1-31-2010

# Making Sense of Software Development and Personality Types

Luiz Fernando Capretz
*University of Western Ontario*, lcapretz@uwo.ca

Faheem Ahmed Dr.
*Thompson River University*, fahmed@tru.ca

# Making Sense of Software Development and Personality Types

**Luiz Fernando Capretz,** *University of Western Ontario*
**Faheem Ahmed,** *United Arab Emirates University*

**By mapping soft skills and psychological traits to the main stages of the software life cycle, the authors claim that assigning people with personality types best suited to a particular stage increases the chances of the project's successful outcome.**

Software is a product of such human activities as problem solving, cognitive information processing, and social interaction. However, people are more complicated and less predictable than computers, thus the complexity of personality entails intricate dynamics that ultimately become an integral, yet often overlooked, part of software development. Sooner or later, major issues relevant to software engineering boil down to the people involved with software production and their personality traits.

Over the past three decades, software engineering has become a very broad field; consequently, the skills necessary to successfully work in this area 30 years ago might no longer apply. For instance, software design has become much more than manipulating formal or rigorous notations—it now revolves around the interaction between designers and users (primarily, the designer's perception of what the user wants, and the user's perception of what he or she really needs). Today, successful software is developed after a tremendous amount of time is spent with the user in the form of prototyping, experimentation, and feedback. In fact, these three processes represent the de facto life cycle of many useful software systems.

Research relating personality styles to software engineering has been scattered and difficult to interpret uniformly. This paucity could indicate that the relationship between software engineering and personality styles is too complex to investigate. For instance, certain personality traits such as introversion/extroversion might have a significant impact on system analysis, but they might not affect the other software life cycle phases. Thus, studies to determine which personality types are more suitable for certain software development activities are of paramount importance.

A major rationale behind this article is to discern connections between personality traits and the process of software development. This interdisciplinary human-centered research incorporates theories about psychological types, human factors, and software engineering. It contributes toward a bridge that links software engineering and software psychology, and it attempts to shed light on several outstanding problems that plague the software industry.

## Myers-Briggs Type Indicator

The Myers-Briggs Type Indicator (MBTI) is a well-known instrument for measuring and understanding individual personality types.[1] It currently ranks among the most popular indicators used in the workplace, establishing four dimensional pairs for assessing personality types: extroversion (E) and introversion (I), sensing (S) and intuition

 Published by the IEEE Computer Society 

(N), thinking (T) and feeling (F), and judging (J) and perceiving (P). We can use these four sets of preferences, selecting one trait from each pair, to delineate a person's preferred type. Table 1 shows the 16 possible configurations, along with percentages of the various types in a representative sample of the US adult population.[1]

### Extroversion (E) and Introversion (I)

While Es prefer looking outward, Is have an inward view. Es are talkative, outgoing, conversation initiators. Is, in contrast, are quiet, reserved, and tend to respond to conversation rather than start it.

### Sensing (S) and Intuition (N)

Although an S individual might need to absorb a whole series of facts in a linear fashion, an N person can take in the same information through abstraction and establish meaning beyond the information captured only by the senses. S individuals dislike new problems unless prior experience shows how to solve them; conversely, N people enjoy solving new problems and dislike performing trivial tasks.

### Thinking (T) and Feeling (F)

The terms *thinking* and *feeling* in this context refer to the process of decision-making. The MTBI scale identifies thinking as the logical way of making a decision, whereas feeling describes the tendency to rely on emotional values as a basis for making decisions. T people are principle-oriented and firm, whereas F people are subjective and have strong interpersonal skills.

### Judging (J) and Perceiving (P)

Judging identifies the tendency to be extremely organized. At the other extreme, a P individual prefers delaying, appears to be disorganized, and seems to be distracted from completing a task until some little bell goes off at the last minute and propels this individual to get the job done. The adherence to deadlines, punctuality, and closure describes J personalities, while the terms open-ended, adaptable and spontaneous apply to P types.

