Why Do We Need Personality Diversity in Software Engineering?

Luiz Fernando Capretz
University of Western Ontario
Dept. Elect. & Computer Engineering
London, ON, Canada N6A5B9
lcapretz@eng.uwo.ca

Faheem Ahmed
United Arab Emirates University
College of Information Technology
Al Ain, United Arab Emirates
f.ahmed@uaeu.ac.ae

Abstract

Diversity of skills is good for society, it is also good in problem solving because different people see a problem from several perspectives, so diversity should be good for software engineering too. This study tackles a difficult to study aspect of software engineering, that is, how to best associate personnel with the various tasks in a software project. The approach uses psychological types to determine who is best suited to particular development roles. The article has four main objectives: (1) to arouse awareness of human factors among software engineers; (2) to investigate how psychological factors can contribute to their effectiveness at work; (3) to catalyze effort among software engineers leading towards a deeper understanding and broader applications of human factors in the light of the activities involving the engineering of software; and (4) to emphasize the important of skill diversity in the software engineering field. This article provides conceptual knowledge, reports findings, and presents both real and hypothesized beliefs from the software engineering community. Likewise, it is hoped that the article will motivate software engineers and psychologists to conduct more research in the area of software psychology, so as to understand more profoundly the possibilities for increased effectiveness and fulfilment among software engineers.

Keywords: Software Psychology, Personality Types, Diversity in Software Engineering, Human Factors in Software Engineering, Software Skills, MBTI

1. Introduction

Software engineering has been one of the most impactful professions of the last 30 years and will continue to expand its boundaries into the next decades. Software development has been considered a socio-technical endeavour for some time. For software engineers the need to communicate effectively with users and team members has been increasingly emphasized. When someone accepts the vision that awareness of psychological dimensions within oneself and human factors within one's work environment increase the software engineer's productivity, we must then wonder which psychological traits would be most worth investigating.

Although in the 70s and 80s software developers (systems analysts and programmers) had the lowest needs for social interaction on their jobs; nowadays human resources people responsible for hiring software engineers indicate that in addition to knowledge in applied computing and business, it is very important that software professionals have the capacity to learn, ability to work in teams, oral and writing skills, and orientation to health and wellness. In short, adaptability, communication, and stress management are seen as key skills for the software professional. Yet, such skills are not developed through logic and algebraic reasoning alone, but involve "soft areas", feelings, and senses.

Thus the time appears to be set to address specific psychological factors as applied to the work of various software engineers. The reason for addressing these human factors is largely the recognition that software engineers could benefit from greater awareness of themselves and others in order to develop their "soft skills", which in turn can positively influence their work. Software engineers have long realized that "soft skills" are increasingly required in the course of their work. However, specific studies have been sporadic and often incidental.

Software production is a result of human activities, which often include problem-solving capabilities, cognitive aspects, and social interaction. However, human beings are more complicated and less predictable than computers. Therefore, the complexity of human personality entails intricate dynamics that are integral and yet often overlooked in software development. Thus, sooner or later, major issues relevant to software engineering boil down to people and their personality traits.

Software is a word with multiple connotations. Individuals discussing software may refer to the structure of a program, the functionality of an application system, the appearance of an interface, or even the overall user experience with a hardware-software environment. Software engineering spans both new software developments and the maintenance of legacy systems, with each software life cycle phase bringing its own context of understanding what matters, what can be created, and what tools and methods are appropriate. Because of its multifaceted work, software development is among the most challenging jobs performed by mankind.

Over the last three decades, the engineering of software has become a very broad field, consequently the skills necessary to successfully work in this area thirty years ago may no longer apply. For instance, software design has became much more than manipulating formal or semi-formal notations; rather, it 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/she really needs. Today, successful software is developed after a tremendous amount of time has been spent with the user in the form of prototyping, experimentation, and feedback. In fact, these three activities represent the *de facto* life cycle of many useful software systems.

As companies rely more heavily on project teams and encourage software engineers to collaborate with their customers, the opportunities for conflicts abound. When it comes to personality types, opposites do not always attract each other [1]. When software professionals discuss how a task needs to be accomplished, they tend to be poor at verbalizing how the task affects the involved individuals. The greatest difference between software engineers and the general population is the percentage who takes action based on

DOI: 10.1145/1734103.1734111 http://doi.acm.org/10.1145/10.1145/1734103.1734111

what they think rather than what they feel. Unfortunately, this mentality does not help bring software engineers closer to their users. Reality shows that software developers require more than computer literacy; they also need emotional literacy and mutual understanding to build teams and to collaborate effectively with users. Specifically, there should be a clear association between activities or roles, software skills, and personality types.

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

A major rationale behind this paper is to discern connections between human factors, particularly psychological types, and the process of developing software. Accordingly, the work focuses on the complex relationship between people's skills and software development. This interdisciplinary, human-centred research makes use of theories about psychological types, human personality factors, and software engineering. It contributes towards a bridge that links software engineering and software psychology, and it attempts to shed light on several outstanding problems that plague the software industry

1.1 Introduction to the Myers-Briggs Type Indicator (MBTI)

First of all, it is essential to introduce the four scales of the MBTI [2]. The MBTI is an instrument for measuring and understanding individual personality types. Specifically, it can reveal how people prefer to receive information, how they form opinions, and how they organize their lives. The MBTI currently ranks among the most popular indicators used in the workplace [3, 4], and it establishes four parameters for assessing personality types. Although all individuals possess the personality qualities contained within each scale or parameter, each individual naturally prefers some qualities or is more comfortable with some traits than others. For the MBTI, each scale is bimodal, with its central point having a zero value. Each respondent is required to choose between preferences; the higher the score on each preference, the stronger that preference is likely to be.

Extroversion (E) versus Introversion (I)

The first scale represents complementary attitudes towards the external world. While the extrovert prefers looking outward, the introvert has an inward view. For example, strong extroverts are sometimes said to "talk to think" whereas introverts "think to talk". Contrary to popular belief, the implications of these terms go beyond the common stereotypes of sociable versus shy. Extroverts are talkative, initiators of conversation, and outgoing. They prefer action and variety. Introverts, in contrast, are quiet, reserved, and respond to conversation rather than initiate it. They prefer silence and time to consider matters. Overall, extroversion implies a tendency to be talkative, lively, and expressive; introversion describes a person that is quiet, introspective, and reserved.

Sensing (S) versus Intuition (N)

The second scale distinguishes the way that individuals assimilate information from the environment. While a sensing individual might need to absorb a whole series of facts in linear fashion, an intuitive person can take in the same information through abstraction and concepts that, while initially appearing to be unrelated, can establish meaning beyond the information captured only by the senses. Sensing individuals dislike new problems unless prior experience shows how to solve them; on the other hand, intuitive people enjoy solving new problems and dislikes performing repetitive tasks. The adjectives that describe a sensing person are realistic, practical and fact-oriented; while those that qualify as an intuitive individual are speculative, imaginative and principle-oriented. Of course, we all share both sets of qualities to some degree, but one set usually predominates.

