# Selected topics in software development

**Today:**

Personality

**Speaker:**

Jyrki Katajainen

- What to do to continue improving?
- Who should work with whom?

**Course home page:**

http://www.diku.dk/forskning/performance-engineering/Software-development/

# Potential topics on programming psychology

- programming as human performance
- programming as an individual activity
- programming as a social activity
- programming tools

[Gerald M. Weinberg, The psychology of computer programming]

# Differences in programming performance

Several studies have found differences on

- the order of 10 to 1 in the time to create a program;
- the order of 10 to 1 in the time required to debug a program; and
- the order of 10 to 1 in the resulting size, speed, error rate, and number of errors detected.

[Steve McConnell, Code complete, 2nd edition, §33.1]

# Diversity of skills and traits needed

As an example, let us consider testing.

1. Detecting the presence of errors requires a conniving mind, one that delights in uncovering flaws. Perhaps a touch of paranoia helps.
2. Locating errors which are known to exist requires a person who has the persistence of a mother-in-law and the collecting instincts of a pack rat.
3. Correcting errors that have been found requires a sense of proportion, plus an extensive repertoire of tricks for one machine or language. The *patcher* must have a *synthetic* mind, as opposed to the *analytical* mind of the *bug-finder*.

[Gerald M. Weinberg, The psychology of computer programming, §7]

# Humility

- The people who are best at programming are the people who realize how small their brains are. They are humble.
- The point of *decomposing* a system is to make it simpler to understand.
- Conducting reviews, inspections, and tests is a way of compensating for anticipated human fallibilities.
- Keeping routines short reduces the load on your brain.
- Writing programs in terms of the problem domain rather than in terms of implementation details reduces your mental workload.
- Using conventions of all sorts frees your brain from the relatively mundane aspects of programming.

[Steve McConnell, Code complete, 2nd edition, §33.2]

# Curiosity

- Build your awareness of the development process.
- Experiment!
- Read about problem solving.
- Learn about successful projects.
- Read manuals, books, and periodicals!
- Affiliate with other professionals.
- Make a commitment to professional development. (If you can't learn at your job, find a new one.)

[Steve McConnell, Code complete, 2nd edition, §33.3]

# Intellectual honesty

- Refuse to pretend you are an expert when you are not.
- Readily admit your mistakes.
- Try to understand a compiler warning rather than suppressing the message.
- Clearly understand your program—not compile it to see if it works.
- Provide realistic status reports.
- Provide realistic schedule estimates and hold your ground when management asks you to adjust them.

[Steve McConnell, Code complete, 2nd edition, §33.4]

# Communication and cooperation

- Truly excellent programmers learn how to work and play well with others.
- Writing readable code is part of being a team player.
- Keep the person who has to modify your code in mind.
- Programming is communicating with another programmer first and communicating with the computer second.

[Steve McConnell, Code complete, 2nd edition, §33.5]

# Discipline

- To write a large program requires discipline.
- To write a complicated program, even if it is a small one, requires discipline.
- Establish conventions in non-critical areas so that you can focus your creative energies in the places that count.
- Methods and tools that emphasize human discipline are especially effective. Form is liberating!

[Steve McConnell, Code complete, 2nd edition, §33.6]

# Laziness

- Do an unpleasant task quickly to get it out of the way. This is called **enlightened laziness**.
- Or write a tool to do the unpleasant task so that you ever have to do the task again.
- The most important work in effective programming is thinking. Think before you act! This is a productive kind of laziness since lots of unpleasant work may be avoided altogether.

[Steve McConnell, Code complete, 2nd edition, §33.7]

# Theory of psychological types

**Psychological types** define the patterns of conscious mental activity which are persistent, predictable, and meaningful in human interaction.

The theory of psychological types was introduced by Carl Gustav Jung (1923) and further refined by Isabel Briggs Myers (1962).

The Myers-Briggs Type Indicator (MBTI), which is based on Jung's theory, is one of the most used instruments to assess the personality of people.

