Improving the teaching of software engineering

Luiz F. Capretz

University of Western Ontario
London, Canada

ABSTRACT: It has been suggested that different learning styles are associated with different personality types. It is believed that if software engineering teachers know the personality profiles of their students, this in turn will encourage the teachers to vary their own teaching styles to conform to the learning preferences of their pupils. The purpose of this research is twofold: firstly, this investigation presents a survey that shows the personality profile of a group of software engineering students; secondly, the article relates their personality profiles (which influence the way students learn) to the various ways a teacher can deliver a lecture in software engineering. In order to reach different types of students in a course, teachers should be aware of several teaching strategies where it is recommended that a range of exercises, assignments and activities be utilised in the classroom so that no software engineering student is left out.

INTRODUCTION

Jung's theory of psychological types assumes that much apparently random behaviour is actually quite orderly and consistent. These consistencies result from differences in the ways people perceive their surroundings and make decisions.

Myers had the vision to apply that knowledge, determining how people take in information, express their thoughts and feelings, and go about their daily lives [1]. The Myers-Briggs Type Indicator (MBTI) is based on Jung's theory that people with different personality profiles organise information and perceive the world in different ways. The theory of psychological types has the power to transform human relationships, particularly teacher-student dynamics.

In short, the MBTI includes four internal scales related to characteristic or preferred ways of becoming aware, reaching conclusions, decision making and general orientation to a private inner world or external world of actions. These dimensions are called:

- Introversion (I) and extroversion (E);
- Sensing (S) and intuition (N);
- Thinking (T) and feeling (F);
- Perception (P) and judging (J).

To expand on this, E's prefer to work interactively with a succession of people, whereas I's prefer work that permits some solitude; N's prefer working on a succession of new problems and S' prefer working with details; T's want work that requires logical thinking, whereas F's want work that provides service to people; J's prefer work that imposes a need for order, whereas P's prefer work that requires adapting to changing situations.

We all have personality qualities of each scale or parameter; we simply prefer some qualities or are more comfortable with some styles than others, just as right-handers are more comfortable with the right hand but sometimes use the left hand.

The MBTI describes 16 types that result from the dynamic interplay of these four preferences – EI, SN, TF, JP. Types are denoted by the letters of preferred orientations (such as ISTJ, ENFP, INTP, etc), as shown in Table 1, together with the percentage of type distribution of the adult population in the USA [1]. It is important to understand that everyone uses all eight preferences, not merely the four that are preferred. The theory describes 16 distinct ways of being normal; no preference is superior over any other preference, and no type is superior over any other type, although in a given situation, the preferences of one type may match the demands of the situation better than those of a different type.

Table 1: The 16 MBTI types.

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%

SOFTWARE ENGINEERING STUDENTS

A sample of 68 software engineering students were invited to participate in this study to identify the personality profile of a group of software engineering students, and were administered the MBTI (Form G) to determine their personality types. This

investigation considered students in upper level university classes. The type distribution of the software engineering students is summarised in Table 2 [2].

Table 2: Type distribution of software engineers (N = 68).

ISTJ N=13 19.1%	ISFJ N=2 2.9%	INFJ N=1 1.5%	INTJ N=5 7.4%
ISTP N=3 4.4%	ISFP N=3 4.4%	INFP N=2 2.9%	INTP N=9 13.2%
ESTP N=8 11.8%	ESFP N=1 1.5%	ENFP N=2 2.9%	ENTP N=5 7.4%
ESTJ N=8 11.8%	ESFJ N=2 2.9%	ENFJ N=1 1.5%	ENTJ N=3 4.4%

This study has shown that ISTJ, ESTP, ESTJ and INTP comprise almost 55% of the sample and are therefore significantly over-represented, whereas ESFP, INFJ and ENFJ are all particularly underrepresented in this sample. It is also worth noting that there are more ISTJ (19%) than any other type. This research also found more introvert (I=54%) than extrovert (E=46%) types; slightly more sensing (S=57%) than intuitive (N=43%); significantly more thinking (T=81%) than feeling (F=19%); and fairly more judging (J=54%) compared to perception (P=36%) type.

It should also be noticed that TJ's comprise 46% of the sample, IT's comprise 43%, ST's compose 46% and NT's make up to 36% of the subjects. On the other hand, SF's add up to 11% only, and NF's a mere 8% of the subjects. TJ's, ST's and NT's are abundant among software students. On the other hand, SF's and NF's are scarce. A cluster of sensing, thinking and judging types (STJ's) were also found and is generally in line with type theories.

Although software engineering attracts people of all psychological types, certain traits are clearly more represented than others in this field. These findings do not mean that career success relates to the number of subjects of a type. The fact that ISTJ's outnumber any other type does not mean that they are perceived to be the best in the area. Ackerman suggests that, although interests and personality types may play a role in the selection of a career, they may not predict success in that particular area [3].

Moreover, Sodan claims that, in order for personal work and relationships to be successful, a number of psychosocial qualities are required, and she proposes a model based on the Yin/Yang duality [4]. Due to the diverse nature of software engineering, it is widely believed that no personality instruments will ever accurately predict success in this area.