Thinking (T) versus Feeling (F)

The third mode of orientation in the MBTI classification involves the dichotomy of thinking and feeling. Again, these terms are more comprehensive than everyday usage would indicate. In particular, these terms refer to the process of decision-making. This scale of preferences identifies thinking as the logical way of making a decision, while feeling describes the tendency to rely on values as a basis for making decisions. Thinking people are principle-oriented, cool-headed and firm, whereas feeling people are emotion-oriented, warm-hearted and have strong interpersonal skills.

Judging (J) versus Perceiving (P)

The fourth scale differentiates between the way in which individuals orient their lifestyles and organize their worlds. Judging identifies the tendency to be extremely organized. If a deadline is to be met, a judging person usually finishes the task well in advance. At the other extreme, a perceiving individual prefers procrastinating, 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. Perceiving people like to delay decisions. Often, the easiest way to distinguish between these two types of individuals is to look at the person's desk. The desk of a judging person is immaculately organized, whereas the desk of a perceiving person appears to be in constant chaos even though the perceiving individual claims to know exactly where everything is located and states that there are rules underlying the apparent chaos. The adherence to deadlines, punctuality, closure and routine describe judging personalities, while the terms open-ended, tentative, adaptable and spontaneous apply to perceiving types.

Summarizing, the MBTI sorts these four sets of preferences, selecting one from each pair, to delineate a person's preferred type. Hence, there are 16 possible configurations, presented in Table 1, in addition to the percentages of the various types among a representative sample of the U.S. adult population; however, these percentages may vary from one country to another. If the MBTI results reveal that a person is ISTP, the terminology suggests that the person *prefers* ISTP, rather than being an ISTP. Thus there are no rights or wrongs in the personality types, there are merely preferences.

DOI: 10.1145/1734103.1734111 http://doi.acm.org/10.1145/10.1145/1734103.1734111

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%

Table 1. The 16 MBTI types and their distribution among the USA adult population [2]

2. Related Work

Human factors in software engineering have different dimensions. Studies have been performed from different perspectives. These perspectives could be the investigation of human factors in different phases of software life cycle, or the effect of team work in software development, or how can a personality profile suit a particular task like code review, or about some other miscellaneous issues. A few studies have investigated the relationship between human skills and the software life cycle phases.

Karn and Cowling [5] investigate the effects of different personality types using MBTI on the working of some software engineering team. The study describes how ethnographic methods could be used to study software teams, and to understand the role of human factors in a software project. The results of the study indicated that certain personality types were more inclined to certain roles.

Using the 16PF test [6], Acuna *et al.* [7] measured the correspondence between individual capabilities, such as intrapersonal, organizational, interpersonal, management, and software roles, including team leader, quality manager, requirements engineer, designer, programmer, maintainer, tester, and configuration manager. Feldt *et al*. [8] evaluated the personality of 47 software professionals using the IPIP 50-item five-factor personality test [9]. After extensive statistical analyses, they found that there are multiple and significant correlations between personality factors and software engineering, and they concluded that individual differences in personality can explain and predict how judgments are made and how decisions are evaluated in software development projects.

Hannay *et al.* [10] report the impact of the Big Five [11] personality traits on the performance of pair programmers together with the impact of expertise and task complexity. The study involved 196 software professionals in three countries forming 98 pairs. The results show that personality may be a valid predictor for long-term team performance; however, they found that personality traits, in general, have modest predictive value on pair programming performance compared with expertise, task complexity, and country. They recommend that more effort should be spent on investigating other performance-related predictors such as expertise, and task complexity, as well as other promising predictors, such as programming and learning skills. They also suggest that effort should be spent on elaborating the effects of personality on various measures of collaboration, which, in turn, may be used to predict and influence performance. Insights into malleable factor such as learning, motivation, and programming skills rather than static, factors may then be used to improve pair programming performance.

Within the field of software engineering, there are tremendous differences among individual performance in programming. Instructors of programming courses witness first hand the huge variety among students in learning achievement and programming assignments. Furthermore, Shneiderman [12] reports that some programmers perform as much as ten times better than other programmers with similar backgrounds. Walz and Wynekoop [13] derive a methodology for identifying the traits and characteristics of top performing software developers: (1) those who are best at making things work; (2) those who can best communicate with end-users, identify requirements, and transform them into a logical design; and (3) those destined for management. Turley and Bieman [14] also seek to identify the attributes that differentiate exceptional and non-exceptional software engineers and map them to the MBTI scale; these differences may result from the fact that productive people are carrying out the tasks they prefer.

Cognitive styles have been examined as factors that may help to explain some of the variability; however, they have failed to consistently explain individual preference towards computer programming as opposed to another task, such as system analysis or design. Accordingly, MBTI offers the potential to provide a suitable model for comparison. Indeed, effective software development demands a broad set of skills. If a project lacks people with certain preferences, some individuals may have to perform tasks to which they are not naturally suited or may not find enjoyable.

Kerth *et al.* [15] are sceptical about the ability of MBTI to predict who is going to make a good software engineer; this is because MBTI does not consider variables such as passion, experience, and financial rewards. They are correct about the inability of a single personality test to predict success in a field as broad as software engineering, where different skills are required for system analysis, design, testing, maintenance and technical support. Nevertheless, 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”.

This debate is long overdue. Psychological assessment instruments have been used for over sixty years and have reached a mature stage for predicting career selection and behaviour. Many of these instruments are based on the theories of C. Jung and S. Freud. In particular, MBTI has been one of the most popular tools used for ascertaining personality types, especially because the instrument has been supported by extensive data. For instance, the profiles of which personality types are attracted to which specific occupations are derived from more than two million indicators administered annually all over the world. Although a few times MBTI measures have been questioned [16], the instrument does not predict success in a career; it does, however, identify preferences for occupations.

There is clear evidence that personality preferences have great impact on motivation, quality of work, and retention in the field of software engineering [17]. Hardiman [18] has claimed that the MBTI may be the best predictor of who will become a competent programmer. He observed that the majority of good programmers were ISTJ, INTJ, ESTJ, ENTJ, ISFJ, or ENTP; in brief, they are mostly NTs and SJs. He also implies that NF types tend to have trouble with the sequential and abstract thinking necessary for writing programs. Capretz [19] has investigated the profile of a group

DOI: 10.1145/1734103.1734111 http://doi.acm.org/10.1145/10.1145/1734103.1734111

of 100 software engineers (80% male and 20% female) who study in private or public universities, work for the government or are employed by software companies. All these individuals are productive and motivated software engineers and were selected to participate in this study based on their occupation. This study has shown that ISTJ, ISTP, ESTP and ESTJ orientations compose over 50% of the sample and are therefore significantly over-represented, whereas the INFJ, ESFP and ENFJ groups are all particularly under-represented.

Bishop-Clark [20] investigated the relationship between cognitive aspects, personality traits and computer programming. She divides programming into several stages: problem representation, program design, implementation, and debugging. Moreover, she organizes the theories and empirical studies of computer programming into four sub-tasks: problem solving, designing, coding, and debugging. The cognitive styles discussed entail dichotomies such as field dependency/independency, analytic/holistic, impulsivity/reflectivity, and divergent thinking; the personality traits include locus of control, and introversion/extroversion. These variables were mentioned because, according to her theory, they were all important within the realm of computer programming. In general, her model suggests that different characteristics are necessary for different tasks; for example, she indicates that the attribute necessary for debugging is reflectivity rather than impulsivity.