**Reading:** §5 of our book *Reengineering a university department*

# MBTI type dynamics

| **Orientation:** The way in which people direct and draw energy | |
|---|---|
| **Extroversion (E):** Focus on the outer world of people and activity | **Introversion (I):** Focus on the inner world of thoughts, ideas, and experiences |

| **Perception:** The way in which people gather information | |
|---|---|
| **Sensing (S):** Pay attention to details and current realities | **Intuition (N):** Pay attention to meanings, patterns, and future possibilities |

| **Judgement:** The way in which people organize information and make decisions | |
|---|---|
| **Thinking (T):** Make decisions based on principles and logical consequences | **Feeling (F):** Make decisions based on values and consequences for people |

| **Attitude:** The way in which people structure, or live, their lives | |
|---|---|
| **Judging (J):** Come to conclusions quickly and enjoy the structure provided by reaching closure | **Perceiving (P):** Take more time to gather information before comfortably coming to closure, enjoy the process, and are comfortable being open-ended |

[http://www.myersbriggs.org/]

# Dominant mental function

The **dominant function** is the most developed and used of all four mental functions (S, N, T, and F). There are four kinds of intelligence: practical, theoretical, logical, and emotional, each corresponding to the four mental functions.

```
char dominant(char[4] type):
    if type[0] == 'I' and type[3] == 'J':
        return type[1]
    if type[0] == 'I' and type[3] == 'P':
        return type[2]
    if type[0] == 'E' and type[3] == 'J':
        return type[2]
    if type[0] == 'E' and type[3] == 'P':
        return type[1]
```

[Christopher Derek Curry & Jyrki Katajainen, Reengineering a university department, §5]

# Sample type description: ENFJ

Warm, empathetic, responsive, and responsible. Highly attuned to the emotions, needs, and motivations of others. Find potential in everyone, want to help others fulfil their potential. May act as catalysts for individual and group growth. Loyal, responsive to praise and criticism. Sociable, facilitate others in a group, and provide inspiring leadership.

[http://www.myersbriggs.org/]

# Traits vs. psychological type

It should be emphasized that a type preference is not a trait. Traits are used to help to define preferences. Type theorists assert that type is inborn. That is, people cannot change their type preferences, but can acquire the traits that define opposite preferences.

[Gordon Lawrence, People types & tiger stripes, pp. 35–36]

# Taking the MBTI instrument

- You fill out a multiple-choice questionnaire either in paper form or online. You select the answers that best fit for you.
- Result are given in person or by phone through an open discussion with a qualified practitioner. The feedback process involves the professional explaining the history and aims of the MBTI, along with a description of the four dimensions and a self-assessment of your type.
- Scored results come in the form of a report that is either delivered via the web or given in printed form. This report is confidential.
- People who are qualified to administer the MBTI instrument are committed to using it in an ethical way, which includes protecting your confidentiality, showing you how to verify your type, giving feedback interactively, and presenting all types as valuable.

[http://www.myersbriggs.org/]

# The 16 personality types

- The guardians (□S□J)
  - Administrators (□STJ)
    - Inspectors (ISTJ)
    - Supervisors (ESTJ)
  - Conservators (□SFJ)
    - Protectors (ISFJ)
    - Providers (ESFJ)
- The rationals (□NT□)
  - Engineers (□NTP)
    - Architects (INTP)
    - Inventors (ENTP)
  - Coordinators (□NTJ)
    - Masterminds (INTJ)
    - Field marshals (ENTJ)
- The idealists (□NF□)
  - Advocates (□NFP)
    - Healers (INFP)
    - Champions (ENFP)
  - Mentors (□NFJ)
    - Counselors (INFJ)
    - Teachers (ENFJ)
- The artisans (□S□P)
  - Entertainers (□SFP)
    - Composers (ISFP)
    - Performers (ESFP)
  - Operators (□STP)
    - Crafters (ISTP)
    - Promoters (ESTP)

[http://keirsey.com/]

---

All types are equal, there is no best type.

---

[http://www.myersbriggs.org/]

Psychological types do **not** indicate levels of skills, but merely preferences of people.

[http://www.myersbriggs.org/]

# Application areas

Type can be introduced into an organization to support many different functions and situations including managing others, development of leadership skills, organizing tasks, creation and management of teams, training for management and staff, conflict resolution, executive coaching, and change management.

Other application areas include: vocational guidance, marriage guidance counselling, selling and influencing, health-care communication, and personal development.

[http://www.myersbriggs.org/]

# Self-awareness and awareness of others

When you understand your type preferences, you can approach your own work in a manner that best suits your style, including how you manage your time, problem solving, best approaches to decision making, and dealing with stress. Knowledge of type can help you deal with the culture of the place you work, the development of new skills, understanding your participation on teams, and coping with change in the workplace.

Knowledge of personality type within a team allows people to expect others to react differently from themselves and equip them to cope more constructively with those differences.

