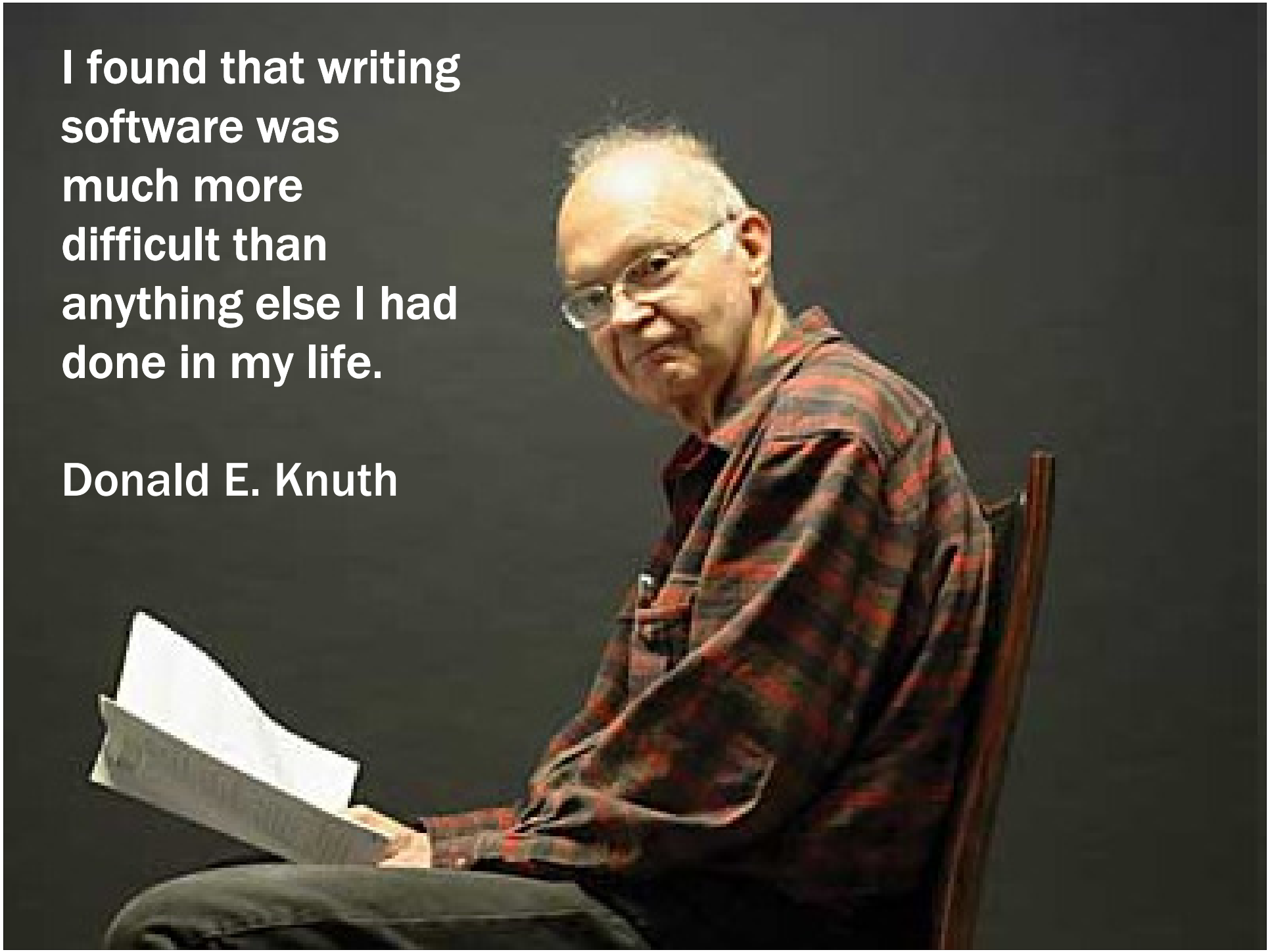
Pair Programming


I found that writing
software was
much more
difficult than
anything else I had
done in my life.
Donald E. Knuth


CLEAN CODE
CLEAN CODE
CLEAN CODE
CLEAN CODE

Pair Programming

Warum funktioniert es?

Pair Programming: Vorteile

- Höhere Disziplin. Paare entwickeln viel eher an der richtigen Stelle und machen kürzere Pausen.
- Besserer Code. Beim Pair Programming entwickelt man sich weniger leicht in Sackgassen und erreicht so eine höhere Qualität.
- Belastbarerer Flow. Pair Programming führt zwar zu einer anderen Art von Flow, ermöglicht diesen aber eher als der konventionelle Ansatz: Ein Programmierer kann seinen Partner jederzeit nach dem aktuellen Stand fragen und dort anknüpfen. Unterbrechungen werden auf diese Art besser abgewehrt.
- Höhere Moral. Pair Programming ist oft spannender und interessanter als alleine zu arbeiten.
- Collective Code Ownership. Wenn das gesamte Projektteam mit der Methode Pair Programming arbeitet und die jeweiligen Partner oft wechseln, erlangen alle Wissen über die gesamte Codebasis.
- Mentoring. Jeder hat Wissen, das andere nicht haben. Pair Programming ist eine bequeme Möglichkeit, dieses Wissen zu verteilen.
- Teambildung. Die Leute lernen sich gegenseitig schneller kennen, wodurch die Zusammenarbeit verbessert werden kann.
- Weniger Unterbrechungen. Paare werden seltener unterbrochen als jemand, der alleine arbeitet.

http://de.wikipedia.org/wiki/Paarprogrammierung

Pair Programming: Nachteile

- Kosten: Da sich Vorteile wie gesteigerte Qualität teils erst in späteren Phasen des Produktlebenszyklus bemerkbar machen, sind in der ursprünglichen Entwicklungsphase die Kosten durch die doppelte Besetzung meist höher.
- Teamfindung: Teamfindung ist aufwendig, nicht alle Personen können miteinander produktiv eingesetzt werden. Eingewöhnung der Teammitglieder erfordert Zeit.
- Autoritätsproblem: Wer hat die Kompetenz, bei konträren Problemlösungen zu entscheiden, welche implementiert wird?
- Zeitliche Belastungen: Wenn zusätzliche Aufgaben wie Mentoring während der Programmierung wahrgenommen werden müssen, kann es zu Verzögerungen in der Entwicklung kommen. Die Teilnehmer müssen sich an unterschiedliche Programmierfähigkeiten und -stile gewöhnen.
- Urheberrecht: Es kann zu Konflikten kommen, da später nicht unbedingt klar ist, wer Urheber der einzelnen Passagen des Codes ist.
- Haftung: Es kann zu Konflikten kommen, da später nicht unbedingt klar ist, wer für fehlerhaften oder urheberrechtsverletzenden Code haftet.
- Teamgröße: Bei steigender Zahl von Programmierern wird es schwieriger zu kommunizieren, wie Probleme zu lösen sind. Deshalb ist diese Arbeitsweise eher für kleinere Teams geeignet.
- Arbeitsaufkommen: Je mehr verschiedenartige Aufgaben zu bewältigen sind, desto mehr muss der Programmierer wissen.

