality products possible. You need to manage change, communicate with stakeholders, and use appropriate tools as needed. We will discuss these things at great length later in this book.

But there are other things that are equally important—the human aspects that will make you effective as a software engineer. Erasmus [Era09] identifies seven traits that are present when a software engineer exhibits "superprofessional" behavior.

An effective software engineer has a sense of *individual responsibility*. This implies a drive to deliver on her promises to peers, stakeholders, and her management. It implies that she will do what needs to be done, when it needs to be done in an overriding effort to achieve a successful outcome.

An effective software engineer has an *acute awareness* of the needs of other team members, the stakeholders requesting changes to an existing software solution, and the managers who have overall control of the project. He observes the environment in which people work and adapts his behavior to take both into account.

An effective software engineer is *brutally honest*. If she sees a flawed design, she points out the flaws in a constructive but honest manner. If asked to distort facts about schedules, features, performance, or other product or project characteristics, she opts to be realistic and truthful.

An effective software engineer exhibits *resilience under pressure.* Software engineering is always on the edge of chaos. Pressure comes in many forms—changes in requirements and priorities, demanding stakeholders, and overbearing managers. An effective software engineer manages pressure so that his performance does not suffer.

An effective software engineer has a *heightened sense of fairness.* She gladly shares credit with her colleagues. She tries to avoid conflicts of interest and never acts to sabotage the work of others.

An effective software engineer exhibits *attention to detail.* This does not imply an obsession with perfection. He carefully considers the broader criteria (e.g., performance, cost, quality) that have been established for the product and the project in making his daily technical decisions.

Finally, an effective software engineer is pragmatic. She recognizes that software engineering is not a religion in which dogmatic rules must be followed, but rather a discipline that can be adapted based on the circumstances at hand.

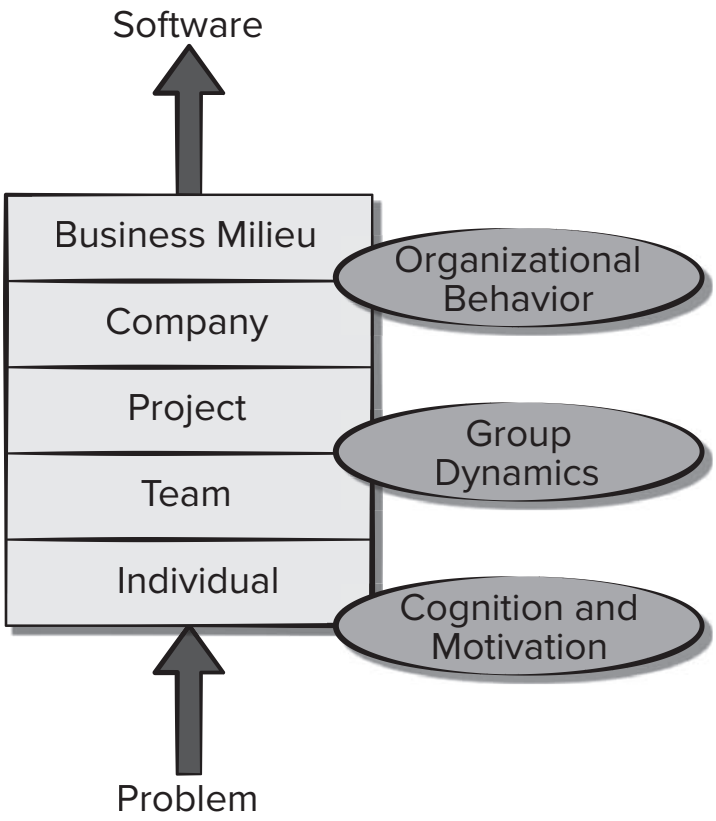
5.2 The Psychology of Software Engineering

In a seminal paper on the psychology of software engineering, Bill Curtis and Diane Walz [Cur90] suggest a layered behavioral model for software development (Figure 5.1). At an individual level, software engineering psychology focuses on recognition of the problem to be solved, the problem-solving skills required to solve it, and the motivation to complete the solution within the constraints established by outer layers in the model. At the team and project levels, group dynamics becomes the

FIGURE 5.1

A layered behavioral model for software engineering (adapted from [Cur90])

Source: Adapted from Curtis, Bill, and Walz, Diane, "The Psychology of Programming in the Large: Team and Organizational Behavior," Psychology of Programming, Academic Press, 1990.



dominating factor. Here, team structure and social factors govern success. Group communication, collaboration, and coordination are as important as the skills of an individual team member. At the outer layers, organizational behavior governs the actions of the company and its response to the business milieu.