## Previous Studies

To date, only a handful of studies have investigated the relationship between human skills and software development life cycle phases[2,3] or attempted to identify the characteristics of top-performing software developers.[4,5] In fact, Norman Kerth and his colleagues[6] are skeptical about the MBTI's ability to predict who will make a good software engineer because the metric doesn't consider variables such as passion, experience, or financial rewards. Although they're correct about a single personality test's inability to predict success in a field as broad as software engineering, they contradict themselves when they state, "We see zero indication that MBTI preference correlates with job success," but later affirm, "systematically excluding certain types from a team produces an imbalance that is likely to have a poor performance."[6]

**Table 1. The 16 Myers-Briggs Type Indicator (MBTI) types and their distribution among the US adult population.***

| | | | |
|---|---|---|---|
| ISTJ = 11.6% | ISFJ = 13.8% | INFJ = 1.5% | INTJ = 2.1% |
| ISTP = 5.4% | ISFP = 8.8% | INFP = 4.4% | INTP = 3.3% |
| ESTP = 4.3% | ESFP = 8.5% | ENFP = 8.1% | ENTP = 3.2% |
| ESTJ = 8.7% | ESFJ = 12.3% | ENFJ = 2.5% | ENTJ = 1.8% |

* *E = extroversion, I = introversion, S = sensing, N = intuition, T = thinking, F = feeling, J = judging, and P = perceiving*

This debate is far from over. Although researchers have questioned MBTI measures in other contexts,[7] the tool is still one of the most popular for ascertaining personality types, especially because extensive data supports its findings. The instrument itself doesn't predict career success—it merely identifies occupational preferences—but personality has a great impact on a worker's motivation, performance, and retention in the field.[8–11] A common thread running through the results of these and other similar studies is not only the prevalence of I, T, and J types, as opposed to fewer E, F, and P types, but also almost as many S as N types among software professionals.

Although empirical studies suggest that the MBTI poles are related to software engineering, they don't specify at which phase of the software life cycle they occur or how they're related. Despite early interests in the importance of human factors in software development—in particular, the personal characteristics of humans involved in software engineering processes—such factors have been neglected, thus hindering process improvements. A more focused approach might help identify at which software life cycle phase a particular personality type has the most significant impact.

## Mapping Job Requirements and Soft Skills to Personality Types

Software engineering is roughly characterized as a set of activities comprising system analysis, design, programming, testing, and maintenance. Logically, these different tasks combine to achieve