http://de.wikipedia.org/wiki/Paarprogrammierung

Self Explanation

Mechanism 1: Pair Programming Chat

For example, Brian Kernighan and Rob Pike recommended explaining problems aloud, even to a stuffed toy, a practice that John Sturdy called the rubber-plant effect. [...]

Research on "self-explanation" by Michelene Chi and others throws some light on this question. Chi and her colleagues described a study that tested a control group of students before and after they received a textbook explanation to read. They tested another group in the same way, but encouraged the students to explain the textbook out loud and "fill in the gaps" for themselves. The self-explainers learned significantly more than the control group, and those who explained the most improved the most. The researchers also prompted the students for their explanations; they weren't just left to their own devices. In particular, they were "prompted for further clarification by the experimenter if what they stated was vague." [...]

Recent work by Rod Roscoe and Chi showed that prompting questions seems to be the key. In their study, one student (the tutor) explained material to another student (the tutee). As expected, the tutor actually learned more than the tutee, but **the questions the tutee asked made a dramatic difference in the quality of the tutor's explanations.**

Stuart Wray. How Pair Programming Really Works. IEEE Software, January/February 2010, pp. 50-55.


Detail


YouTube
selective attention test
theinvisiblegorilla.com/
gorilla_experiment.html
Learn more a
Replay
www.theinvisiblego

Mechanism 2: Pair Programmers Notice More Details

Research on change blindness and inattentional blindness illustrates something that stage magicians have known for a long time: if we don't know what to look for, we can stare right at it and still miss it. What we notice depends on what we expect to see and what we unconsciously consider salient. [...]

For example, it might seem unlikely that people would miss a woman in a gorilla suit walking into the shot in a video, but that's what half the subjects did in a study by Daniel Simons and Christopher Chabris. [...]

This second mechanism also partially explains the phenomenon of pair fatigue, which I've noticed in myself and others. When two programmers pair together, the things they notice and fail to notice become more similar. Eventually, the benefit from two pairs of eyes becomes negligible. Beck suggested that pairs should rotate at frequent intervals, perhaps once or twice a day. Arlo Belshee found that in a jelled team, rotating after two hours was optimal.

Stuart Wray. How Pair Programming Really Works. IEEE Software, January/February 2010, pp. 50-55.


Self
Discipline?

Mechanism 3: Fighting Poor Practices

An advantage of pair programming is said to be pair pressure, the feeling of not wanting to let your partner down. But why is this necessary? Why do we persist in poor programming practices when we know they're poor? Is there something special about programming that makes it more difficult to do the right thing?

Let's look at a particular example of worst practice: the code-and-fix style of programming most often used by novices (and sadly, often used by more experienced programmers). Programmers write some code that they hope will do a particular thing and then run it to see what happens. If it appears to work, they press on with other code, without systematically searching for flaws. [...]

Stuart Wray. How Pair Programming Really Works. IEEE Software, January/February 2010, pp. 50-55.

Mechanism 3: Fighting Poor Practices

In our habitual patterns of software development, we too can be conditioned by our machines. This is the special property of interactive programming that makes it difficult to do the right thing. With code and fix, we tinker haphazardly with our programs, effectively putting a coin into the slot machine each time we run our code. Slot machines are known as the most addictive form of gambling, and the similarly unpredictable rewards from code-and-fix programming mean that it could be equally addictive. [...]

Pair programmers might be less susceptible to poor practices because they can promise to write code in a particular way and ensure that each other's promises are kept. The prevalence of two-people working in jobs where human fallibility is a serious problem should lead us to seriously consider that pair pressure might be the solution for us, too. However, you can only keep a promise if you made one in the first place. We should therefore expect that to benefit from the third mechanism, programmers must agree in advance how they're going to write and test their code.

Stuart Wray. How Pair Programming Really Works. IEEE Software, January/February 2010, pp. 50-55.

Mechanism 4: Sharing and Judging Expertise

Most programmers work on problems on their own, so no one knows how good (or bad) they really are. But with pair programming, people continually work together. Because they keep swapping pairs, everyone on the team learns who's the most expert at particular things. From this comparison, they also realize their own level of expertise.