As a matter of fact, the software field is dominated by introverts and thinking types, who typically have difficulty in communicating with users and empathising with their problems. This may partially explain why software systems are notorious for not meeting users' requirements. When software engineers discuss how a task needs to be accomplished, the majority tend to be poor at verbalising how the task affects the people involved. In fact, the greatest difference between software engineers and the general population is the percentage that takes action based on what they think rather than on what somebody else feels; this does not help bring software engineers any closer to users.

It takes variety to conquer variety. Putting this in software terms, it takes a variety of skills and personalities to solve the myriad of problems related to software development. It might be suggested that organisations would be well served by a conscious attempt to diversify the styles or personalities of their software engineers.

Nowadays, there are very few solo performers in most software organisations; people have to work together in teams of some sort, and it is almost always good to have some diversity on the team in terms of psychological types. In other words, better software will result from the combined efforts of a variety of mental processes, outlooks and values. Therefore, all types are important to software development as every type can make a contribution to solve the so-called *software crisis*. Thus, the software industry cannot afford to lose would-be professionals who may come from a diverse group of students. But a crucial question still remains: how to reach all types of students so that they can be retained in a software engineering programme?

REACHING SOFTWARE ENGINEERING STUDENTS

A number of approaches exist to aid the understanding of individual differences and their effects on teaching and learning. Educators have been using the Myers-Briggs Type Indicator (MBTI) to understand differences in learning styles and to develop teaching methods that cater for the various personality styles. Inspired by the MBTI, teachers have developed a range of practices for effective teaching and learning in a software engineering course. By devising various approaches to teaching, teachers aim to reach every student in different ways [5].

Understanding Extroverts and Introverts

Teachers can conduct classes with opportunities to talk and problem solving aloud or in groups. Extroverts often learn better when they can talk about the concepts they have just heard in a lecture. They learn best when they have action projects before or along the instruction. On the other hand, introverts like to think through a problem before talking about it; they should be given adequate time to formulate their responses before discussing it and are more comfortable when they can prepare their responses in advance.

In one of his lectures, the author, immediately after a lesson on software design, asked his students to come up with a quick design for a weather station system. The author divided the students into groups so that each group contained only extroverts or introverts. The groups with extroverts enjoyed the exercise much more than the introverts, and came up with a better design solution in a shorter period of time. The author believes that, given the time and opportunity to do the exercise as homework, members from the introverts' groups would be able to work out good solutions as well.

Recommended Tasks for Extroverts and Introverts

Extroverts: The task objective is to understand more clearly the difficulties of carrying out the requirements specification for a software system. Students are divided into groups of four

people, in which two of them act as users (or clients), while the other two act as systems analysts. A possible scenario for the above role-play exercise occurs when the management from a multi-screen cinema complex has decided that it is time to replace its current manual ticket issue system with a new state-of-the-art computer system.

Introverts: As they need time, quiet and space for internal processing, once a task is assigned, allow them some private time to reflect on the assignment and organise their thoughts before expecting participation. A good task could be:

- Make a list of all software development tools that already used. Classify them as standalone or integrated tools. Which activities of the software lifecycle does each one support? Which ones solely support engineering requirements?

Challenging the Sensing and Intuitive

Sensing students favour understanding from *trying it out* compared with intuitive students who are more inclined to *think it through*. However, intuitive teachers find it easier to deal with concepts rather than facts and prefer teaching courses that deal with *ideas and theories* rather than *real life situations*. For effective teaching, it is important for faculty to acknowledge their own natural inclination towards intuition and to make a conscious effort to recognise the learning preferences of their sensing students.

By frequently introducing specific examples, facts and practical applications, the sensing students will profit more from a software engineering course that gives them the chance to come up with real-life designs using particular methodology, rather than just listening to the main concepts and formalities dictated by a design methodology.

Meaningful Exercises for Sensing and Intuitives

Sensing: These students prefer the concrete to the abstract, and tend to learn best in a step-by-step progression. As they rely on experience rather than theory, these students should be provided with two or three practical examples each time a concept is introduced. Audiovisuals should also be used, like movies and models, as straight lectures usually are not enough to attract their attention. Hands-on exercises will engage their senses, such as comparing the similarities and differences between software design and hardware design.

Intuitives: They thrive in classroom situations that place a premium on imagination, but are bored with factual lectures and drills. As they need opportunities to be creative and original, intuitive students should be challenged with problems for which there are multiple solutions or different perspectives. Exercises such as the following should be proposed:

- Write down a list of reasons in favour of using any standardised design description (eg UML) and a list of reasons against standardising the same form of description.
- When the Ariane-5 rocket was destroyed, the news made headlines in France. The *Liberation* newspaper called it *A 37-billion-franc Fireworks Display* on the front page. What is the responsibility of the press when reporting software related accidents?

Caring for the Thinking and Feeling Types

Software engineers need not only a broad-based technical competence but also the ability to cope with societal change and personal relationships. They need an appreciation of society's ethical problems and the interpersonal skills to work effectively in teams towards a common solution. Therefore, feeling types are much needed as software engineers.