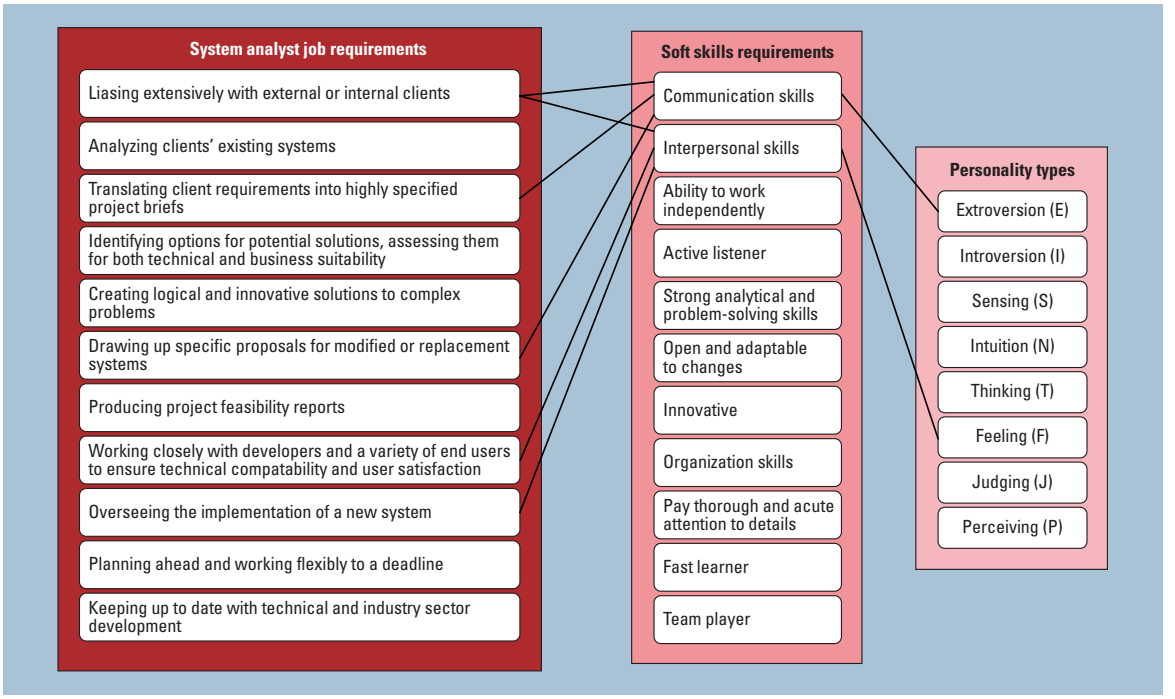


**Figure 1. Mapping system analysts and their skills to personality types. When appointing a system analyst, it's preferable to look for people possessing extrovert (E) and feeling (F) traits.**

the objectives of software construction and operation. The micro-level interpretation demands a set of abilities to carry them out effectively—for example, the skills required to design a software system are quite different from those needed to test it. The hypothesis that not everyone can perform all tasks effectively suggests that personality traits play a critical role; thus, if we can map job and skill requirements with personality characteristics, we could establish a link between software life cycle phases and corresponding personality types.

After analyzing job descriptions for software engineers running in newspapers and magazines, posted on monster.com, and described in various texts,[12] we determined the preferable skills and related them to skills requirements. Subsequently, we mapped the skills rated as desirable and highly desirable for effectively performing the tasks in each phase to MBTI dimensions. Job advertisements generally divide software engineering skill requirements into two categories: hard and soft skills. Hard skills are the technical requirements and knowledge a person should possess to perform a task; they include the theoretical foundations and practical experience a person should have to comfortably execute the planned task.

Although soft skills incorporate the psychological phenomena that include personality traits, social interaction abilities, communication, and personal habits, potential employers tend to imply that soft skills should complement hard skills. Consequently, we related job requirements (or hard skills) to personality requirements (or soft skills) for different positions that reflect the various software life cycle phases, such as system analysts, designers, programmers, testers, and maintainers. Moreover, we also mapped the different soft skills to an individual's personality characteristics by rating them as highly desirable or desirable.

## System Analysis

The system analysis phase emphasizes the identification of high-level components in a real-world application and involves the software system's decomposition into its main modules. In addition to other minor skills, this phase requires that the system analyst determine user needs, consider the system's client requirements, understand the system's essential features, and create an abstract application model that meets these requirements.

System analysis demands a great deal of human interaction with users and clients. To communicate with users, Es are better at talking and getting responses than Is because the latter have a difficult time working with users to accurately represent a problem due to their internal orientation. Thus, it seems reasonable to assume that