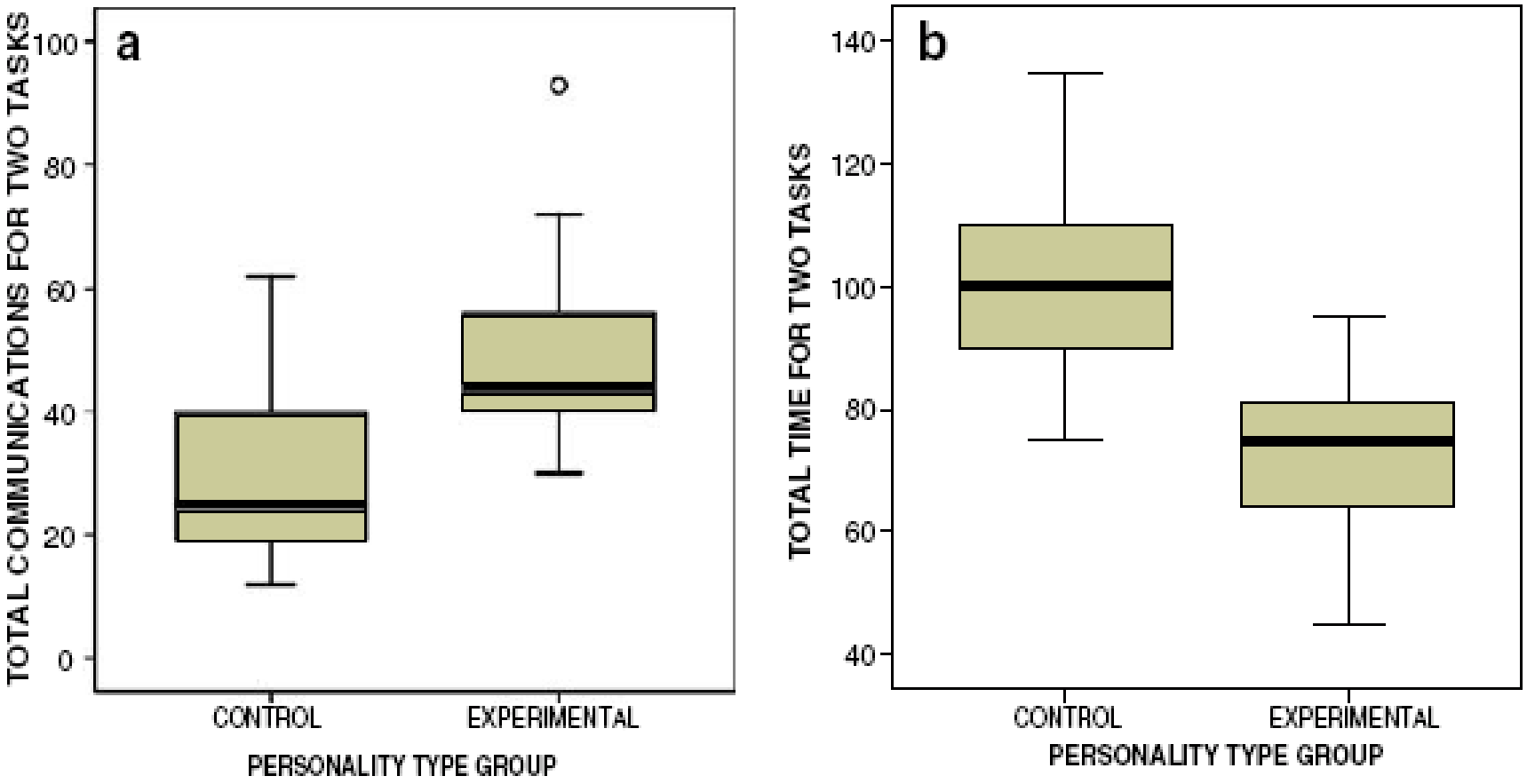
We should therefore expect more accurate estimates of time and difficulty by a pair programming team than from a solo programming team.

Stuart Wray. How Pair Programming Really Works. IEEE Software, January/February 2010, pp. 50-55.

Pair Programming

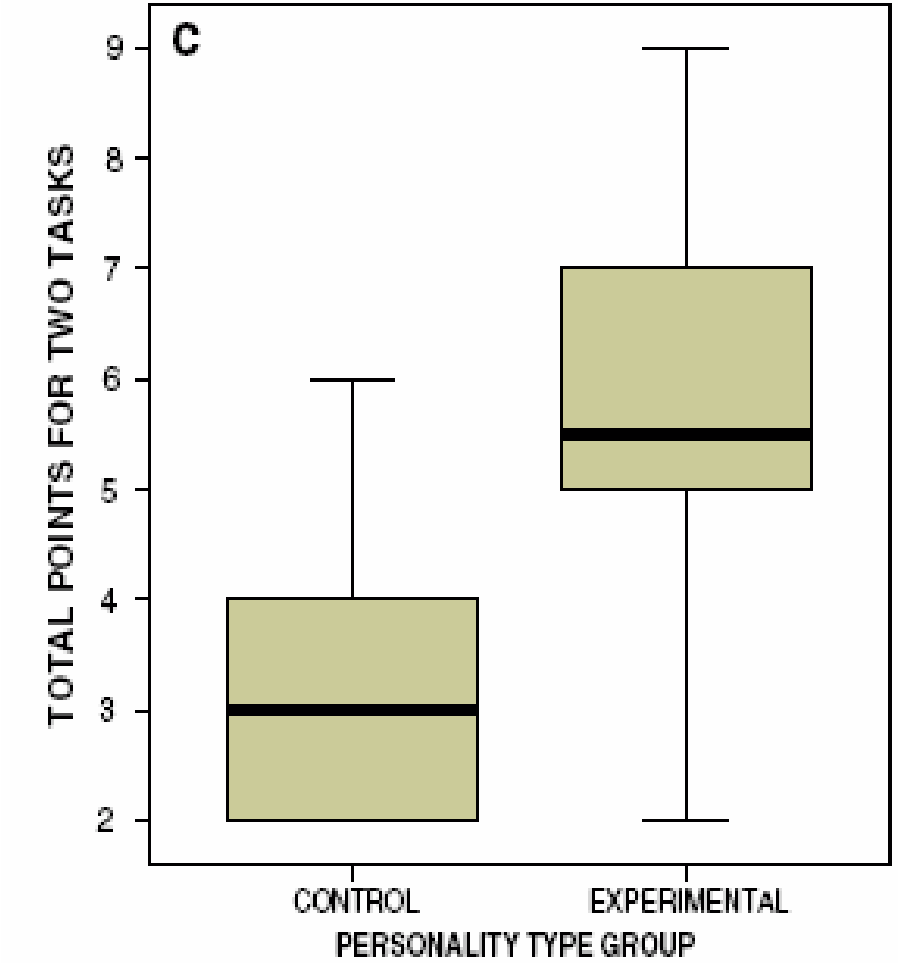
Wie funktioniert es besser?

Pairs with heterogeneous developer personalities and temperaments perform better than pairs with homogeneous developer personalities and temperaments.



