29

Human-Centered System Development

29.1 Introduction

In the previous edition of this handbook, this chapter opened with the declaration, “The last decade has seen a significant change in the information technology landscape, resulting in a fundamental shift in the activities and skills required of technology professionals and teams.” Those words were typed at about the same time that Facebook was being coded in a Harvard dorm room, 3 years before the first iPhone app launched, and 4 years before the Android source code was released. In only a few years, the landscape of information technology (IT) and systems development has again fundamentally changed.

With the evolution of computer languages, target platforms, and development methods, the skills demanded of IT professionals also evolve. In addition to maintaining and expanding technical skills, most IT professionals must communicate with an increasing technologically literate user base, navigate faster development cycles, and work in complex interdisciplinary teams. There is, of course, still the mystical archetype of the lone programmer huddled in the basement coding through the night with pizza boxes piled high. Today’s zeitgeist, however, also recognizes that this same programmer might become the next tech company’s Chief Executive Officer; and the project that begins as a loner’s experiment might become a venture capitalist’s target.

The success stories of today’s social web, where every programmer is a potential millionaire, paint a positive view of the state of systems and software development. Statistics across the industry, however, raise more concern. While some question the study methods and criteria, the often-quoted Standish CHAOS reports signal ongoing industry dysfunction. The CHAOS 2000 report found that 23% of the projects studied essentially failed (Standish Group, 2001). The CHAOS Summary 2009 (Standish Group, 2010) concluded that just 32% of all projects were delivered on time, on budget, with required features and functions; that 44% of projects were “challenged” (e.g., they were considered late, over budget, and/or with less than the required features and functions); and that 24% failed (e.g., were canceled or just never used) (Standish Group, 2009).

These and other studies highlight different failure rates for different project types, but importantly also highlight the common factors contributing to these failures. These reasons often include lack of user involvement, poor project planning and execution, lack of executive support, stakeholder politics, an
overemphasis on process activities, and poor communication or lack of shared understanding between business and technology groups (Standish Group, 2001, 2010; Schneider, 2002; Charette, 2005; Dyba, 2005; Kelly, 2007; Krigsman, 2009; McCafferty, 2010). Even with our best development methodologies and technological sophistication, the human dimension of systems and software development remains the key element to success.

At the most basic level, the output from any development project emerges from the conversations and collaboration among individuals working together over time. As such, success can be influenced by the ability to manage the following types of human dynamics:

- **Individual development**: Today’s technology professional must be a lifelong learner of different application development frameworks, platforms, programming and markup languages, and development tools. Individuals, therefore, must have self-awareness about personal learning style and make strategic choices about whether and where to specialize and generalize. Like all professionals, technologists should also have a sense of their strengths, weaknesses, and preferences in how they work with others. Do they prefer working in a structured, predictable environment, or do they like flexibility and change? Do they prefer to work alone or with others? Too often, technology professionals are drawn to the technology aspects of the work, without fully considering the people-oriented dimension of the career choices being made. Furthermore, technical mentors may not always help their technical protégées navigate these more personal questions of self-development.

- **Team collaboration**: The range of talents required of an IT development team today is both broad and deep. Today’s development teams include functional experts, business analysts, architects, programmers, content strategists and writers, data modelers and scientists, network and systems administrators, and project managers. In one study of IT-focused organizations, only 12% of IT professionals surveyed reported their role as being programmer or developer (Tucker et al., 2004). The result is that IT teams, which once shared a common technical base for building relationships, are coming together from separate specialties and backgrounds within their own fields. This diversity creates profound opportunities for collaboration if team members are able to transfer knowledge internally and communicate effectively, but it can also lead to miscommunication and misfires if not managed well.

- **User communication**: Technical individuals and teams rarely work in a vacuum. Even in teams where a project manager or business analyst serves as a primary point person for customers, developers need to interpret requirements and understand how their work will meet the user’s or customer’s needs. Self-perception, self-expression, interpersonal skills, decision making, and stress management are all factors of emotional intelligence (EQ) that shape the ability to work effectively with customers (Stein and Book, 2006), and motivational states and cognitive preferences can dramatically shape understanding and interpretation. Deciding how users are to be treated and engaged in a systems development process, and then following through even when conflict is encountered, is a critical factor in the design of a technology program.