5.3 THE SOFTWARE TEAM

In their classic book *Peopleware,* Tom DeMarco and Tim Lister [DeM98] discuss the cohesiveness of a software team:

> We tend to use the word *team* loosely in the business world, calling any group of people assigned to work together a "team." But many of these groups just don't behave like teams. They may not have a common definition of success or any identifiable team spirit. What is missing is a phenomenon that we call *jell.*
>
> A jelled team is a group of people so strongly knit that the whole is greater than the sum of the parts. . . .
>
> Once a team begins to jell, the probability of success goes way up. The team can become unstoppable, a juggernaut for success. . . . They don't need to be managed in the traditional way, and they certainly don't need to be motivated. They've got momentum.

DeMarco and Lister contend that members of jelled teams are significantly more productive and more motivated than average. They share a common goal, a common culture, and in many cases, a "sense of eliteness" that makes them unique.

There is no foolproof method for creating a jelled team. But there are attributes that are normally found in effective software teams.[1] Miguel Carrasco [Car08] suggests that an effective software team must establish a *sense of purpose*. An effective team must also inculcate a *sense of involvement* that allows every member to feel that his skill set and contributions are valued.

An effective team should foster a *sense of trust.* Software engineers on the team should trust the skills and competence of their peers and their managers. The team should encourage a *sense of improvement,* by periodically reflecting on its approach to software engineering and looking for ways to improve their work.

The most effective software teams are diverse in the sense that they combine a variety of different skill sets. Highly skilled technologists are complemented by members who may have less technical background but are more empathetic to the needs of stakeholders.

But not all teams are effective and not all teams jell. In fact, many teams suffer from what Jackman [Jac98] calls "team toxicity." She defines five factors that "foster a potentially toxic team environment": (1) a frenzied work atmosphere, (2) high frustration that causes friction among team members, (3) a "fragmented or poorly coordinated" software process, (4) an unclear definition of roles on the software team, and (5) "continuous and repeated exposure to failure."

To avoid a frenzied work environment, the team should have access to all information required to do the job. Major goals and objectives, once defined, should not be modified unless absolutely necessary. A software team can avoid frustration if it is given as much responsibility for decision making as possible. An inappropriate process (e.g., unnecessary or burdensome work tasks or poorly chosen work products) can be avoided by understanding the product to be built and the people doing the work and by allowing the team to select the process model. The team itself should establish its own mechanisms for accountability (technical reviews[2] are an excellent way to accomplish this) and define a series of corrective approaches when a member of the team fails to perform. And finally, the key to avoiding an atmosphere of failure is to establish team-based techniques for feedback and problem solving.

In addition to the five toxins described by Jackman, a software team often struggles with the differing human traits of its members. Some people gather information intuitively, distilling broad concepts from disparate facts. Others process information linearly, collecting and organizing minute details from the data provided. Some team members are comfortable making decisions only when a logical, orderly argument is presented. Others are intuitive, willing to make a decision based on "feel." Some work

1 Bruce Tuckman observes that successful teams go through four phases (forming, storming, norming, and performing) on their way to becoming productive (http://en.wikipedia.org/wiki/Tuckman%27s_stages_of_group_development).

2 Technical reviews are discussed in detail in Chapter 16.

hard to get things done long before a milestone date, thereby avoiding stress as the date approaches, while others are energized by the rush to make a last-minute deadline. Recognition of human differences, along with other guidelines presented in this section, provide a higher likelihood of creating teams that jell.

5.4 TEAM STRUCTURES

The "best" team structure depends on the management style of your organization, the number of people who will populate the team and their skill levels, and the overall problem difficulty. Mantei [Man81] describes a number of project factors that should be considered when planning the structure of software engineering teams: (1) difficulty of the problem to be solved, (2) "size" of the resultant program(s) in lines of code or function points,[3] (3) time that the team will stay together (team lifetime), (4) degree to which the problem can be modularized, (5) required quality and reliability of the system to be built, (6) rigidity of the delivery date, and (7) degree of sociability (communication) required for the project.