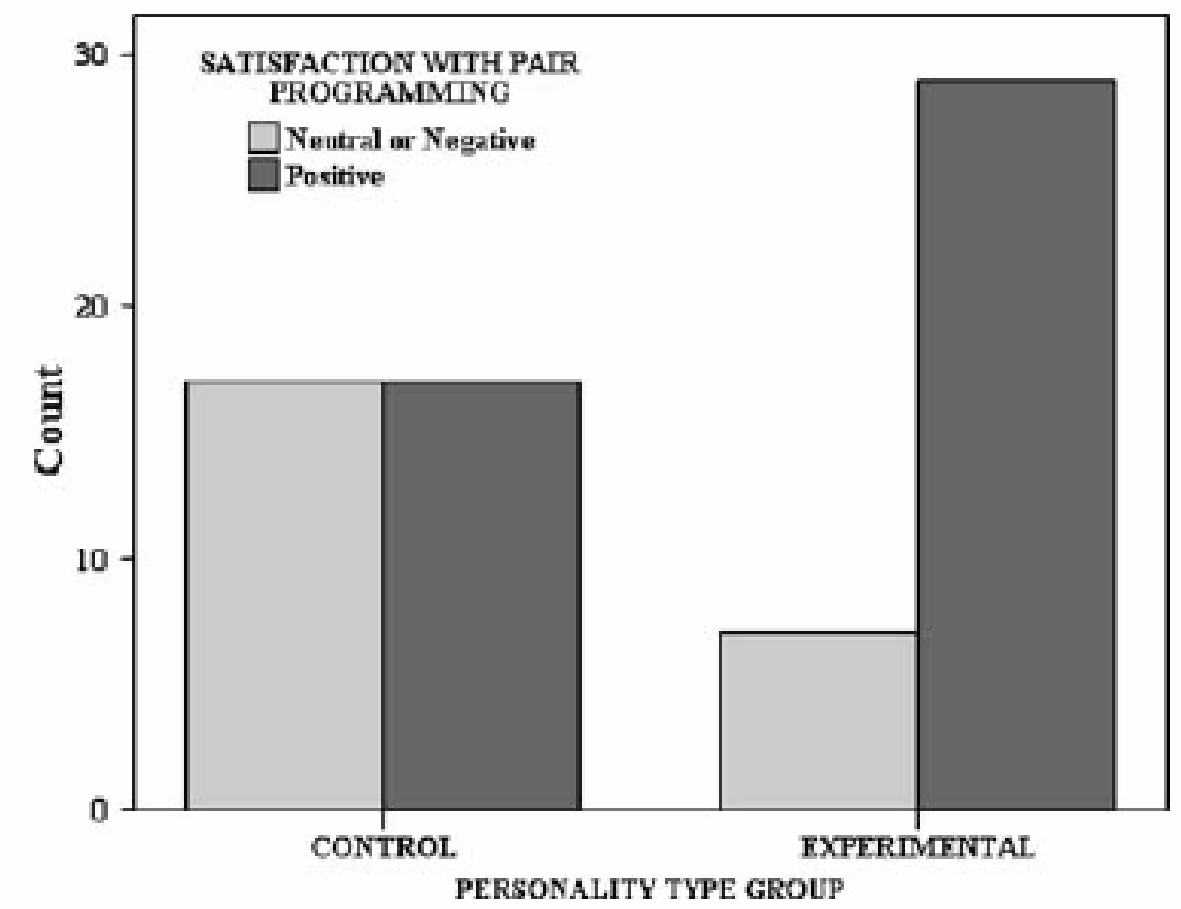
Panagiotis Sfetsos & Ioannis Stamelos & Lefteris Angelis & Ignatios Deligiannis. An experimental investigation of personality types impact on pair effectiveness in pair programming. Empir Software Eng (2009) 14:187–226.

Pairs with heterogeneous developer personalities and temperaments perform better than pairs with homogeneous developer personalities and temperaments.



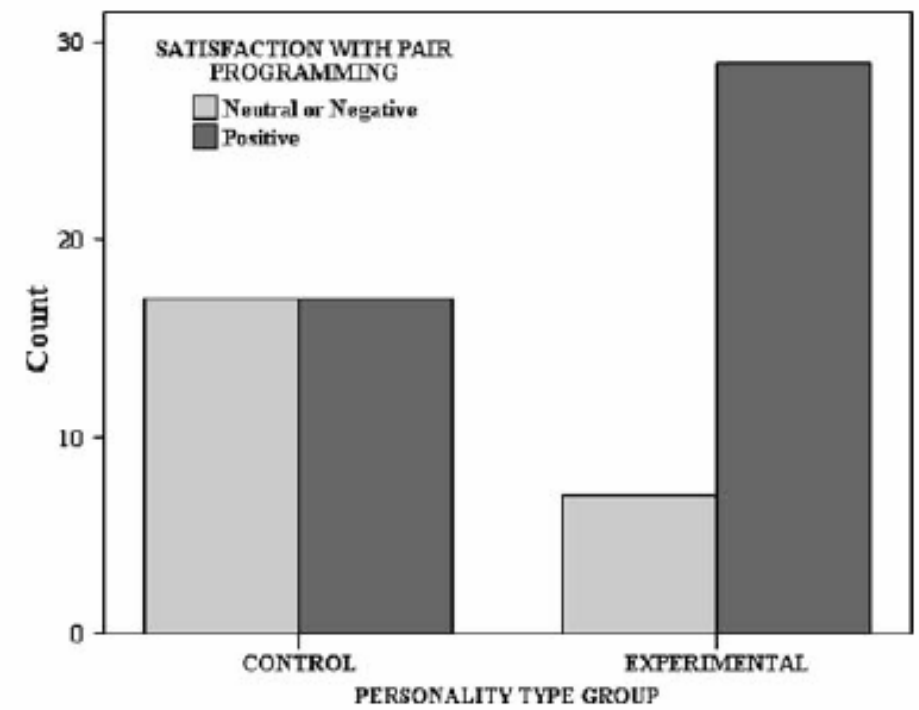
Panagiotis Sfetsos & Ioannis Stamelos & Lefteris Angelis & Ignatios Deligiannis. An experimental investigation of personality types impact on pair effectiveness in pair programming. Empir Software Eng (2009) 14:187–226.