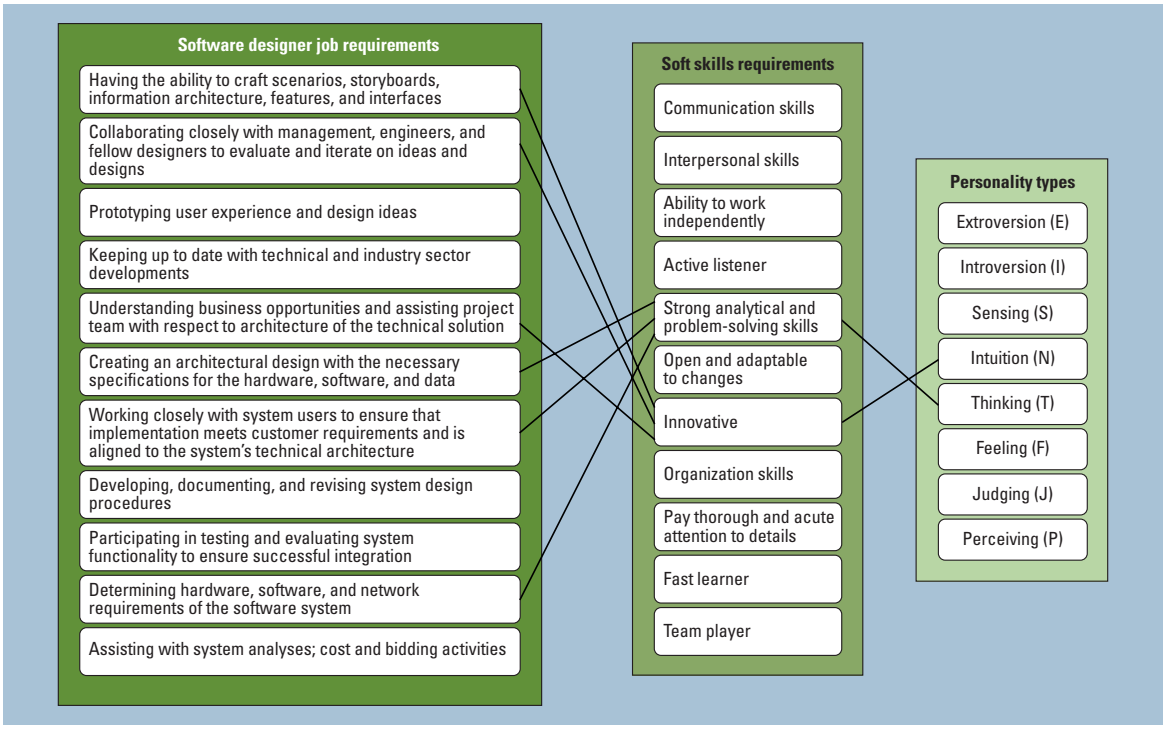


**Figure 2. Mapping software designers and their skills to personality types. A combination of intuition (N) and thinking (T) are paramount to thrive in design.**

extroversion would affect this phase positively. Additionally, system analysts must be able to empathize with users' problems to fully understand their needs, hence interpersonal skills are highly desirable. Recognizing this fact can offer a critical insight to software professionals, who are often viewed as being disconnected from users.

In general, software engineers tend to assume that because they possess more technical expertise than most users, their solutions are more appropriate, but users don't always agree with this assessment. Es and Fs interact with users better than Is and Ts; in particular, Fs excel at making people feel comfortable, whereas Ts aren't attuned to user feelings. Therefore, when appointing system analysts, it's preferable to look for EFs (see Figure 1).

## Software Design

"Design" is an ambiguous word: although there are great variations among design principles, it's possible to find a common set of features that apply to any artifact's design, whether it's a poster, a household appliance, or a housing development. Although software design is still a relatively new field and far from a consensus on its relevant principles, it requires the human creativity evident in other disciplines such as architecture, marketing, and graphic design, rather than the hard-edged formulaic construction of other engineering fields. Software design is an exploratory process: the designer searches for components by trying out a variety of schemes to discover the most natural and reasonable way of refining a solution. Although software design might seem like an easy task, in the design of large and complex software, the identification of key components is an arduous and time-consuming endeavor. Repetitions aren't unusual, since a good design usually takes several iterations. Furthermore, the number of iterations also depends on the designer's insight and experience in the application domain.

Software designers should have the ability to see the big picture. They should be able to isolate relevant items from large quantities of fuzzy and imprecise data, which requires the intuition to discern patterns. Naturally, designers should be intuitive, as those who are imaginative and innovative thrive at designing, especially in comparison to their fact-oriented, black-and-white S counterparts. Software designers perform a wide range of tasks, which include prototyping, elaborating processing functions, and defining inputs and outputs. The first part of the design stage might require skills similar to those needed for analysis, as designing involves team discussions and interaction with the user. As Figure 2 shows,