Over the past decade, agile software development (Chapter 3) has been suggested as an antidote to many of the problems that have plagued software project work. The agile philosophy encourages customer satisfaction and early incremental delivery of software, small highly motivated project teams, informal methods, minimal software engineering work products, and overall development simplicity.

A small, highly motivated project team, also called an *agile team,* adopts many of the characteristics of successful software project teams discussed in Section 5.3 and avoids many of the toxins that create problems. However, the agile philosophy stresses individual (team member) competency, coupled with group collaboration as critical success factors for the team. Channeling creative energy into a high-performance team must be a central goal of a software engineering organization. Cockburn and Highsmith [Coc01a] suggest that "good" software people can work within the framework of any software process, and weak performers will struggle regardless. The bottom line, they contend, is that "people trump process" but that even good people can be hampered by an ill-defined process and poor resource support. We agree.

To make effective use of the competencies of each team member and to foster effective collaboration through a software project, agile teams are *self-organizing.* A self-organizing team does not necessarily maintain a single team structure. The team makes the changes needed to its structure to respond to the changes in the development environment or changes in the evolving engineering problem solution.

Communication between team members and all project stakeholders is essential. Agile teams often have customer representatives as team members. This fosters respect among the developers and stakeholders, as well as providing avenues for timely and frequent feedback on the evolving products.

3 Lines of code (LOC) and function points are measures of the size of a computer program and are discussed in Chapter 24.

SAFEHOME

Team Structure

The scene: Doug Miller's office prior to the initiation of the *SafeHome* software project.

The players: Doug Miller (manager of the *SafeHome* software engineering team), Vinod Raman, Jamie Lazar, and other members of the product software engineering team.

The conversation:

Doug: Have you guys had a chance to look over the preliminary info on *SafeHome* that marketing has prepared?

Vinod (nodding and looking at his teammates): Yes. But we have a bunch of questions.

Doug: Let's hold on to that for a moment. I'd like to talk about how we're going to structure the team, who's responsible for what . . .

Jamie: I'm really into the agile philosophy, Doug. I think we should be a self-organizing team.

Vinod: I agree. Given the tight time line and some of the uncertainty, and the fact that we're all really competent [laughs], that seems like the right way to go.

Doug: That's okay with me, but you guys know the drill.

Jamie (smiling and talking as if she was reciting something): We make tactical decisions, about who does what and when, but it's our responsibility to get product out the door on time.

Vinod: And with quality.

Doug: Exactly. But remember there are constraints. Marketing defines the software increments to be produced—in consultation with us, of course.

5.5 THE IMPACT OF SOCIAL MEDIA

E-mail, texting, and videoconferencing have become ubiquitous activities in software engineering work. But these communication mechanisms are really nothing more than modern substitutes or supplements for the face-to-face contact. Social media is different.

Begel [Beg10] and his colleagues address the growth and application of social media in software engineering when they write:

> The social processes around software development are . . . highly dependent on engineers' abilities to find and connect with individuals who share similar goals and complementary skills, to harmonize each team member's communication and teaming preferences, to collaborate and coordinate during the entire software lifecycle, and advocate for their product's success in the marketplace.

In some ways, this "connection" can be as important as face-to-face communication. The value of social media grows as team size increases and is magnified further when the team is geographically dispersed.

Social networking tools (e.g., Facebook, LinkedIn, Slack, Twitter) allow degrees-of-separation connections among software developers and related technologists. This allows "friends" on a social networking site to learn about friends of friends who may have knowledge or expertise related to the application domain or problem to be solved. Specialized private networks built on the social networking paradigm can be used within an organization.

It is very important to note that privacy and security issues should not be overlooked when using social media for software engineering work. Much of the work performed by software engineers may be proprietary to their employer and disclosure could be very harmful. For that reason, the distinct benefits of social media must be weighed against the threat of uncontrolled disclosure of private information.