In contemporary society, the software industry has become a major employer. It has, in fact, generated many commentaries on the unique contributions of professionals engaged in its many subareas. Specialties within software engineering are as diverse as those in any other profession. Software engineering comprises stages in separate and distinct phases including system analysis, design, programming, testing, and maintenance. It may be that certain personality dimensions affect one phase but not others, or affect specific phases in different fashions.

Teague [21] tried to map the MBTI dimensions into three major subtasks of computing: system analysis, system design, and programming. A study of 38 computing professionals confirmed that computer specialists are not a homogeneous group. The personality types preferred for system analysts included those with a combination of extroversion (E) and intuition (N), as nine of the thirteen participants preferring analysis had these two characteristics. Furthermore, ten of the thirteen participants favouring system design fell within the most preferred range of attributes, with six having the intuition (N) and thinking (T) characteristics considered useful for higher level design, two having the sensing (S) and Judging (J) characteristics desired for dealing with the detailed aspects of design, and two being ISTP – practical problem solvers. Finally, seven out of the ten respondents preferring programming also corresponded to personality types of traditional programmers, such as ISTJ, who are sensors (S) with attention to detail; the other three were classified as intuitives (N) and thinking (T).

More recently, Capretz and Faheem [22] have mapped some opposing psychological traits, such as extroversion-introversion, sensing-intuition, thinking-feeling, and judging-perceiving, to the main stages of a software development life cycle. Subsequently, they have argued that assigning a person with specific psychological characteristics to the stage of the software life cycle best suited for his or her traits increases the chances of a successful outcome for the project.

Software is developed by people, used by people, and supports people's work. As such human characteristics, behaviour, and cooperation are central to practical software development [23]. Specifically, the personality of software engineers has become increasingly important in recent years. First, software engineers nowadays are expected to have a broader range of skills than in the past. Secondly, many users are dissatisfied with the personal rather than the technical services they receive from software engineers. Several studies involving the personality traits of software engineers have been reported. However, much of the research has sought to classify software engineers into personality profiles according to particular psychological attributes measured by a personality instrument. A common thread running through the results of these and other similar studies is the prevalence of introverts (I), thinking (T), judging (J), and almost as many sensing (S) as intuitive (N) types among software professionals. While these empirical studies suggest that the MBTI poles are related to software engineering, they do not specify at which phase of the software life cycle they occur or how they are related.

Despite early interests in the importance of human factors in the engineering of software [24], particularly the personal characteristics of people involved in the software engineering processes, such factors have been largely overlooked. This oversight, in turn, has prevented the acquisition of detailed knowledge about how different aspects of personality correlate with software life cycle phases and hindered the use of this knowledge to improve software engineering processes. A more focused approach may help identify at which software life cycle phase a particular personality type has the most significant impact. This fundamental issue is further examined in the next section.

3. Mapping Job Requirements and Skills to Personality Characteristics

Software engineering has been roughly characterized as set of activities comprising system analysis, design, programming, testing, and maintenance. Logically, they are different tasks which are put together to achieve the objective of software construction and operation. The micro-level interpretation of these activities 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 the software. The psychological hypothesis that not every one can perform all tasks effectively suggests that personality traits play a critical role in the performance of people executing the same task. Hence if we map the job and skill requirements with personality characteristics, we would likely be able to establish a link between the software life cycle phases and corresponding personality traits.

After analyzing various job descriptions for software engineers appearing in newspapers and magazines, posted at the website monster.com, and described in [25], we determined the preferable skills and related them to personality characteristics. Subsequently, the skills desirable and highly desirable for effectively performing the tasks in each phase of the software life cycle were mapped to the MBTI dimensions. The job advertisements in the area of software engineering generally divide the skill requirements into two categories: "hard skills" and "soft skills". Hard skills are the tech-

DOI: 10.1145/1734103.1734111 http://doi.acm.org/10.1145/10.1145/1734103.1734111

nical requirements and knowledge a person should possess to carry out a task; these skills include the theoretical foundations and practical experience that a person should have to comfortably execute the planned task.

Even though soft skills incorporate the psychological phenomena that include the personality types, social interaction abilities, communication, and personal habits, it is apparent that people imply that soft skills should complement the hard skills. Consequently, we related the job requirements or "hard skills" to personality requirements or "soft skills" for different positions such as system analysts, designers, programmers, testers, and maintainers, which reflect the various software life cycle phases. Moreover, we also mapped the different "soft skills" to the personality characteristics of an individual by rating them as "highly desirable" or "desirable". Although in Figures 1-5 most skill requirements are "desirable" and are connected to the personality characteristics, only the "highly desirable" skill requirements for a particular phase of the software life cycle are considered.

3.1 System Analysis

The system analysis phase emphasizes the identification of high-level components in a real-world application and involves the decomposition of the software system into its main modules. In addition to other minor skills, the system analysis phase requires that the system analyst determine the users' needs, consider the clients' requirements of the software system, understand the system's essential features, and create an abstract model of the application in which these requirements are met. Overall, the main product of the system analysis phase is a graphical and/or textual description, either informal or formal, of an abstract model of the application.