Those feeling students who find it difficult to go through a software engineering course may be retained if teaching is enhanced to encompass their preferred learning styles. Specific addition to courses might include greater discussion of design aesthetics, ethics, social values and human factors. These issues are particularly dealt with in the author's software engineering courses. Two lectures in the course, which focus on human factors in software engineering and ego-less programming, have been incorporated to appeal more to students who have a feeling preference.

Suggested Assignments for Thinking and Feeling Types

Thinking: These students excel in inductive reasoning and perform well when there is a single correct answer. A possible assignment for them would be:

- A well-known word processor consists of a million lines of code. Calculate how many programmers would be needed to write it, assuming that it has to be completed in two years. Given that they are each paid $50,000 per year, what are the costs of that development? (Remember that the average programmer productivity is 20 lines of code per day).

Feeling: These students are skilled in understanding other people, so they should be presented with opportunities for friendly interaction and positive feedback. An example of an assignment is:

- Suppose you are the manager of a software development project. One of the team members fails to meet the deadline for the coding and testing of a module. What would you do? For the same software project, three months before the software is due to be delivered, the customer requests a change that will require massive efforts. What would you do?

Helping the Judging and Perceiving Types

Research has shown that the majority of teachers hold a preference for judging, and thus demonstrate biases for order and structure in the classroom. A teacher should use previous successes in order to reinforce learners' progression in a systematic manner towards a specific outcome.

Nevertheless, a less organised (or somewhat lax) system of instruction could be followed as well. Under such a scheme, students determine their own rate and amount of learning, considering their preferences, as they advance through a series of tasks. With this method, the teacher acts as a motivator through the use of cues and support on those tasks being carried out in such a way that a student has the autonomy to take up a particular task, learn and then move on to the next activity.

Suggested Activities for Judging and Perceiving Types

Judging: These students like schedule and predictability, and closure of one topic before moving to the next. In order to cater for them, judging preference students should be provided with a course outline, showing those topics to be covered in each grading period. A marking system should be utilised that recognises and honours individual achievement. For instance:

- In order to pass the capstone project course, a student must design and implement a prototype for a small software system. In this course, designing, coding and testing should be carried out by each of the students individually or in teams under the supervision of a faculty member.
- Progress report during the course and a final report should be prepared. Each student must deliver a public lecture on the work performed, followed by a demonstration on the prototype developed.
- The deadlines and marking scheme are: prepare a project proposal (3 weeks-10%), demonstrate a design (10 weeks-10%), write a mid-term report (2 weeks-20%), carry out an implementation (10 weeks-10%), deliver a public lecture (2 weeks-10%), run a software demonstration (1 week-10%), and write a final report (2 weeks-30%).

Perceiving: Perceivers tend to resist closure. They prefer spontaneity so that they can explore things without pre-planning. They like to work on multiple tasks simultaneously and often work right up to, and even beyond the deadlines. An instructor needs to make sure that there is some allowance for flexibility as too many rules weigh heavily on this type of student. Perceiving students could be assisted by teaching them to work backwards from a deadline, this can be achieved by helping them determine the latest date at which a project can be started and still meet expectations, or even dealing with some deadline slippage. The student's progress is thereby improved and learning is unhindered when reasonable bonuses and penalties are used. It is far better to present an excellent report one day after the deadline than to produce a lousy report in advance; such a compromise can be easily applied to project courses.

CONCLUSION

The ideas described in this article have been followed in two 4th year courses named *Software Requirements*, and *Testing and Risk Assessment*, which go hand-in-hand with another course on *Software Engineering Design II*, wherein students develop their capstone projects at the University of Western Ontario, London, Canada.

The results have demonstrated that these ideas are extremely effective in producing significant projects and have also increased students' satisfaction with these courses. This last point was expressed and reinforced in course evaluations given by students. The course evaluations are monitored by the University and made available to the public worldwide over the Internet.

Finally, a respectful learning environment invites both instructors and students to listen and be heard, it respects individual differences and honours each person's right to hold his/her beliefs and values; it encourages personal self growth and other aspects. This is a collective responsibility, so it depends on both students as well as educators.

REFERENCES

1. Myers, I.B., McCaulley, M.H., Quenk, N.L. and Hammer, A.L., *MBTI Manual*. Palo Alto: Consulting Psychologists Press (1998).
2. Capretz, L.F., Are software engineers really engineers? *World Trans. on Engng and Technology Educ.*, 1, **2**, 233-235 (2002).
3. Ackerman, P.L., A theory of adult intellectual development: process, personality, interests and knowledge. *Intelligence*, 22, 229-259 (1996).
4. Sodan, A.C., Toward successful personal work and relations – applying a Yin Yang model for classification and synthesis. *J. of Social Behaviour and Personality*, 27, **1**, 38-71 (1999).
5. Capretz, L.F., Implications of MBTI in software engineering education. *ACM SIGCSE Bulletin - Inroads*, 34, **4**, 134-137 (2002).