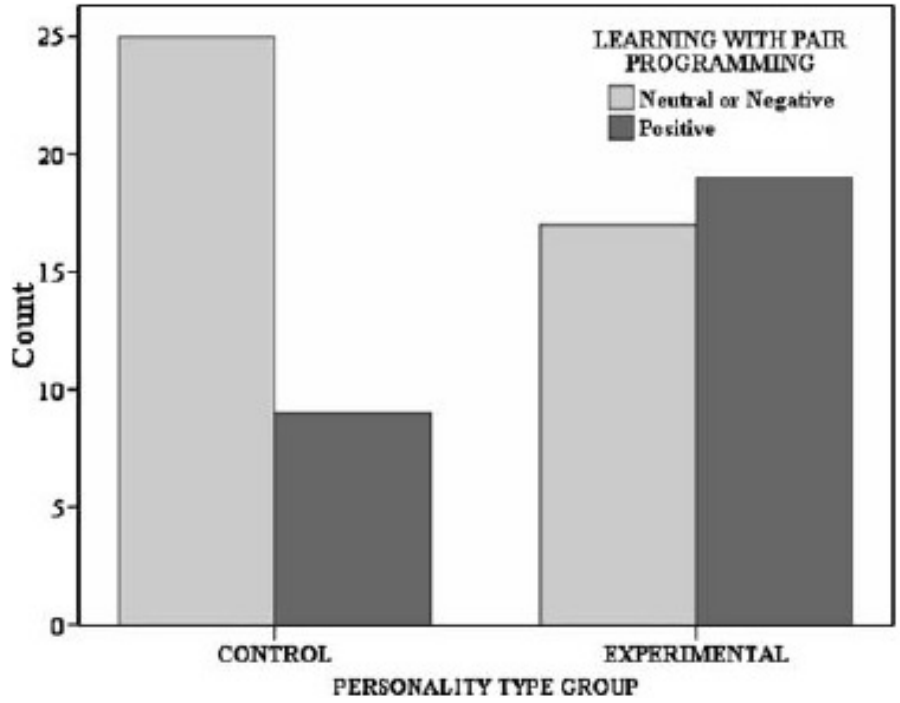
Pairs with heterogeneous developer personalities and temperaments perform better than pairs with homogeneous developer personalities and temperaments.



Panagiotis Sfetsos & Ioannis Stamelos & Lefteris Angelis & Ignatios Deligiannis. An experimental investigation of personality types impact on pair effectiveness in pair programming. Empir Software Eng (2009) 14:187–226.

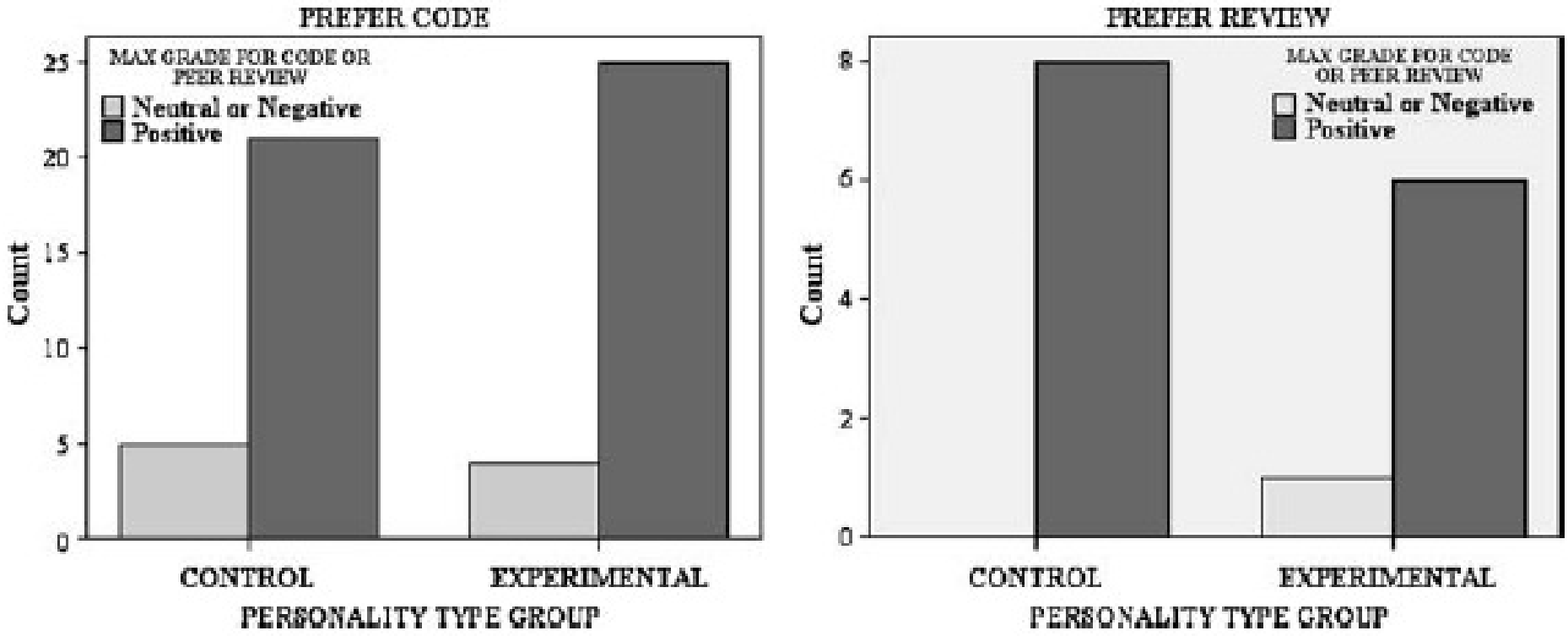
Pairs with heterogeneous developer personalities and temperaments perform better than pairs with homogeneous developer personalities and temperaments.





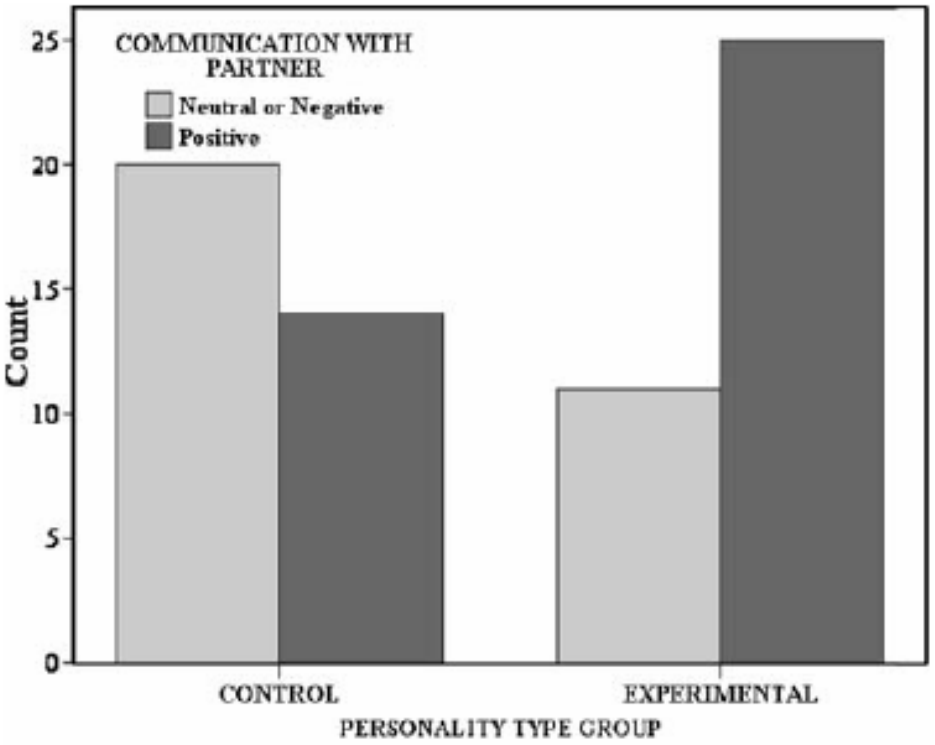
Panagiotis Sfetsos & Ioannis Stamelos & Lefteris Angelis & Ignatios Deligiannis. An experimental investigation of personality types impact on pair effectiveness in pair programming. Empir Software Eng (2009) 14:187–226.