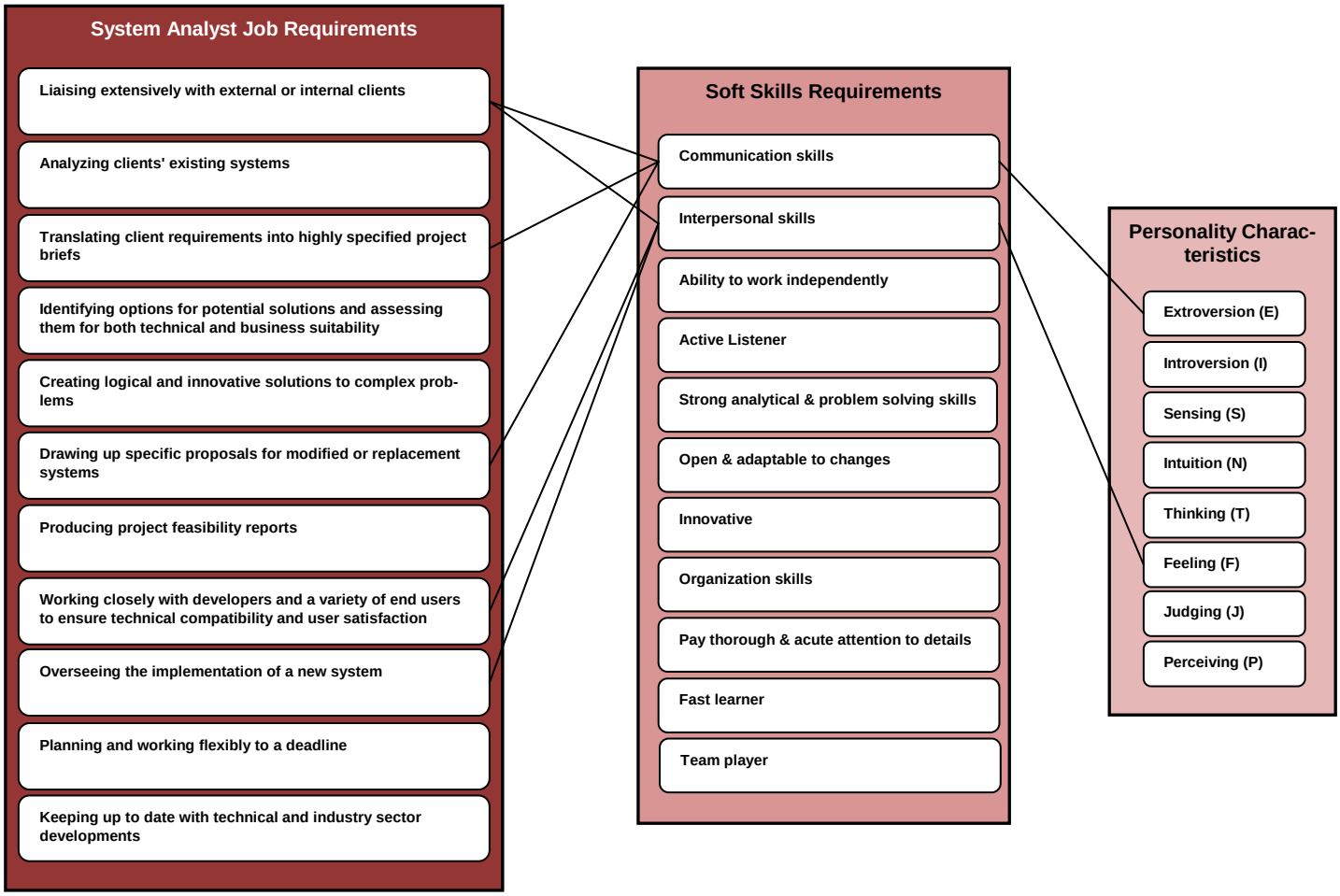

Figure 1. Mapping system analysts and their skills to personality characteristics [22]

System analysis demands a great deal of human interaction with users and clients. To communicate with users and management, extroverts are better at talking and getting responses than introverts, since introverts have a difficult time achieving a problem representation with users due to their internal orientation. Thus, it seems reasonable to assume that extroversion would affect this phase in a positive manner. Additionally, system analysts must be able to empathize with the users' problems in order to fully understand their needs, hence interpersonal skills are highly desirable. Recognizing this fact can offer a critical insight for software professionals, who are often viewed as being disconnected from the users.

In general, there is a tendency for software engineers to assume that because they possess more technical expertise than most users, their solutions are more appropriate, but the users do not always agree with this assessment. Extroverts (Es) and feelers (Fs) interact with users better than introverts (Is) and thinkers (Ts); in particular, feelers (Fs) excel at making people feel comfortable, whereas

DOI: 10.1145/1734103.1734111 http://doi.acm.org/10.1145/10.1145/1734103.1734111

thinkers (Ts) are not attuned to the user's feelings. Therefore, when appointing system analysts, it is preferable to look for EFs, which are indicated as highly desirable characteristics for system analysts in Figure 1.

3.2 Software Design

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 require 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 sensing 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 may require characteristics similar to those needed for analysis, as designing involves team discussions and interaction with the user. As depicted in Figure 2, intuition and thinking characteristics are highly desirable for software designers, whereas feeling is only somewhat desirable. The ability to be intuitive (N) is paramount, also important is the capacity to predict how the users will feel about the design. Furthermore, a combination of judgers (J) and perceivers (P) would ensure that the best, rather than the first, design solution is found.

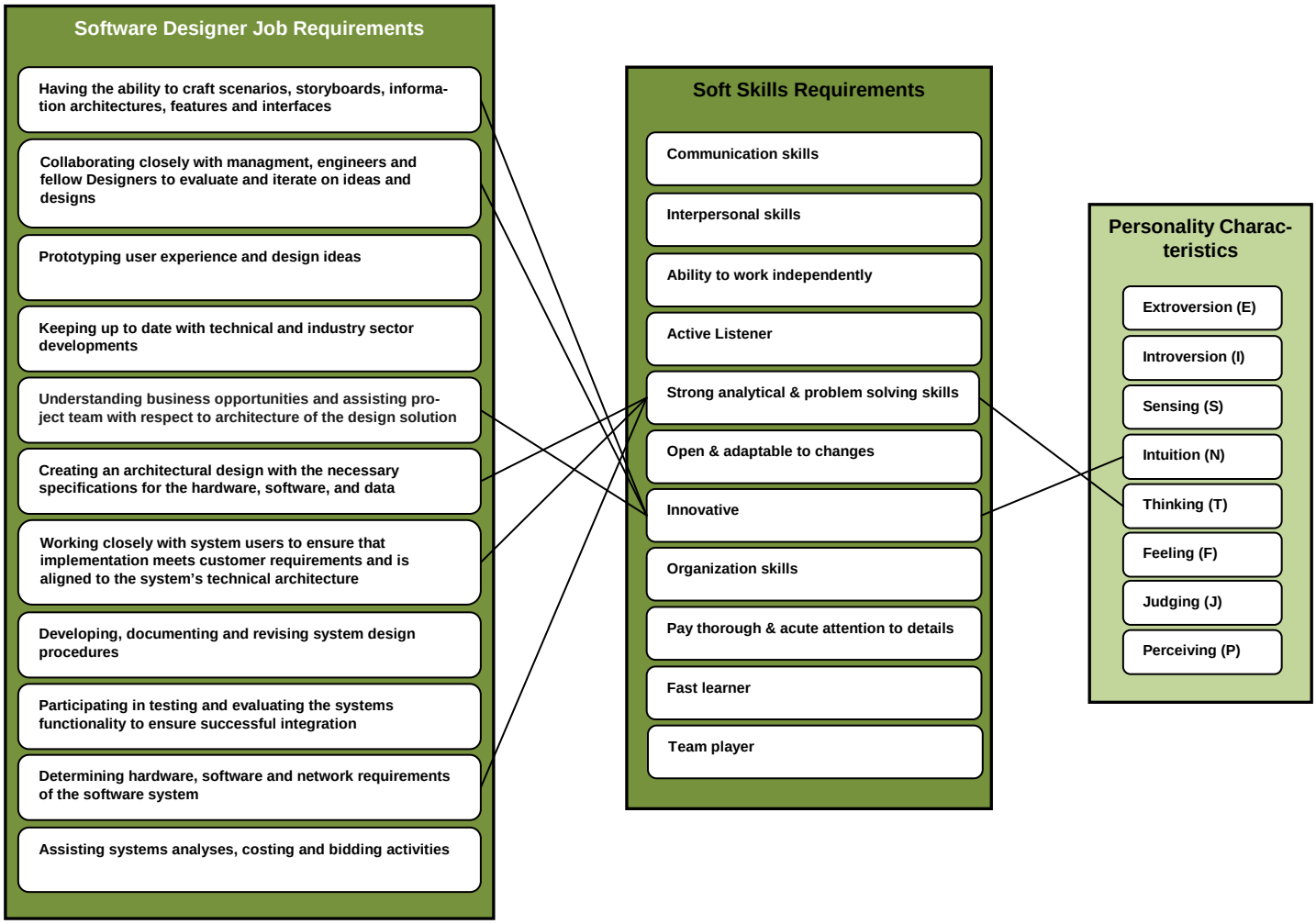

Figure 2. Mapping software designers and their skills to personality characteristics [22]

3.3 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 the syntax and specifics of a programming language. Programmers need to follow an iterative stepwise refinement process that is 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 may be a headache especially for feeling types rather than for detached analytical, thinking types, suggesting that the programming stage is more suitable for thinkers (T). 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.