These realities and dimensions of the systems development experience call us to again reexamine IT and systems development work from a human dynamics perspective. This chapter considers the underlying principles informing human-centered systems development and considers applications and directions for future research.

### 29.2 Principles and Applications

Understanding human-centered systems development begins with defining the term itself. For the purpose of this chapter, human-centered systems development considers the people side of systems development: the personality dynamics of IT professionals, the people-related dynamics of IT teams, and the interpersonal elements of the systems development life cycle.
29.2.1 Personality Dynamics of IT Professionals

Many research studies have used psychometric tools to describe the personality dynamics of IT professionals. While these studies have ranged in design, they collectively reveal that certain personality characteristics tend to be more and less prevalent among IT specialists than the general population (Mastor and Ismael, 2004; Tucker et al., 2004; Williamson et al., 2005; Lounsbury et al., 2007). These distinctive characteristics include:

- A higher preference for logical, impersonal, and objective analysis, sometimes captured as tough-mindedness
- Higher levels of intrinsic motivation, personal control, and emotional resilience
- Higher levels of independence
- High desire for supervisor support, but without the desire for increased levels of control by those same supervisors (consistent with independence)
- Lower levels of rules orientation and conformity, which in some psychological tests are characterized as conscientiousness
- Lower levels of political savvy and interest, sometimes known as image management
- Lower needs for interpersonal connection and inclusion in other people’s social activities

The practical impact of this research is that understanding these personality characteristics both reveals the common strengths that information technologists might bring to their work and points to possible systemic weaknesses or blind spots that may occur when strengths are overdone. The strengths of logic, objectivity, independence, and control can yield powerful technology solutions, but they also make it understandable why the mythos of the lone, independent, even awkward coder is so persistent and so often encountered in real life.

When overdone, the strengths most commonly associated with IT professionals can be perceived as a lack of empathy for the user, a lack of concern for relationship management aspects of the development process, or a bias toward working on interesting individual tasks rather than attending to more routine team needs.

Independence and personal control over work are often repeated themes when referencing technology professional preferences. Developing high-quality software requires attention to detail and clear controls. Controlling quality, however, is different from controlling the people developing the code, particularly when those people value high levels of independence and personal autonomy. Structured project management addresses software quality because it introduces controls and testing check points that prevent low quality code from being committed and released. Too much management, however, can stifle innovation and creativity among the people creating that code. An awareness of these human dynamics is vital for leaders seeking ways to control the code without overly controlling the coders.

Interpersonal and relationship management skills are vital in any systems development effort, but while increasingly referenced in curriculum models, are not central in the formal education of most technology professionals. Once in the workplace, many technology professionals tend to avoid training that focused on the development of interpersonal and communication skills, believing that such investment of time and resources is not as beneficial as learning a new technical skill.

At the same time, personal experience suggests that many technical specialists have at least occasionally experienced conflict with team members, misunderstandings with the boss, or confusion with end users or clients. Many of these interpersonal issues can be traced to fundamental style or communication differences rather than a technical deficiency. Technical professionals who recognize these root causes are better prepared to manage problems effectively, leading to better technical work. Understanding what one brings to the team from a human dynamics standpoint, and how that might mesh or clash with coworkers, is a key starting point.
29.2.2 IT Team Dynamics

The increasing specialization of roles in systems development brings together teams of engineers, business analysts, architects, developers, content strategists, testers, and project managers to guide technology projects to fruition. Larger team sizes lead to more complexity, and while many acknowledge that project success is often linked to the “people side” of the equation, there are no agreed upon algorithms for project success. Project success in the real world, in fact, comes from carefully balancing a variety of factors. Just as overdone personality strengths become weaknesses, an overemphasis on any particular project or team factor becomes a liability as well.