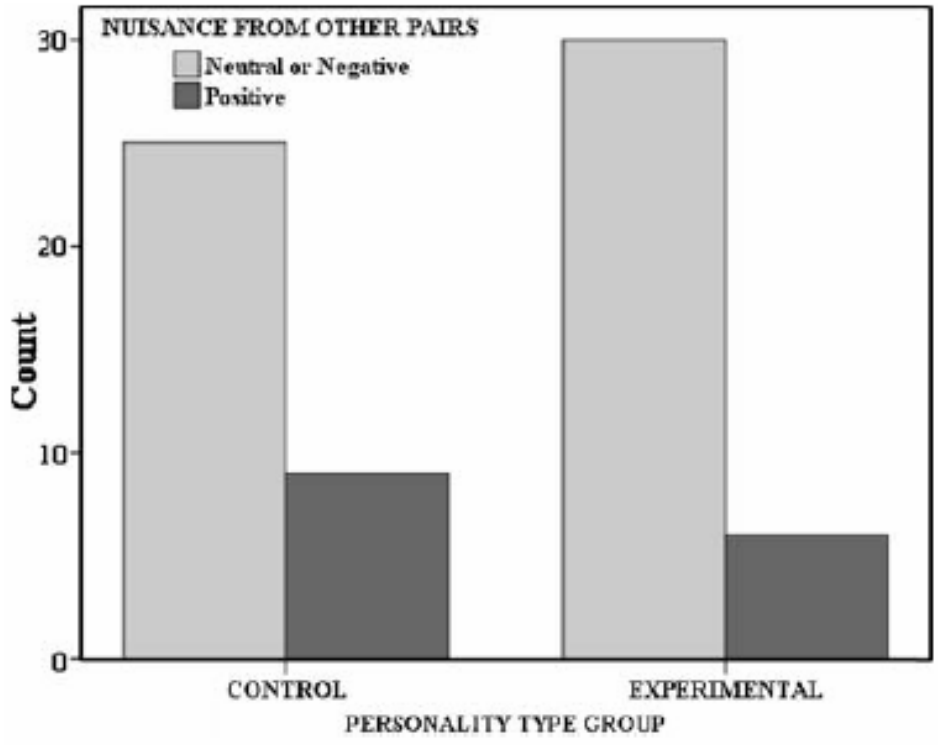
Pairs with heterogeneous developer personalities and temperaments perform better than pairs with homogeneous developer personalities and temperaments.



Panagiotis Sfetsos & Ioannis Stamelos & Lefteris Angelis & Ignatios Deligiannis. An experimental investigation of personality types impact on pair effectiveness in pair programming. Empir Software Eng (2009) 14:187–226.

Pairs with heterogeneous developer personalities and temperaments perform better than pairs with homogeneous developer personalities and temperaments.





Panagiotis Sfetsos & Ioannis Stamelos & Lefteris Angelis & Ignatios Deligiannis. An experimental investigation of personality types impact on pair effectiveness in pair programming. Empir Software Eng (2009) 14:187–226.

Pairs with heterogeneous developer personalities and temperaments perform better than pairs with homogeneous developer personalities and temperaments.

Personalities

Extroverting (E)–Introverting (I)

Extroverts get their energy from the outside world, from experiences and interactions, while Introverts from within themselves, from their internal thoughts, feelings, and reflections. Extroverts tend to talk easily about anything, whereas Introverts are comfortable with long silences. Introverts prefer finished ideas, namely to read and think about something before start talking, so we must give them time to process new information, especially in meetings. For interactions with users and management, Extroverts are better in talking (and getting responses), and at presenting ideas than Introverts.

Panagiotis Sfetsos & Ioannis Stamelos & Lefteris Angelis & Ignatios Deligiannis. An experimental investigation of personality types impact on pair effectiveness in pair programming. Empir Software Eng (2009) 14:187–226.

Pairs with heterogeneous developer personalities and temperaments perform better than pairs with homogeneous developer personalities and temperaments.

Personalities

Sensing (S)–Intuiting (N)

Sensors gather information linearly and tend to take things literally and sequentially, preferring tangible results clearly described. They are observant of what is happening around them, and are especially good at recognizing the practical realities of a situation. Their concentration to details makes them the most capable programmers. Intuitives like to gather information more abstractly (seeing the big picture), focusing on the relationships and connections between facts. They prefer to speak in concepts and global futuristic ideas and they are good at seeing new possibilities and different ways of doing things. Therefore, they can be used more as system analysts determining users/clients' needs and identifying problems in systems.

Panagiotis Sfetsos & Ioannis Stamelos & Lefteris Angelis & Ignatios Deligiannis. An experimental investigation of personality types impact on pair effectiveness in pair programming. Empir Software Eng (2009) 14:187–226.

Pairs with heterogeneous developer personalities and temperaments perform better than pairs with homogeneous developer personalities and temperaments.

Personalities

Thinking (T)–Feeling (F)

Thinkers make decisions based on objective information, while feelers are based on subjective information. Thinkers, tend to be logical, critical, orderly, and prefer to work with facts. They examine carefully cause and effect of a choice or action to enhance their problem-solving abilities. Their tendency to be logical is needed mostly in programming. Feelers, driven by personal values, make decisions considering what is important to themselves and others. Their strengths include understanding, appreciation and support of others. They tend to be more people-oriented and good team-builders.

Panagiotis Sfetsos & Ioannis Stamelos & Lefteris Angelis & Ignatios Deligiannis. An experimental investigation of personality types impact on pair effectiveness in pair programming. Empir Software Eng (2009) 14:187–226.