DOI: 10.1145/1734103.1734111 http://doi.acm.org/10.1145/10.1145/1734103.1734111

Programming is an activity that demands logical, impersonal analysis. As shown in Figure 3, programmers working with the specifications from designers need to be logical (Thinkers), pay attention to details (Sensing), and have the capacity to work independently (Introverts). They may 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 is not surprising that so many software engineers are ISTs.

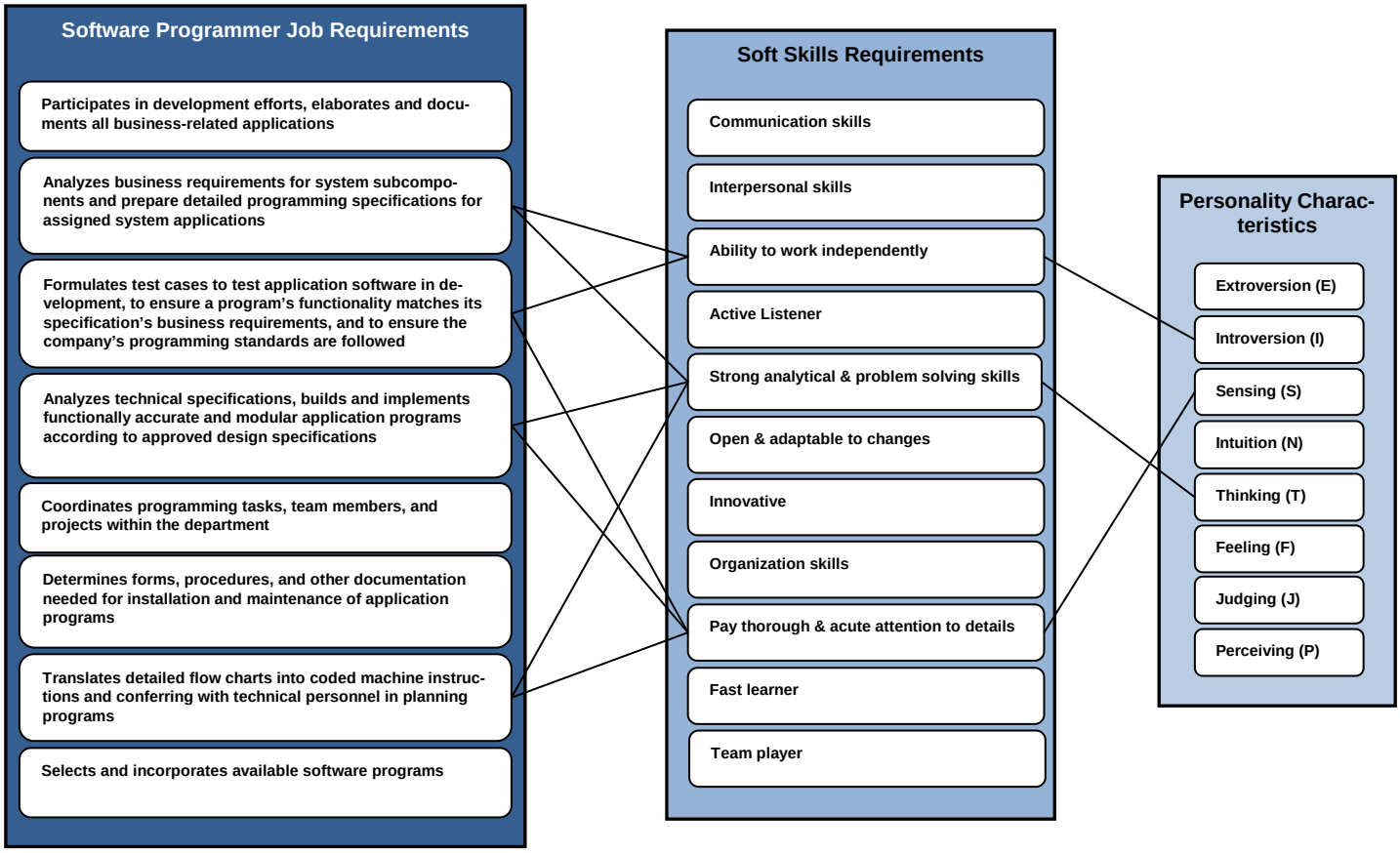

Figure 3. Mapping programmers and their skills to personality characteristics [22]

3.4 Testing

Testing involves finding defects in software. The testing stage is not the first time that defects are found; they can emerge in system analysis and design phases. But the main focus of testing is to find as many defects as possible, and there are several techniques to make testing more effective.

First, each module is isolated from the 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 have been unit-tested, the next step is to ensure 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.

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 may 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 (Sensing) and order (Judging) characteristics are highly desirable. 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, sensing (S) and judging (J) people would be more successful in the testing phase, as illustrated in Figure 4.