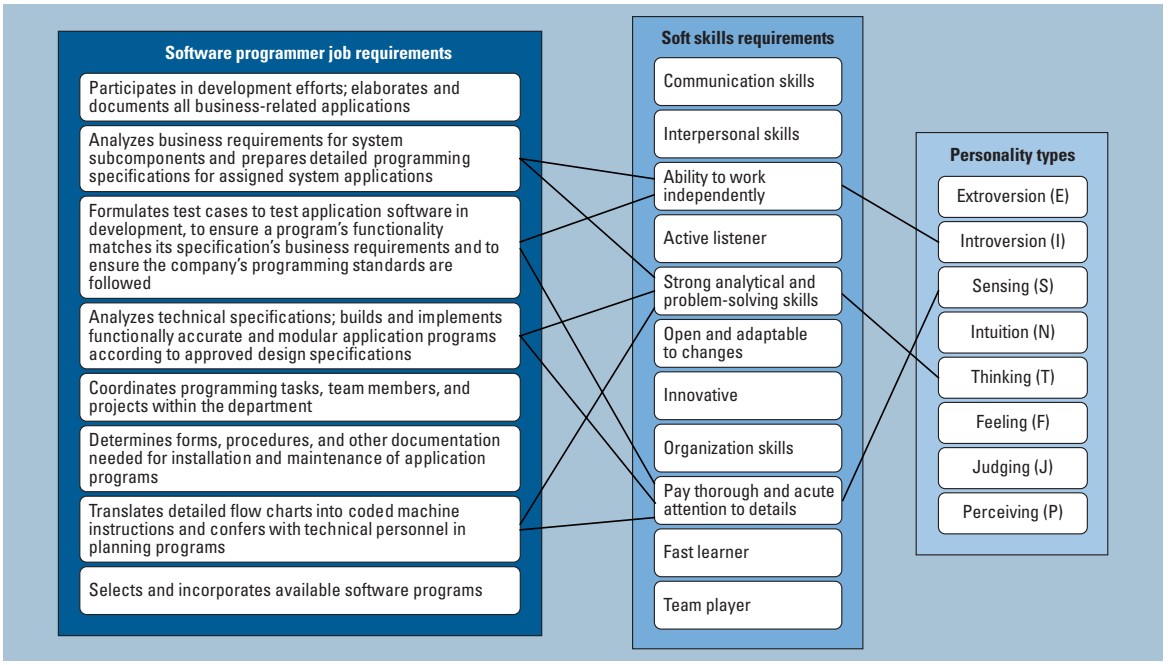


**Figure 3. Mapping programmers and their skills to personality types. Most programmers are introvert (I), sensing (S), thinking (T) types.**

N and T types are highly desirable for software designers, whereas perceiving and feeling are only somewhat desirable. Ps would help reach the best, rather than the first, design solution. Also important is the capacity to predict how users will feel about the design.

## Programming

Programming involves translating a refined version of the design into programs. This phase entails the identification of control structures, relevant variables, and data structures, as well as a detailed understanding of a programming language's syntax and specifics. Programmers must follow an iterative stepwise refinement process that's mostly top-down, breadth first. Thus, programmers should attend to details and keep a logical and analytical thinking style.

The thinking dimension of the MBTI describes the way in which someone makes logical decisions. The problem of interpreting and giving meaning to variables might be a headache, especially for F types rather than for detached analytical, T types, suggesting that the programming stage is more suitable for Ts. Moreover, programming tasks such as determining the details of module logic, establishing file layout, and coding programs demand little interpersonal contact and reveal the programmer's work life as essentially a solitary one.

Programming is an activity that demands logical, impersonal analysis. As Figure 3 shows, programmers working with the specifications from designers need to be logical (Ts), pay attention to details (Ss), and have the capacity to work independently (Is). They might sometimes program in pairs or even within a team, but the core of programming requires the ability to concentrate and work alone for many hours. Given these characteristics, it isn't surprising that so many software engineers are ISTs.

## Testing

Testing involves finding defects in software. The testing stage isn't the first time that defects are found—they can emerge in system analysis and design phases—but testing's main focus is to find as many defects as possible. Several techniques can make testing more effective. First, each module is isolated from other components in the system and tested individually. Such testing, known as unit testing, verifies that a module functions properly with the various input expected (and unexpected!) based on the module's specification. After collections of modules are unit-tested, the next step is to ensure that the interfaces among them are well-defined—this is called integration testing. Finally, system testing is the process of verifying and validating whether the whole software works properly.