Pairs with heterogeneous developer personalities and temperaments perform better than pairs with homogeneous developer personalities and temperaments.

Personalities

Judging (J)–Perceiving (P)

The most problems concerning communication are caused by differences in the work styles. Judgers tend to live in an orderly and planned way, with detailed schedules. They like things decided and concluded and they prefer a bad decision than no decision. Perceivers tend to live in a flexible, spontaneous way, are based on experience and have open issues. They like to explore every possibility, and consequently have difficulty making decisions working on deadlines. On the contrary Judgers prefer to avoid last-minute stresses, and try to conclude quickly: once they have found a good solution they accept it and move on to something else. A combination of Judgers and Perceivers in a pair would help to ensure that the best solution will be found in a reasonable time.

Panagiotis Sfetsos & Ioannis Stamelos & Lefteris Angelis & Ignatios Deligiannis. An experimental investigation of personality types impact on pair effectiveness in pair programming. Empir Software Eng (2009) 14:187–226.

Pairs with heterogeneous developer personalities and temperaments perform better than pairs with homogeneous developer personalities and temperaments.

Temperaments

Artisans and guardians prefer concrete communications, while idealist and rational prefer more abstract communications. Artisan and rational types prefer a cooperative path to goal accomplishment, while guardian and idealist types prefer more a utilitarian approach.

Artisans, combining a sensor's sense of reality and a perceiver's spontaneity, feel most comfortable with clear and direct communications in cooperative pursuit of a goal. Possessing a correct sense of timing, they are great start-up persons, are lateral thinkers, problem solvers and great brainstormers. They are troubleshooters proposing practical solutions and making quick decisions. Their adaptability and sense of innovation can help the pair achieve requirements with new technology.

Panagiotis Sfetsos & Ioannis Stamelos & Lefteris Angelis & Ignatios Deligiannis. An experimental investigation of personality types impact on pair effectiveness in pair programming. Empir Software Eng (2009) 14:187–226.

Pairs with heterogeneous developer personalities and temperaments perform better than pairs with homogeneous developer personalities and temperaments.

Temperaments

Guardians, combining a sensor's sense of reality and a judger's desire for reaching a conclusion, are traditionalists and stabilizers. They are good as administrators, establishing policies, rules, schedules, regulations and a functioning hierarchy. However, they do not easily perceive project delays or problems. Because of a strong need for conclusions, they may not allow enough time for processing. They tend to want things to remain the same, without allowing for sparks of creativity or innovation. They will not be very successful in times of rapid change.

Panagiotis Sfetsos & Ioannis Stamelos & Lefteris Angelis & Ignatios Deligiannis. An experimental investigation of personality types impact on pair effectiveness in pair programming. Empir Software Eng (2009) 14:187–226.

Pairs with heterogeneous developer personalities and temperaments perform better than pairs with homogeneous developer personalities and temperaments.

Temperaments

Combining an intuitive's focus on the relationships and a feeler's preference in personal values, Idealists are successful in leveraging their own and other people skills. They like to guide others and are committed to the progress of the people in a given setting. They are also transactional as Artisans, excellent communicators and people motivators. An Idealist developer adds a personalised touch to a pair, contributing to team spirit and morale. However, he does not react well to sudden changes.

Panagiotis Sfetsos & Ioannis Stamelos & Lefteris Angelis & Ignatios Deligiannis. An experimental investigation of personality types impact on pair effectiveness in pair programming. Empir Software Eng (2009) 14:187–226.

Pairs with heterogeneous developer personalities and temperaments perform better than pairs with homogeneous developer personalities and temperaments.

Temperaments

Rationalists, combining the objectivity of the thinkers and the visionary of the intuitives, are described as theorists, innovators, systems planners and architects of change. They focus on competence and excellence, valuing high logic and reason, adopting a logical and strategic analysis approach of resolving issues. Thus their most practiced and successful operations tend to be planning, inventing and configuring.

Panagiotis Sfetsos & Ioannis Stamelos & Lefteris Angelis & Ignatios Deligiannis. An experimental investigation of personality types impact on pair effectiveness in pair programming. Empir Software Eng (2009) 14:187–226.

http://keirsey.com

Pair Programming

Wie gut funktioniert es?

Dybå, Tore, Arisholm, Erik, Sjøberg, Dag I. K., Hannay, Jo E., Shull, Forrest. **Are Two Heads Better than One?** On the Effectiveness of Pair Programming. IEEE Software, 2007, 24(6), pages 12—15.