Human-centered systems development requires understanding factors and models from a variety of fields, including psychology, process improvement, project management, and organization development. This section reviews a number of factors important to the success of technology teams (Ahern et al., 2001; Rutledge and Tucker, 2006; Basri and O’Connor, 2011).

29.2.2.1 Mission and Goal Clarity

A shared understanding of the goal of any particular project keeps both the project team and user base focused on the end state and provides a stable benchmark when tough trade-offs and decisions must be made. When different segments of a project team hold different understandings of the mission, disconnects and conflicts may occur without people actually realizing their cause.

For example, drawing from a true case study, picture a large-scale software radio development program with teams distributed across a number of sites. While the program had an overarching and rather lengthy mission statement, people generally reported either not knowing what it was, or, not really understanding what it meant. To explore this, two teams in the program were asked to develop a statement summarizing the overarching mission of the project. One team responded that the goal was to create a low-cost piece of integrated software and hardware that would require little maintenance once fielded. The second team responded that the goal was to develop cutting-edge technology that would advance the software radio field and spark new technology investments. There had been a variety of conflicts between these two teams, and integration between systems had consistently encountered difficulties and delays. Unfortunately, the teams had not ever discovered their fundamental disconnect, leading to conflicts that were ultimately based on foundationally different ideas of the project mission.

Any factor that is overemphasized can, of course, lead to blind spots. A shared understanding of a mission provides teams with a target against which to mutually align and a common goal around which to rally. When there is too much emphasis on mission achievement, however, the team can miss shifts in the environment that led to the initial need, leading to possible obsolescence in the mission itself. Additionally, an overemphasis on goal orientation can lead to high levels of team stress if too much pressure is imposed.

29.2.2.2 Structure and Leadership

Structure and leadership includes such formal elements as role clarity, reporting relationships, and organizational structure, and informal elements such as power dynamics, personal accountability, and empowerment. Structure provides information about hierarchies, governance, and team organization. Leadership can be either formal as managers give direction, make decisions, and allocate resources; or informal and emergent as different team members lead each other as the project unfolds (Howell et al., 1986; Carter, 2004; Rutledge and Tucker, 2006; Senge, 2006).

Structure and leadership often evolve as organizations and teams grow. The team that begins as two programmers grows into a small team and beyond. Technical questions become supplemented by organizational questions of centralization versus decentralization, and self-directed teams versus matrix relationships. These are important questions about how the team functions and also reveal dynamics about how a technology team engages with others outside the group.
For example, in formal organizations, examining an organization chart (if there is one), can reveal the team's relationship with its users. Who is the person or group responsible for ensuring that the users' voices are heard? Where does this role reside on the chart, and how much power is inherent in the position?

The same teams described earlier that struggled with mission clarity also struggled with user relationships and senior stakeholder buy-in. The team responsible for customer relationships was organizationally located in an entirely different team than the engineering and testing teams, and there were few formal or even informal connections between them. As such, the team responsible for relationship management was listening to the user base, but the structure of the large program did not facilitate information flow from one part of the program to another. One key consequence of this was that the customer felt heard, but when elements of the product were delivered, there were significant usability gaps between what the customer had asked for and what was delivered.

Over time, informal connections between project team members built informal pathways for questions and feedback, but leaders did not make easy transmission of customer feedback a priority, causing inefficiency in communication and a great deal of invisible labor behind the scenes. The informal leadership of team members ultimately evolved into an unwritten “shadow org chart” for information sharing. The degree of match between formal and informal power structures on a team reveals a great deal about the efficiency and effectiveness of team operations, and the degree to which formal leadership is respected and followed.

A very different case study can be seen in the structure and leadership of the open source community that develops the operating system Linux. This is a model where structure and leadership are highly interdependent. The Linux organizational structure is centered on a person rather than a role; Linus Torvalds, the “benevolent dictator of Planet Linux,” (Rivlin, 2003) and a small set of key deputies hold decision-making authority for all elements of a Linux release. Outside this structure, there are loose networks of developers and contributors but no formal organization charts, reporting relationships, or hierarchies.