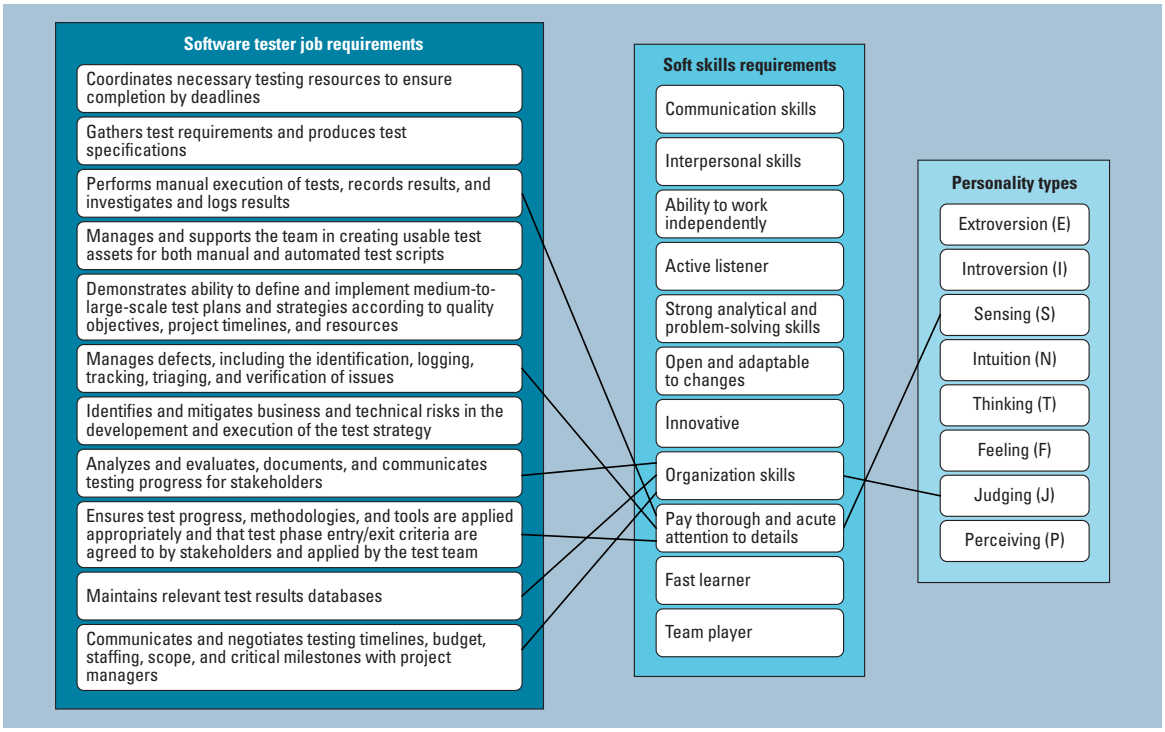


**Figure 4. Mapping testers and their skills to personality types. In theory, sensing (S) and judging (J) people would be more successful in the testing phase.**

Testing strategies are neither random nor haphazard; rather, they should be approached in a methodical and systematic manner. After a defect is detected, debugging can be a frustrating and emotionally challenging activity that can lead software engineers to restructure their thinking and decisions. Testing requires attention to details and is often performed by individuals working independently, and the pressure to meet deadlines and deliver the product is enormous. Thus, precision (S) and order (J) are highly desirable traits. The process of testing demands a great amount of persistence, especially the task of choosing from a wide range of possibilities and keeping an incredible degree of attention to detail. In theory, S and J people would be more successful in the testing phase, as illustrated in Figure 4.

## Maintenance

Software is normally subject to continual change after it's written and while it's operational, thus indicating the necessity of maintaining an evolving system. Projects involving research and state-of-the-art development tend to attract more N people, whereas those having tasks concerned with maintaining and enhancing software systems tend to attract more S types, who tend to be practical, realistic, and observant.

In general, an S person prefers to perform a task in a particular way because it has proven to be successful in the past. Conversely, the N person prefers to perform the task in a totally different way because it has never been done in that manner before. Thus, Ns are likely to be bored with the incremental improvements and small fixes that software maintenance entails because they put more emphasis on new projects. S people prefer jobs that require the use of well-learned knowledge, rather than the development of new solutions; they're also very good observers and tend to focus on details. Maintenance compels a thorough understanding of the software system, especially in terms of how one part can affect the other, thus S people would excel at maintenance because they like to figure out how things work.