[http://www.myersbriggs.org/]

# Fallacy

- In one study, the type of 100 software engineers was determined; the majority of them was of type ISTJ.
- In another study, of the 64 (student) developers studied, people of type □NT□ turned out to perform best in a code-review task.
- But, in general, personality tests cannot identify good software engineers over bad, nor can their results predict "on the job" performance.

[Sharon McDonald & Helen M. Edwards, Who should test whom?]

# Effective software project teams

20 small software-development teams, 92 developers in total

| Criterion | Dimension | Team performance |
|---|---|---|
| Team leader | Information gathering<br>Decision making | N > S<br>F > T |
| System analyst | Decision making | T > F |
| Programmer | Interaction | E > I |
| Heterogeneity leader-member | Interaction<br>Information gathering | E–I<br>N–S |
| Heterogeneity member-member | all dimensions | not significant |

> means outperforms; − means difference

[Narasimhaiah Gorla & Yan Wah Lam, Who should work with whom?]

# Recommendations for building software teams

- Heterogeneous team is best for some tasks (requirement analysis) and homogeneous team for others (programming).
- There must be a heterogeneity between the team leader and team members. (EN□□, ES□□, IN□□, and IS□□ all present)
- A team leader must be able to visualize future requirements and make decisions that affect individuals. (□NF□)
- A system analyst must use a scientific approach in decision making. (□□T□)
- A programmer should be able to communicate easily with other participants. (E□□□)

[Narasimhaiah Gorla & Yan Wah Lam, Who should work with whom?]

Success for any enterprise demands a variety of types, each in the right place.

[Isabel Briggs Myers, Introduction to type]

---

the number one mistake I've seen in team formation over the years is that of adding members to a team based on some presumed skill, rather than on compatibility of the new members with the rest of team members. Skill may be hard to come by, but it's always easier to buy or train than compatibility.

---

[Gerald M. Weinberg, The psychology of computer programming, p. 5.ii]

# Summary

- Your personal character affects your ability to write programs.
- The characteristics that matter most are humility, curiosity, intellectual honesty, discipline, and enlightened laziness.
- The characteristics of a superior programmer have almost nothing to do with talent and everything to do with a commitment to personal development.
- The theory of psychological types provides several pathways for personal development.
- To stay valuable, you have to stay current. Read!
- Good character is mainly a matter of having the right habits.

[Steve McConnell, Code complete, 2nd edition, §33]

# Professional development

**Level 1:** A **beginner** is a programmer capable of using the basic capabilities of one language.

**Level 2:** A **practitioner** is capable of using the basic capabilities of multiple languages and is very comfortable in at least one language.

**Level 3:** A **professional** has expertise in a language or an environment or both. A programmer at this level might know all the intricacies of J2EE or have the C++ standard memorized.

**Level 4:** A **guru** has the expertise of a professional programmer and recognizes that programming is only 15% communicating with the computer and 85% communicating with people.

[Steve McConnell, Code complete, 2nd edition, §33.3]

# Beginner level

James L. Adams, Conceptual blockbusting: A guide to better ideas, 4th edition.

Jon Bentley, Programming pearls, 2nd edition.

Robert L. Glass, Facts and fallacies of software engineering.

Steve McConnell, Software project survival guide.

Steve McConnell, Code complete, 2nd edition.

[Steve McConnell, Code complete, 2nd edition, §35.4]

# Practitioner level

Stephen P. Berczuk and Brad Appleton, Software configuration management patterns: Effective teamwork, practical integration.

Martin Fowler, UML distilled, 3rd edition.

Robert L. Glass, Software creativity.

Cem Kaner et al., Testing computer software, 2nd edition.

Craig Larman, Applying UML and patterns, 2nd edition.

Steve McConnell, Rapid development.

Karl Wiegers, Software requirements, 2nd edition.

NASA Goddard Space Flight Center, Manager's handbook for software development.

[Steve McConnell, Code complete, 2nd edition, §35.4]

# Professional level

Len Bass, Paul Clements, and Rick Kazman, Software architecture in practice, 2nd edition.

Martin Fowler, Refactoring: Improving the design of existing code.

Erich Gamma et al., Design patterns.

Tom Gilb, Principles of software engineering management.

Steve Maguire, Writing solid code.

Bertrand Meyer, Object-oriented software construction, 2nd edition.

NASA Goddard Space Flight Center, Software measurement guidebook.

[Steve McConnell, Code complete, 2nd edition, §35.4]