DOI: 10.1145/1734103.1734111 http://doi.acm.org/10.1145/10.1145/1734103.1734111

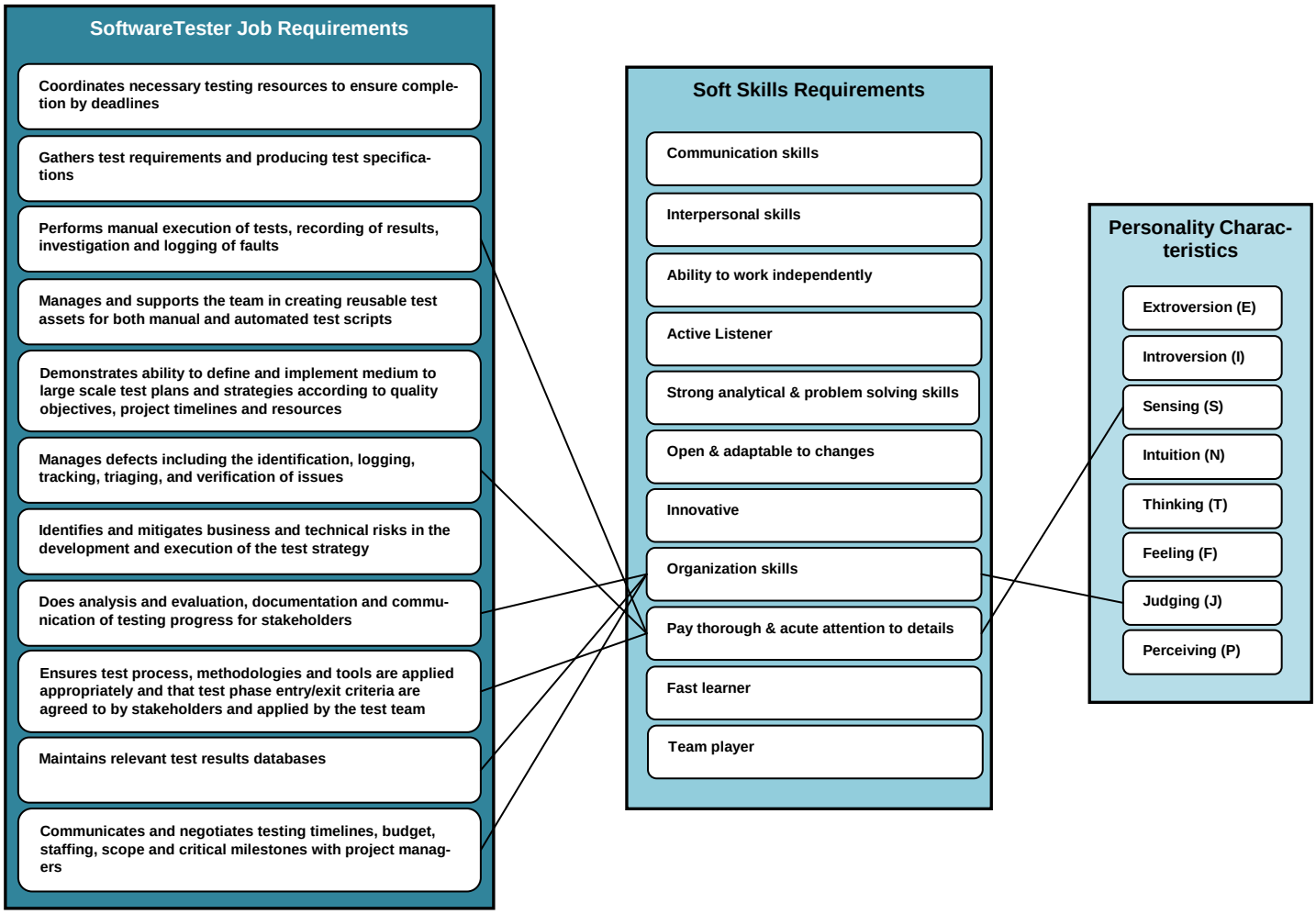

Figure 4. Mapping testers and their skills to personality characteristics [22]

3.5 Maintenance

Software is normally subject to continual change after it is written and while it is operational, thus indicating the necessity of maintaining an evolving system. It has been observed that projects involving research and state-of-the-art development tend to attract more intuitive people, whereas those having tasks concerned with maintaining and enhancing software systems tend to attract more sensing types, who tend to be practical, realistic, and observant.

In general, a sensing (S) person prefers to perform a task in a particular way because it has proven to be successful in the past. Conversely, the intuitive (N) person prefers to perform the task in totally different way because it has never been done that manner before. Thus, intuitives (Ns) are likely to be bored with the incremental improvements and small fixes that software maintenance entails, they put more emphasis on new projects. On the other hand, sensing (S) people enjoy jobs that require the use of well-learned knowledge, rather than the development of new solutions; also they are very good observers and focus on details. Intuitives (Ns) are creative and enjoy abstract symbolic relations, which involves finding patterns rather than dealing with details, they like to create new knowledge rather than applying existing techniques. Maintenance compels a thorough understanding of the software system, especially in terms of how one part can affect the other, and sensing (S) people would excel at maintenance because they like to figure out how things work.

Perceivers (Ps) like to explore every possibility, and consequently, they have difficulty making decisions, whereas judgers (Js) seek closure. Perceivers should also enjoy maintenance because they are more open to changes and adaptations, and they would be more sympathetic with the constant changes requested by users. The problem-solving ability and hands-on approach of SPs is an asset for maintenance because they like to solve practical problems and would enjoy the challenge of fixing programs and systems. Figure 5 displays these relationships, highlighting the qualities of software maintainers.