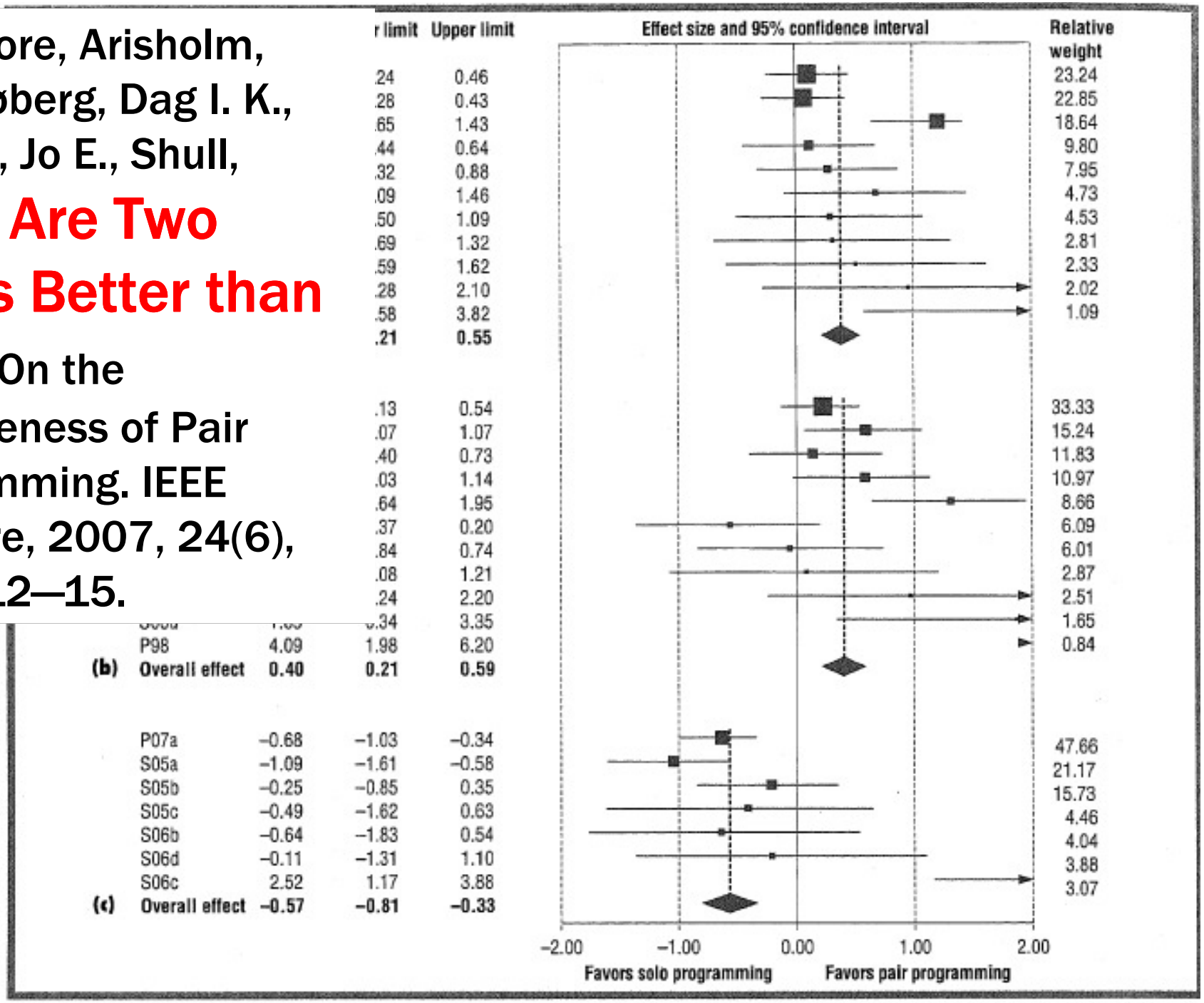
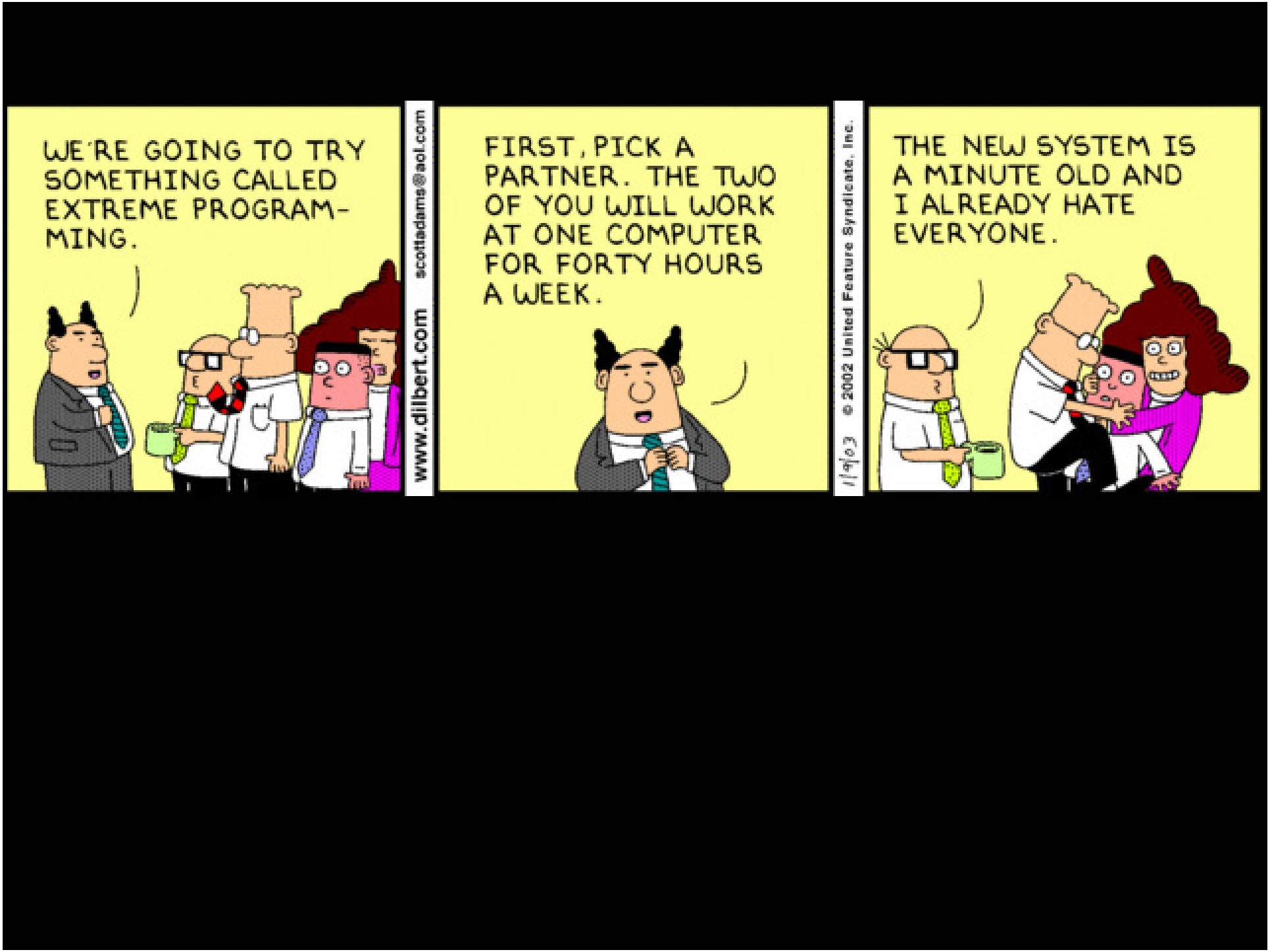


Figure I. Meta-analyses of pair programming's effects on (a) quality, (b) duration, and (c) effort.


WE'RE GOING TO TRY SOMETHING CALLED EXTREME PROGRAM-MING.
www.dilbert.com scottadams@aol.com
FIRST, PICK A PARTNER. THE TWO OF YOU WILL WORK AT ONE COMPUTER FOR FORTY HOURS A WEEK.
1/9/03 © 2002 United Feature Syndicate, Inc.
THE NEW SYSTEM IS A MINUTE OLD AND I ALREADY HATE EVERYONE.