Ps like to explore every possibility, and, consequently, they have difficulty making decisions, whereas Js seek closure. Ps should therefore also enjoy maintenance because they're more open to changes and adaptations, and they're more sympathetic to the constant changes requested by users. SPs' problem-solving ability and hands-on approach are an asset for maintenance because such people like to solve practical problems and enjoy the challenge of fixing

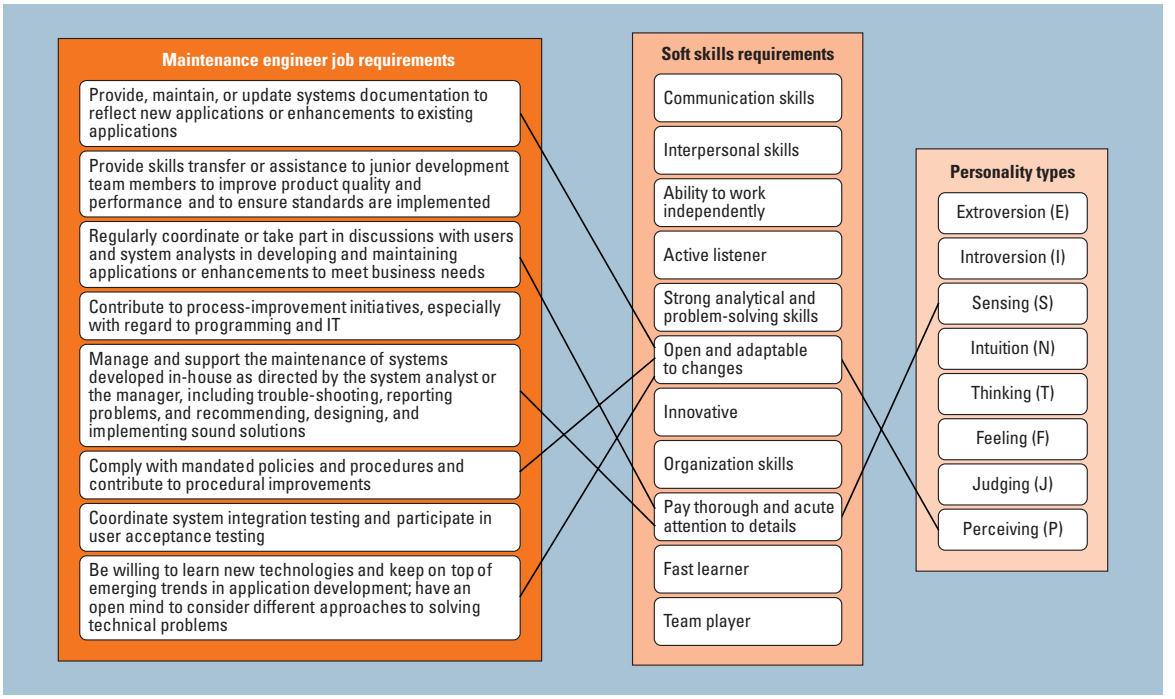


**Figure 5. Mapping maintainers and skills to personality types. Sensing (S) and perceiving (P) types are best suited to the detail-oriented tasks and constant changes inherent to software maintenance.**

programs and systems. Figure 5 displays these relationships, highlighting the qualities of software maintainers.

### Software Life Cycle plus Personality Types

Table 2 shows the five main stages of a software life cycle model and proposes a framework to conceptualize the points at which a particular personality type could have more effect. We assume that system analysis, design, programming, testing, and maintenance are the stages occurring most often in well-accepted software life cycle models, despite some models not considering a few of these stages or including others. Regardless of the model used, a particular personality dimension influences each of the five stages to some extent. The theory behind personality types implies that each one is likely to affect some phases of the software life cycle more than others. Table 2 shows the personality types that appear most relevant to each stage; we've already explained the rationale for each selection.

It's time to recognize that no single personality type fits the wide spectrum of tasks that encompass the engineering of software. The software industry can't afford to lose professionals who might come from a diverse group of people.