DOI: 10.1145/1734103.1734111 http://doi.acm.org/10.1145/10.1145/1734103.1734111

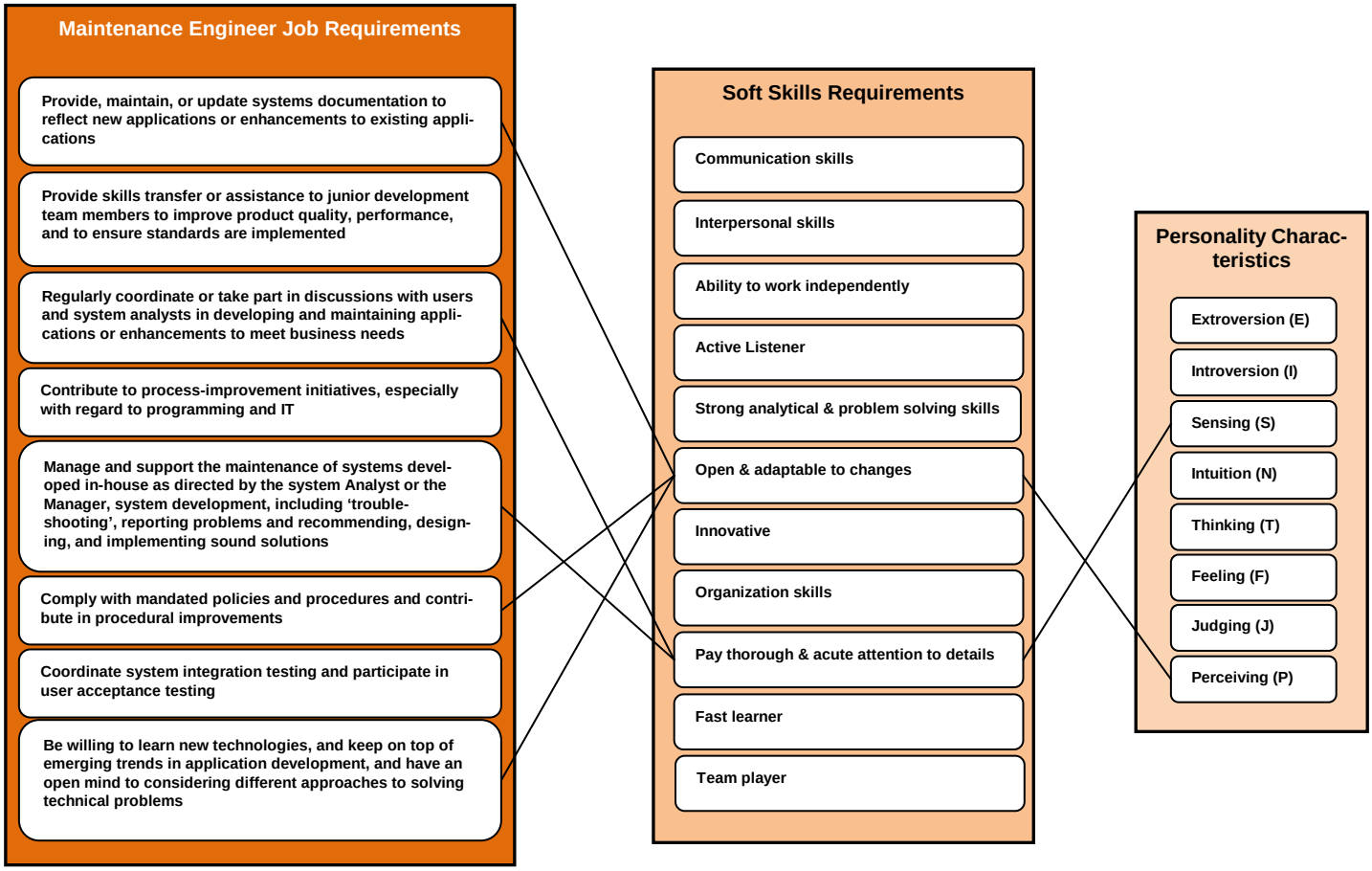

Figure 5. Mapping maintainers and their skills to personality characteristics [22]

Figure 6 shows the five main stages of a software life cycle model and depicts a framework to conceptualize the points at which a particular personality trait could have more effect. We assume that system analysis, design, programming, testing, and maintenance are the stages often occurring in well-accepted software life cycle models, despite some models not consider a few of these stages or include other stages. 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 personality type is likely to affect some phases of the software life cycle more than others.

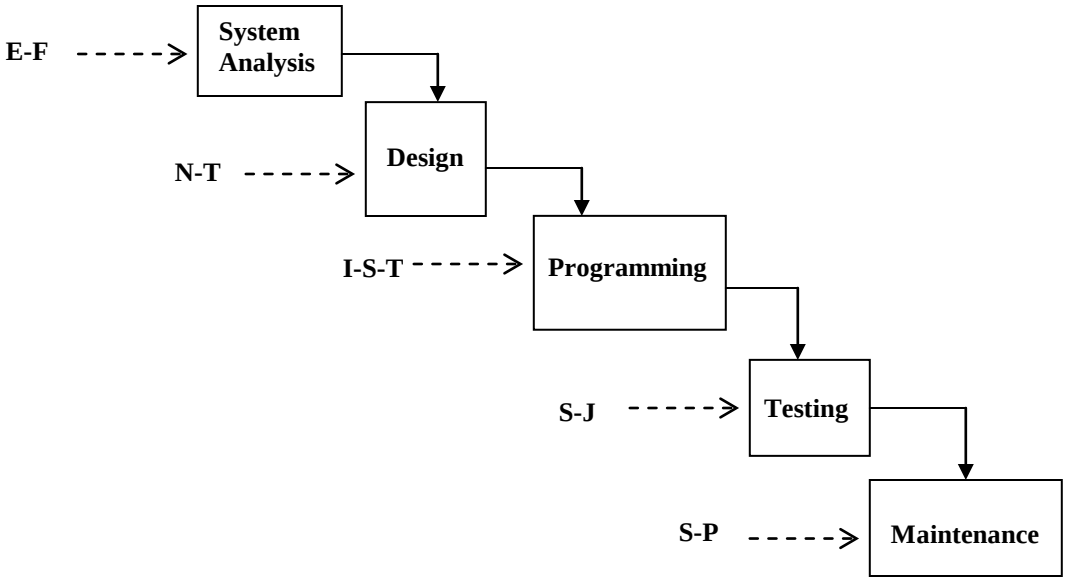

Figure 6. MBTI dimensions and the phases of a software life cycle model

DOI: 10.1145/1734103.1734111 http://doi.acm.org/10.1145/10.1145/1734103.1734111

The personality traits that appear most relevant to each stage are portrayed in Figure 6 and the rationale for each selection was previously explained. These results are based on the MBTI theory and the general empirical data collected for this work. Specifically, the figure has been developed by relating the aspects of the personality theory to the tasks defining each stages of the software life cycle model. Overall, the table describes the potential impact of personality types on the software life cycle.

4. Final Remarks

It is common sense to state that the production of any software involves a human element, at least to some extent. People have different personality traits, and the way they perceive, plan, and execute any activity is influenced by these traits. Most of the time, software development is a product of teamwork involving several people performing various tasks. The success and failure stories of software projects reveal the human factor as one of vital importance. According to psychology, not everybody can excel at all kinds of tasks. Thus better results are achieved if people with particular personality traits are assigned to different aspects of a project, especially the roles best suited to their ability.

In software engineering, human factors are usually overlooked because the relationship between software production and personality types is extremely complex and challenging to investigate. Nevertheless, it has been worthwhile studying a possible relationship between the engineering of software and personality types. This research addresses questions of great importance for software engineers, including the profiles of professional community members, as well as the relationship between software development skills and personality types. It does not offer generic, one-size-fits-all advice though. The MBTI indicates preferences only. Most careers have more of the types that in theory should find that career attractive. However, all types can be found in all careers. Employment and life decisions should not be based on the results of any single assessment, such as the MBTI instrument.

Due to the diverse nature of software engineering, it is time to recognize that there is no single personality type that fits the wide spectrum of tasks that encompass the engineering of software. Although a predominant type may be found within the software engineering community, it is important to remember that all types are likely to be represented within that group. For example, 80% of software engineers have preference for thinking type. People of this type may be attracted to this occupation because it requires logical thinking. However, each of the other dimensions can also participating in the engineering of software and may be just as successful and satisfied in this career field as the more common thinkers.

Under-represented types are likely attracted to the career for other reasons and may contribute in unique ways. Most people are able to find or create a rewarding niche for themselves in a profession with the predominance of particular types, by aligning their personality with their job duties and thus achieving satisfaction in a job where they can respect their preferences and feel comfortable with themselves and their roles. Indeed, a broad range of personality characteristics is beneficial to software engineering. It may be advantageous for software organizations to consider the strengths of their employees when assigning tasks in a project.

Our study only considers the traditional stages of system analysis, software design, programming, testing, and maintenance, so it omits the different characteristics that may be more appropriate for other software occupations, such as project manager, trouble shooter, helpdesk personnel, database administrator, and so forth. Moreover, certain characteristics may be less desirable now than they were in the past. With the diminishing demand for the traditional lonely programmers and the increasing need for people who can communicate well at all levels of an organization, the software industry requires a much lower proportion of introverts and thinkers than it needed in the past. Conversely, the software industry will employ a higher proportion of most of the under-represented personality types. Thus, the software industry cannot afford to lose potential professionals who may come from a diverse group of people [26].

Nowadays there are very few solo performers in most software organizations; people have to work collectively in teams of some sort, consequently, there should be a certain amount of diversity on the teams in terms of psychological type. In this case, better software will result from the combined efforts of a variety of mental processes, experience, and values. Therefore all types are important to software engineering, as every type can make a contribution towards solving the so-called *software crisis*. More than ever, software engineering needs diversity of traits.