In this model, a small core of formal leadership is at the center, making the institutional decisions that define the product. In the absence of formal institutional structures, technical and people leadership from Torvalds is cited as a defining factor in the success of the overall enterprise (Rivlin, 2003; Vaughan-Nichols, 2010). Complementing this, informal and emergent leadership is exercised as individual developers choose what features to work on and who to follow; it is this grassroots leadership that ultimately provides the assets from which the formal leadership core can select. While Torvalds may be the central figure, the relationships across this community form an informal network that provides intellectual capital and becomes its own type of organizational structure.

29.2.2.3 Process and Communication

How much team processes are known, agreed on, communicated, and acted upon greatly impacts team effectiveness and morale. While structure defines roles and relationships, process defines how work flows between these roles to lead to decisions and deliverables. Software development models, business processes, workflow, and decision matrices all fall under the category of process. While process and communication are often treated separately in process management, they are fundamentally linked; communication between team members, whether technically or interpersonally mediated, is generally the mechanism by which process is expressed.

In the past, the technology industry attacked the dynamic problems of system development with an increased focus on process, creating a plethora of standards, protocols, and maturity models. As noted previously, however, any strength maximized can become a liability; and process management taken too far can result in an overemphasis on project reporting and process documentation, compromising functional delivery.

This is an area where the pendulum may be swinging, or at least diversifying. In the previous edition of this handbook, this chapter declared, “Every complex endeavor needs an organized plan to keep it
on track.” Since that edition, however, the success of emergent start-ups and open source projects has called this project management assumption into question. For example, Facebook proudly proclaims its preference for “The Hacker Way” through the mantra: “Done is better than perfect.” Facebook has embraced a development process centered upon small iterative releases, with constant testing both to correct errors and to identify the next steps (Zuckerberg, 2012). Free and open source software development also usually does not follow formal process models or plan-driven approaches because it is generally based on volunteer collaboration; yet, these projects can generate high-quality products quite quickly (Magdalenoa et al., 2012).

These development processes are very different from the structured waterfall methods of the past and require different communication approaches and styles. The absence of plan-driven process does not, of course, mean the absence of process. The question is: How do the formal and informal processes and communication pathways within a team support or detract from the team’s success, however that is measured, and how do the processes encourage or discourage the participation of the technical professionals most qualified to do the work? An emphasis on structured, plan-driven process and communication appeals to many professionals. It is predictable, clear, and when managed effectively, efficient. Other professionals are more attracted to the “Hacker Way,” where the product emerges from the unpredictability of trying out new ideas. It is innovative, flexible, and when managed effectively, efficient.

Different process approaches have different impacts that can be both positive and negative. These impacts may also be perceived differently by people with different personalities. What is a positive process practice or communication approach for one person may be problematic for another. Just as engineers must understand the impact of different architecture and design choices, those working in a human-centered systems development context must anticipate how different process and communication approaches are likely to impact different people and plan accordingly. This may mean recruiting people that either enhance or counter-balance different team characteristics, customizing communication for different people based on individual values and styles, and identifying and monitoring the risks that may emerge when specific approaches are too far over-emphasized.

29.2.2.4 Interpersonal Skills

The unequaled mourning over the death of Steve Jobs sent tremors of paradox through the leadership development community. After years of advocating the importance of interpersonal skills in leading organizations and teams, many struggled with the sense of cognitive dissonance that a man so feared would also be so celebrated for his ability to inspire other people. How could this leader, known for his angry outbursts and harsh feedback, be so revered?

A review of leadership articles and blogs published shortly after Jobs’ death reveals a common theme: Jobs’ leadership style worked because of his unique blend of technical and design intelligence, because of his insistence on excellence from everyone around him, and because he was inspirational in how he communicated his vision to others (Isaacson, 2012). Many articles, however, also carried a warning echoed by leadership coaches far and wide: “Don’t try this at home” (Allen, 2011; Nocera, 2011; Riggio, 2012). Jobs’ ability to lead despite his difficult interpersonal style was cited as many as an anomaly; something “mere mortals” should not attempt.