A broad range of personality types is beneficial to software engineering.[13] Most software organizations don't have solo performers because better software results from the combined efforts of a variety of mental processes, outlooks, and values. It might be advantageous for software organizations to consider employee strengths when assigning project tasks. More than ever, software engineering needs a diversity of personality types. Putting it in a software context, a diversity of skills and personality traits can solve the myriad problems associated with software development and maintenance. Organizations would benefit from a conscious attempt to diversify the styles or personalities of their software engineers because the strongest teams have the most diverse perspectives. Exposure to software psychology can help this diversity flourish.

## References

1. I.B. Myers et al., *MBTI Manual: A Guide to the Development and Use of the Myers-Briggs Type Indicator*, Consulting Psychologists Press, 1998.
2. S.T. Acuna, N. Juristo, and A.M. Moreno, "Emphasizing Human Capabilities in Software Development," *IEEE Software*, vol. 23, no. 2, 2006, pp. 94–101.
3. R. Feldt et al., "Towards Individualized Software Engineering: Empirical Studies Should Collect Psycho-

**Table 2. The personality types with the strongest impact on the software life cycle.**

| | Software life cycle stages | | | | |
|---|---|---|---|---|---|
| **Personality types** | **System analysis** | **Software design** | **Programming** | **Testing** | **Maintenance** |
| Extroversion (E) | √ | | | | |
| Introversion (I) | | | √ | | |
| Sensing (S) | | | √ | √ | √ |
| Intuition (N) | | √ | | | |
| Thinking (T) | | √ | √ | | |
| Feeling (F) | √ | | | | |
| Judging (J) | | | | √ | |
| Perceiving (P) | | | | | √ |


metrics," *Proc. Workshop Cooperative and Human Aspect of Software Eng.* (CHASE), ACM Press, 2008, pp. 49–52.

4. D.B. Walz and J.L. Wynekoop, "Identifying and Cultivating Exceptional Software Developers," *J. Computer Information Systems*, vol. 37, no. 4, 1997, pp. 82–87.
5. E.A. Turley and J.M. Bieman "Competencies of Exceptional and Non-Exceptional Software Engineers," *J. Systems and Software*, vol. 28, no. 1, 1995, pp. 19–38.
6. N.L. Kerth, J. Coplien, and J. Weinberg, "Call for the Rational Use of Personality Indicators," *Computer*, vol. 31, no. 1, 1998, pp. 146–147.
7. D.J. Pittenger, "The Utility of the Myers-Briggs Type Indicator," *Rev. Educational Research*, vol. 63, no. 4, 1993, pp. 467–488.
8. E. Kaluzniacky, *Managing Psychological Factors in Information Systems Work*, Information Science Publishing, 2004.
9. L.T. Hardiman, "Personality Types and Software Engineers," *Computer*, vol. 30, no. 10, 1997, p. 10.
10. L.F. Capretz, "Personality Types in Software Engineering," *Int'l J. Human-Computer Studies*, vol. 58, no. 2, 2003, pp. 207–214.
11. G.J. Teague, "Personality Type, Career Preference and Implications for Computer Science Recruitment and Teaching," *Proc. 3rd Australian Conf. Computer Science Education*, ACM Press, 1998, pp. 155–163.
12. J. Dolney, "Designing Job Descriptions for Software Development," C. Barry, ed., *Information Systems Development Challenges in Practice, Theory and Education*, Springer, 2009, pp. 447–460.
13. L.F. Capretz, "Implications of MBTI in Software Engineering Education," *ACM SIGCSE Bull.*, vol. 34, no. 4, 2002, pp. 134–137.



***Luiz Fernando Capretz*** *is an associate professor and the director of the software engineering program at the University of Western Ontario, Canada. His research interests include software engineering, human factors in software engineering, software estimation, software product lines, and software engineering education. Capretz has a PhD in computing science from the University of Newcastle upon Tyne. He is a senior member of the IEEE, a distinguished member of the ACM, an MBTI certified practitioner, and a Professional Engineer in Ontario (Canada). Contact him at lcapretz@eng.uwo.ca.*

***Faheem Ahmed*** *is an assistant professor at the College of Information Technology, United Arab Emirates University. His research interests are software product lines, software process modeling, software process assessment, and empirical software engineering. Ahmed has a PhD in electrical engineering from the University of Western Ontario. He is a member of the IEEE. Contact him at f.ahmed@uaeu.ac.ae.*