Finally, it takes variety to conquer complexity. Putting it in software context, diversity of skills and personalities are needed to solve the myriad of problems related to software development and maintenance. Organizations would benefit from a conscious attempt to diversify the styles or personalities of their software engineers, since strong teams are the ones made up of diverse perspectives. Exposure to software psychology can help this diversity to flourish. This variety will enable us to bring a richness of talents and points of view to bear upon the inherent complexity of software systems.

References

[1] S. Nash (1999): Turning Team Performance Inside Out, Davies-Black Publishing, Mountain View, CA.

[2] I.B. Myers, M.H McCaulley, N.L. Quenk, and A.L. Hammer (1998): MBTI Manual: A Guide to the Development and Use of the Myers-Briggs Type Indicator, Consulting Psychologists Press, Mountain View, CA.

[3] N.A. Schaubhut and R.C. Thompson (2008): MBTI Type Tables for Occupations, Consulting Psychologists Press, Mountain View, CA.

[4] P.D. Tieger and B. Barron (2007): Do What You Are, 4th ed., Little Brown and Company, New York, NY.

[5] J.S. Karn and A.J. Cowling (2006): Using Ethnographic Methods to Carry Out Human Factors Research in Software Engineering. Behavior Research Methods, vol. 38, n. 3, pp. 495-503.

[6] M.T. Russell and D.L. Karol (1994): 16PF Fifth Edition Administrator's Manual, Institute for Personality and Ability Testing.

[7] S.T. Acuna, N. Juristo, and A.M. Moreno (2006): Emphasizing Human Capabilities in Software Development. IEEE Software, vol. 23, n. 2, pp. 94-101.

DOI: 10.1145/1734103.1734111 http://doi.acm.org/10.1145/10.1145/1734103.1734111

[8] R. Feldt, R. Torkar, L. Angelis and M. Samuelsson (2008): Towards Individualized Software Engineering: Empirical Studies Should Collect Psychometrics. Workshop on Cooperative and Human Aspect of Software Engineering (CHASE), 2008, Leipzig, Germany, ACM, pp. 49-52.

[9] T. Buchanan, J.A. Johnson and L.R. Goldberg (2005): Implementing a Five-Factor Personality Inventory for Use on the Internet. European Journal of Psychological Assessment, vol. 21, n. 2, pp. 116-128.

[10] J.E. Hannay, E. Arisholm, H. Engvik, and D.I.K. Sjoberg (2010): Effects of Personality on Pair Programming. IEEE Transactions on Software Engineering, vol. 36, n. 1, pp. 61-80.

[11] L.R. Goldberg (1990): An Alternative Description of Personality: The Big-Five Factor Structure. Journal of Personality and Social Psychology, vol. 59, pp. 1216-1229.

[12] D. Shneiderman (1980): Software Psychology: Human Factors in Computer and Information Systems, Winthrop Publishers, Cambridge, MA.

[13] D.B. Walz and J.L. Wynekoop (1997): Identifying and Cultivating Exceptional Software Developers. Journal of Computer Information Systems, vol. 37, n. 4, pp. 82-87.

[14] E.A. Turley and J.M. Bieman (1995): Competencies of Exceptional and Non-Exceptional Software Engineers. J. of Systems and Software, vol. 28, n. 1, pp. 19-38.

[15] N.L. Kerth, J. Coplien, and J. Weinberg (1998): Call for the Rational Use of Personality Indicators. IEEE Computer, vol. 31, n. 1, pp. 146-147.

[16] D.J. Pittenger (1993): The Utility of the Myers-Briggs Type Indicator. Review of Educational Research, vol. 63, n. 4, pp. 467-488.

[17] E. Kaluzniacky (2004): Managing Psychological Factors in Information Systems Work, Information Science Publishing, London.

[18] L.T. Hardiman (1997): Personality Types and Software Engineers. IEEE Computer, vol. 30, n.10, pp. 10.

[19] L.F. Capretz (2003): Personality Types in Software Engineering. International Journal of Human-Computer Studies, vol. 58, n. 2, pp. 207-214.

[20] C. Bishop-Clark (1995): Cognitive Style, Personality, and Computer Programming. Computers in Human Behaviour, vol. 11, n. 2, pp. 241-260.

[21] G.J. Teague (1998): Personality Type, Career Preference and Implications for Computer Science Recruitment and Teaching. Proceedings of the Third Australian Conference on Computer Science Education, 1998, ACM, pp. 155-163.

[22] L.F. Capretz and F. Ahmed (2010): Making Sense of Software Development and Personality Types. IEEE IT Professional, vol. 12, n. 1, pp. 6-13.

[23] C.R.B. DeSouza, H. Sharp, J. Singer, L.Cheng, and G. Venolia (2009): Cooperative and Human Aspects of Software Engineering. IEEE Software, vol. 26, n. 6, pp. 17-19.

[24] G.M. Weinberg (1998): The Psychology of Computer Programming, 2nd Edition, Van Nostrand Reinhold, New York, NY.

[25] J. Dolney (2009): Designing Job Descriptions for Software Development. In Information Systems Development Challenges in Practice, Theory and Education, ed., C. Barry. Springer, pp. 447-460.

[26] L.F. Capretz (2002): Implications of MBTI in Software Engineering Education. ACM SIGCSE Bulletin, vol. 34, n. 4, pp. 134-137.

Luiz Fernando Capretz has almost 30 years of international experience in the software engineering field as a practitioner, manager and educator. Having worked in Brazil, Argentina, U.K., Japan, Italy, and the United Arab Emirates, he is currently an Associate Professor and the Director of the Software Engineering Program at the University of Western Ontario, Canada. He has published over 100 peer-reviewed research papers on software engineering in leading international journals and conference proceedings, and he has co-authored two books in the area. His present research interests include software engineering (SE), human factors in SE, software estimation, software product lines, and software engineering education. Dr. Capretz received his Ph.D. in Computing Science from the University of Newcastle upon Tyne (U.K.), his M.Sc. in Applied Computing from the National Institute for Space Research (INPE, Brazil), and his B.Sc. in Computer Science from State University of Campinas (UNICAMP, Brazil). He is an IEEE senior member, ACM distinguished member, MBTI certified practitioner, Professional Engineer in Ontario (Canada), and he can be contacted at lcapretz@eng.uwo.ca.

Faheem Ahmed received his M.E.Sc. (2004) and Ph.D. (2006) in Electrical Engineering from the University of Western Ontario (Canada). Currently he is an assistant professor at the College of Information Technology, United Arab Emirates University, Al Ain, United Arab Emirates. Ahmed has several years of industrial experience, holding various technical positions in software development organizations. During his professional career, he has been actively involved in the entire life cycle of the software development process, including requirements management, system analysis and design, software development, testing, delivery and maintenance. Ahmed has authored and co-authored many peer-reviewed research articles in leading journals and conference proceedings in the area of software engineering, and he has co-authored the book, Software Product Lines: A Process Assessment Methodology – A Practitioner's Approach, published by VDM-Verlag. Ahmed's current research interests are software product lines, software process modelling, software process assessment, and empirical software engineering. He is a member of IEEE, and he can be reached at f.ahmed@uaeu.ac.ae.

DOI: 10.1145/1734103.1734111 http://doi.acm.org/10.1145/10.1145/1734103.1734111