From a practical viewpoint, it is more useful, perhaps, to reframe the situation as a matter of fit. Jobs had a specific style that attracted certain people to him, but there are likely many that self-selected away from him as a result of that same style. Investing in developing interpersonal skills provides the flexibility needed to work well with a range of other people, increasing the opportunities for both leadership and followership.

Interpersonal skills include the ability to listen and demonstrate understanding of others’ views, display trust, build relationships, and support others. Closely linked with communication and leadership, interpersonal skills allow people to manage conflict effectively, negotiate, advocate their views compellingly, and use power to achieve desired ends. Interpersonal skills are not dependent on a particular set
of personality preferences and styles. Read biographies of both Linus Torvald and Steve Jobs, and you will read portraits of very different men. Interpersonal skills result from self-awareness, the ability to read and empathize with others, and the ability to work with others toward a common goal. While “code may win arguments” in hacker culture, it is the combination of both technical and interpersonal skills that brings people together for the argument in the first place.

Psychological type, popularized through the Myers-Briggs Type Indicator (MBTI), is a popular model in understanding a set of different dimensions of personality that can contribute to interpersonal skills (Myers and Myers, 1980, 1995; Kroeger et al., 2002). Like technical skills, interpersonal skills are teachable, and the psychological type model has helped many professionals understand how their personalities impact behavior and their relationships with others (Peslak, 2006).

The MBTI assesses four dimensions of personality: your source of energy (internal or external); your preference for data gathering (detailed and specific, or abstract and global); your preference for decision making (logical and objective, or subjective and values driven); and your orientation to the world (whether you prefer to show others your decisions or your perceptions) (Kroeger et al., 2002).

Preferences often manifest themselves in behaviors, which can either support or irritate relationships with others. As an example, imagine a developer who is energized by the internal world (rather than engaging with others), and when in the external world, prefers a sense of order, structure, and closure (rather than openness and flexibility). Because this developer prefers to communicate final thoughts in a decision-based format, she expects that others generally operate this way as well. Our developer is in a prototype review session with a user who is energized by the external world, and who prefers to bounce ideas around to explore and learn in real time. This user may view interaction periods as open brainstorming sessions, designed to explore new ideas, cover the possibilities, and think out loud.

What might happen as an outcome of this prototype review session? From the developer’s perspective, the meeting resulted in a series of decisions, and a long list of new requirements. From the user’s perspective, the discussion was simply the sharing of ideas and options, and alternatives for consideration, no strings attached. This scenario, which is neither exaggerated nor uncommon, is an example of how personality type can impact relationships. It also reframes the enduring problem of requirement creep, a key issue facing the IT community. Requirements creep is usually described as a problem at the system or project level; however, every need ever articulated for a system came from a human user expressing an individual thought. How that thought is heard and managed is ultimately what leads to the project-level problem of requirement creep. This is only one example. Consider the full range of preference combinations between developers and users, and the complexity and potential impact of human dynamics in the development process becomes even more dramatic.

The MBTI is, of course, only one psychological assessment used to study personality and team style and preference, and it is by no means universally accepted. Some, for example, reject the idea of personality testing altogether. Others reject the MBTI based upon reviews of its psychometric characteristics, such as construct validity and applicability (Pittenger, 1993; Capraro and Capraro, 2002).

Other models with associated assessments that provide personality-related insights across different scales include the “Big Five,” a generic term for assessments that study the five personality characteristics of extraversion, openness, agreeableness, conscientiousness, and emotional stability (McCrae and John, 1992; Gruca and Goldberg, 2007); EQ indicators, which assess a variety of factors related to self-awareness, self-management, social awareness, and relationship management (Petrides and Furnham, 2000; Stein and Book, 2006); and reversal theory, which assesses motivational states, changeability, and emotion (Apter, 2007; Tucker and Rutledge, 2007).