5.6 Global Teams

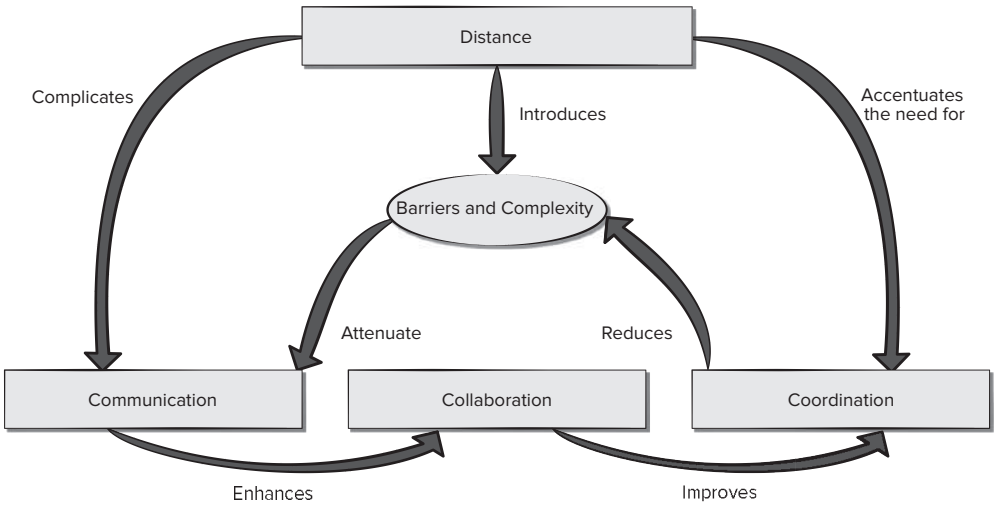
In the software domain, globalization implies more than the transfer of goods and services across international boundaries. For the past few decades, an increasing number of major software products have been built by software teams that are often located in different countries. These global software development (GSD) teams have unique challenges that include coordination, collaboration, communication, and specialized decision making. Approaches to coordination, collaboration, and communication are influenced by the team structure that has been established. Decision making on all software teams is complicated by four factors [Gar10a]:

- Complexity of the problem.
- Uncertainty and risk associated with the decision.
- The law of unintended consequences (i.e., a work-associated decision has an unintended effect on another project objective).
- Different views of the problem that lead to different conclusions about the way forward.

For a GSD team, the challenges associated with coordination, collaboration, and communication can have a profound effect on decision making. Figure 5.2 illustrates the impact of distance on the challenges that face a GSD team. Distance complicates

FIGURE 5.2

Factors affecting a GSD team



communication, but, at the same time, accentuates the need for coordination. Distance also introduces barriers and complexity that can be driven by cultural differences. Barriers and complexity attenuate communication (i.e., the signal-to-noise ratio decreases). The problems inherent in this dynamic can result in a project that becomes unstable.

5.7 SUMMARY

A successful software engineer must have technical skills. But, in addition, he must take responsibility for his commitments, be aware of the needs of his peers, be honest in his assessment of the product and the project, be resilient under pressure, treat his peers fairly, and exhibit attention to detail.

The psychology of software engineering includes individual cognition and motivation, the group dynamics of a software team, and the organizational behavior of the company. A successful ("jelled") software team is more productive and motivated than average. To be effective, a software team must have a sense of purpose, a sense of involvement, a sense of trust, and a sense of improvement. In addition, the team must avoid "toxicity" that is characterized by a frenzied and frustrating work atmosphere, an inappropriate software process, an unclear definition of roles on the software team, and continuous exposure to failure.

Agile teams subscribe to the agile philosophy and generally have more autonomy than more conventional software teams with rigid member roles and external management control. Agile teams emphasize communication, simplicity, feedback, courage, and respect.

Social media tools are becoming an integral part of many software projects, providing services that enhance communication and collaboration for a software team. Social media and electronic communication are particularly useful for global software development where geographic separation can precipitate barriers to successful software engineering.

PROBLEMS AND POINTS TO PONDER

5.1. Based on your personal observation of people who are excellent software developers, name three personality traits that appear to be common among them.

5.2. How can you be "brutally honest" and still not be perceived (by others) as insulting or aggressive?

5.3. How does a software team construct "artificial boundaries" that reduce their ability to communicate with others?

5.4. Write a scenario in which the *SafeHome* team members make use of one or more forms of social media as part of their software project.

5.5. Referring to Figure 5.2, why does distance complicate communication? Why does distance accentuate the need for coordination? What types of barriers and complexities are introduced by distance?