What is the best practice that can be gained from using psychometric tools? Investing the time and effort to learn about personality preferences, and learning how to recognize them in others yields better relationship management skills and less frustration, because having a language with which to talk about differences makes them easier to resolve. This conclusion is also supported by evidence from a study of 40 software development teams, which concluded “how people work together is a stronger predictor of performance than the individual skills and abilities of team members” (Sawyer, 2001).
29.2.3 Systems Development Life Cycle Models

The previous section touched upon process as a critical element of team dynamics. Process includes systems and software development models. Given their importance, this section considers different systems development life cycle models or methodologies, through the lens of the human-centered demands that they place on a development professional. While there are many overlaps between methods, the presentation here is based upon the type of people-driven factors that tend to be called upon the most for that method type.

29.2.3.1 Formal Structured Methods

Formal structured methods, such as the classic waterfall methodology and the risk-based spiral model, are methods that are generally governed by a plan-driven predictable and repeatable process (McConnell, 1996; Boehm, 1998; Ahren, 2001). In the formal structured methods, development occurs through a series of sequential phases, with a major milestone review marking the end of each phase or iteration. While phases differ across projects and different variations of the development model, they generally cover the following types of activities: concept development or needs assessment, requirements analysis, high-level or general design, detailed design, coding/unit test/debugging, system testing, and implementation. Spiral models add an iterative element to the structure, allowing the team to learn between iterations and the user to see some outcomes, while still maintaining the discipline of structured methods.

Formal structured methods work well for low-speed and low-change projects, in which there is a very clear and stable definition of the requirements, the product, and the supporting technology. It is also used for large projects where high quality, low error outputs must be combined with other engineered systems, or for long-term expensive projects where budget and resource planning is vital to keep the project funded and staffed.

Ultimately, the strengths of any method also hold the source of its weaknesses. The discipline of formal methods can lead to a project that is inflexible to changing needs, and an overemphasis on process documentation can take time away from actual project development. Structured development models are likely to appeal to developers and users who have lower tolerance for ambiguity, enjoy the stability of long-term projects, do not mind formal process and documentation, prefer concrete milestones and metrics, and enjoy the process of creating high-quality code. It can be frustrating for users and developers who want to see tangible outcomes more quickly.

29.2.3.2 Agile and Other Adaptive Methods

Agile, adaptive methods are defined here primarily by their orientation toward user involvement and the degree of change and flexibility encouraged in the development process. Agile methods emphasize interpersonal interactions, communication, customer collaboration, rapid production of code, and frequent feedback (Agile Alliance, 2001; Vinekar et al., 2006; Moe et al., 2010; Stettina and Heijstek, 2011; Thibodeau, 2011). In contrast to formal structured methods, agile development methods have been praised for bringing together developers and users and for allowing fluidity in the development process as participants co-learn and evolve their understanding of the true business needs and requirements.

While agile methods have become widely adopted over the past few years, there are critiques as well (Boehm and Turner, 2004; Dyba and Dingsoyr, 2008; Barlow et al., 2011; Gualtieri, 2011). The bias toward rapid development of code can lead to software modules that solve an immediate problem but that are not built with an eye to the customer organization’s business strategies and need for larger-scale integration points. Furthermore, the method is heavily dependent on the presence of knowledgeable users to iterate business requirements in real time. This can lead to solutions that represent the local needs of users that participated in the process, but that may not sufficiently represent the needs of a diverse user base across a larger enterprise.

Agile, adaptive methods require significant interpersonal skills for those engaging frequently with users and comfort with ambiguity, change, and real-time problem solving. It can be frustrating for
those wanting more structure and a broader scope or planning horizon, or for those that are uncomfortable working with users or receiving uncensored feedback in real-time application development environments.

29.2.3.3 Hackathon Method

Some may bristle at the classification of the hackathon as a method, seeing it as an extension of the iterative methods discussed earlier; however, these problem-focused coding approach and events have made a sufficient mark on the programming landscape and culture to be considered separately here. Described as “a contest to pitch, program, and present,” (Leckart, 2012) hackathons bring together programmers and systems engineers to work on specific software problems or to compete for funding for further development. A time-bound variation on the long-standing “code and fix” process, hackathons are characterized by their emphasis on real-time, intensive, outcome-driven coding processes, with developers often using preexisting software resources and toolkits to generate a usable product (such as a new app, or new application programming interface) in a time-boxed period.

Hackathons have become a normal activity for many software firms and programs and have even become a practice in government agencies. In the initial public offering documentation for Facebook, Mark Zuckerberg (2012) described the nature of hacking: “Hacking is also an inherently hands-on and active discipline. Instead of debating for days whether a new idea is possible or what the best way to build something is, hackers would rather just prototype something and see what works.” At hackathon events, a problem or goal is presented, and a time span defined in which to work. At the end, the best solution is generally selected with some kind of acknowledgement or award.

Hackathons require personality factors such as comfort and skill with working informally with new people in a small team, the ability to quickly extend and extrapolate preexisting code and tools to solve particular problems, the ability to work intensely under time pressure, and the ability to present one’s work to others in a compelling way in a short time. Not everyone thrives in these conditions, but these events have proven to be a valuable mechanism by which new software components can be created quickly.

29.3 Research Opportunities and Conclusions

No technical team sets out to develop a system that does not meet user needs. Moreover, no user purposefully sets out to confuse or muddy the requirements that he or she is trying to articulate. In today’s environment of mobile apps and social software, human-centered software is user-driven software, and it is becoming common to refer to interactions with software as “experiences.” Given this, “experience creators” must be increasingly trained to be aligned with both the tangible and emotional needs of an increasingly technologically savvy user base. Research and education are two vital paths to maximize this alignment.

29.3.1 Research Landscape and Opportunities

To better assess the research opportunities in human-centered systems development, it is useful to review the type of research generally conducted across selected different technologically oriented disciplines interested in this area. Primary fields of interest here include information systems, systems engineering, and human–computer interaction.

Information systems is a broad and interdisciplinary academic field, covering a range of topics related to the processing and management of information: technologically, organizationally, and cognitively. Information systems covers such domains as data and information management, enterprise architecture, project management, infrastructure, systems analysis and design, information strategy, cognitive science, and informatics (Topi et al., 2010; Buckland, 2011). The field of information systems provides a broad social and systems level perspective of the intersection between people, technology, and information.
Systems engineering research is also a vital resource for those interested in human-centered systems development. It focuses on the theory, practice, and methods associated with the development of often large socio-technical systems. In the realm of systems engineering, case-based studies concerning personality and team performance is the research that most closely relates to the content of this chapter.

Human–computer interaction research also addresses topics at the intersection of people and technology, but generally looks at this intersection from the users’ perspective, not the developers (Helander, 1997). For example, a search on “human computer interaction emotion” yields multiple articles about the simulation of emotion by computers, and methods for invoking emotions among users, but includes only isolated references to the emotions of information technologists themselves.

Outside these technology-oriented fields, researchers interested in human-centered systems development might look to the fields of organization development, management, and organization and industrial psychology for both theoretical and applied research. Subfields of leadership development, technology innovation, and team dynamics also can yield valuable insights that can be applied to human-centered systems development.

Unfortunately, in many real-world environments, the work of organizational development professionals and IT professionals remains quite distinct, even though they share the same foundational goals of supporting organization change. Organizational development professionals tend to approach problems from the people perspective; IT professionals tend to approach problems from the technical or information flow perspective. The different language and tools used by each type of professional makes cross-fertilization less common in professional settings than one might hope.

Examples of research that would further support the principles and applications presented in this chapter are as follows:

- **Empirically based case studies:** More empirical research using validated psychological assessments is required to solidify the connection between effective human dynamics, communication, and project success. Too often, empirical studies rely not on published and controlled validated assessments (such as the MBTI), but rather, on publicly available “spin-offs,” which while grounded in the same theories, lack the validity to support confidence in study results.

- **Cross-team comparison studies:** Unfortunately, the lack of standardization between existing studies of personality and team project success makes their results difficult to generalize or to usefully compare findings across projects with different characteristics. More studies using similar methods across well-defined but distinct environments are needed to better understand the intersection of personality preferences, development models, and product outcomes and successes over time.

- **Ethnographic research:** Observational studies of development teams at work, including their interaction with each other and with users, would provide additional ground-level evidence that could inform future practice. For example, comparing the actual content of joint applications development sessions or agile-based user-developer discussions against the final specifications of a completed product could yield useful insights about the types of elicitation techniques or conversational arcs that are most effective in yielding useful requirements and efficient user feedback.

Research helps to support the work of practitioners by providing insight into human-centered systems development and its processes, and cross-disciplinary research helps bridge the existing gaps between computer science and the social sciences. The key, however, is to continue to work to bridge the boundaries between academic communities and industry and government so that the research generated in the academic world is communicated in a way that is practical and useful for nonacademic professionals fighting the fires of delayed and at-risk projects in real time. Collaborative research efforts with industry can also help unite today’s curriculum and students with the industry challenges and opportunities of tomorrow.
29.3.2 Teaching Methods

A focus on human-centered systems development points to several themes for educators to consider in developing curricula for technical professionals. Key competencies to develop in these efforts include:

- Self-awareness and learning
- User interaction and communication
- Team dynamics
- Innovation
- Working with ambiguity and uncertainty
- Process management and quality improvement

**Encourage a broad curriculum:** Educators are in a unique position to help technology professionals experience a broad curriculum that includes interpersonal skills development and learning projects that focus on team-based work (Crawley et al., 2007). Teaching skills such as critical thinking and analysis, writing, strategies to learn new development tools and languages as they emerge, and helping students develop the habits of self-awareness and introspection will serve them well in any professional setting.

**Balance theory and practice:** Personality type theorists argue that how people perceive or gather information is the most important factor in determining how people learn. Some people learn best when given a theoretical framework within which to place new information. Others learn best when presented with practical applications or when they are able to interact with tools that allow practice and interaction with the new knowledge. Balancing theory and practice develops concrete applications knowledge, while also providing a broad theoretical foundation from which future applications can be derived. Practice also reminds hands-on students why they self-selected into a field in the first place, particularly important for students who selected into the field because of its practical applications and potential for impact.

**Incorporate human uncertainty into assignments:** Many educators use simulated exercises or case studies to allow programmers to practice programming skills. These methods are even more useful when they integrate real users and user needs into the equation. Projects being completed under the humanitarian free and open source software program are a good example of instances where educators are providing students with real-life software development opportunities. These opportunities are not just isolated exercises, but are full life cycle experiences completed over the course of a semester, giving students exposure to a wide range of activities and interactions that they will encounter in the future as professionals (Morelli et al., 2010).

These projects benefit both students and nonprofit organizations; the students gain invaluable exposure and experience, and the users gain a piece of software that supports a humanitarian-focused nonprofit mission. Research related to the long-term impact that these experiences have on students as they enter their careers would be a useful step to validate the usefulness of these experiences over the long term.

29.3.3 Conclusions

Recognizing the human-centered elements of systems development, including personality preferences, team dynamics, and development models, may help technical professionals select projects or roles that support personal values and areas of personal strength. Some people simply would rather work alone to generate a product that later can be joined with the efforts of others. Other people enjoy the shared nature of actively collaborative efforts. Some developers enjoy the iterative nature of evolutionary prototyping; others find the constant change and uncertainty more frustrating than freeing.

Technology professionals should take the time and be given the experiences that allow them to recognize their personal strengths, preferences, and potential blind spots when selecting jobs or projects. If a developer, for example, chafes against the requirements for structure and documentation that come
with working on a project governed by structured methods and capability maturity models, it is better to know this in advance. Research provides useful context for asking new questions, but in the end, self-knowledge is the foundational key to making more effective decisions: decisions that will impact the individual, the development team, and ultimately the end user.

References


29-13


© 2014 by Taylor & Francis Group, LLC


