View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by Bournemouth University Research Online

Helping programmers get what they want

Gail Ollis
Faculty of Science and Technology
Bournemouth University
gollis@bournemouth.ac.uk

Abstract

Notions of "good practice" exist for many aspects of a programmer's work. They are intended to bring benefits on a variety of dimensions from the most concrete and measurable, such as program speed, to more subjective characteristics such as readability. This research addresses programmer practices good or bad in any part of their professional work but considers them from the perspective of a single outcome: impact on the productivity of fellow programmers. The findings are now being applied in an experimental new framework for professional development.

1. Introduction

The inspiration for this research was the author's experience as a commercial programmer. It was evident that colleagues were more or less equally capable of creating a working program. People employed as programmers could, unsurprisingly, program; not necessarily at the same pace or with equal mistakes, but they could all get a computer to follow their instructions to produce the desired output. The differences between them were was not so much the success they had in producing machine behaviour but the degree of difficulty for a human reader in understanding the code that brought it about.

Programming work more often involves wrangling existing code than starting with a blank slate (Verhoef, 2000). Even a brand new project written from scratch - a pristine greenfield site upon which software can be built without historic constraints - soon becomes constrained as new decisions are made about tools, interfaces, standards, system partitioning and so on. Time is also a constraint; project managers are reluctant to permit rework when poor programming or infrastructure decisions become apparent. A system can easily become the sprawling, unstructured development that Foote and Yoder (1997) describe as a "big ball of mud". A comment from a participant in the study recounted in section 2 illustrates just how rapidly this can happen: "I saw him write a legacy system in three weeks".

In the author's experience of working on existing code two patterns repeatedly emerged. In one pattern, making the necessary changes felt like following an obvious route made straightforward by the careful planning and signposting done by a predecessor. In the other it resembled struggling through a maze, encountering numerous dead ends and seeing the same features over and over again, sometimes only to discover that it wasn't even the right maze. Given enough time at the same workplace programmer "fingerprints" became evident in both patterns, often to the point of correctly guessing "this looks like X's code".

The research began with curiosity about why these differences exist in people with similar qualifications, experience and even theoretical grasp of "good practice". Many of those whose work had been most impenetrable to the author were vocal advocates of entirely respectable, mainstream teachings. They were not ignorant of programming axioms, but maladroit in applying them. Some advice, such as adding comments for things "that are not obvious" (Hunt & Thomas, 2000, p.249), is easy enough to cite as a principle but demands a deeper empathy and understanding to put it into practice effectively.

Formally researching the question demanded first a clear impression from a wide cross-section of programmers about what matters in peer behaviour. The study which elicited this information is described in section 2. Its findings invited a new focus for the research (section 3): applying the methods used and important topics discussed to facilitate professional awareness of peer impact issues.

2. Initial study: What programmers want

One programmer's experiences are insufficient foundation for research. To identify issues on which there is a broader consensus, an exploratory study collected the opinions of other experienced professional programmers.

2.1. Data collection interviews

A total of 28 participants from 7 companies across a variety of application domains were interviewed one-to-one so that they could speak freely in the absence of colleagues. It was important for both the ethical standards of the study and the willingness of companies to take part in it that individual opinions remained anonymous and commercial confidentiality was respected.

The participants were sufficiently experienced that any problems they reported were not the transitory difficulties of a novice. Each had been employed in the industry for at least 5 years, a level of experience at which managers are willing to commit customer-critical and mentoring tasks to developers (Zhou & Mockus, 2010). The goal was to find which peer behaviours have most impact (positive or negative) on professional programmers at this level in their day-to-day work.

The scope was intentionally broad so as not to pre-judge the outcome by focusing solely on, for example, existing notions of good practice. These can be "good" for a variety of reasons but the sole criterion of interest was the effect a practice has on colleagues; this criterion had to be carefully explained because participants came from a population more accustomed to discussing technical measures of impact.

Faced with a question about how the actions of peers (whether predecessors or current team colleagues) affect the difficulty of a task, programmers easily recalled a few recent or particularly significant events such as memorably bizarre variable names; recent events are usually more accessible than older ones (Baddeley, Eysenck, & Anderson, 2015, pp.51) and autobiographical recollection is also better for emotional or unusual incidents (Bower & Forgas, 2000, p.92). Incidents which were more mundane or about auxiliary activities such as writing tests or bug reports rather than code were less freely recalled. Interviews therefore started with an open question but followed it up with a card sorting exercise, a form of cued recall (Baddeley et al., 2015, p.304) to prompt participants to consider all the elements of their work.

The cards covered categories such as testing, bug reporting and interpersonal interactions as well as coding. Each card described a design decision or style of behaviour drawn from texts on good practice (Hunt & Thomas, 2000; Henney, 2010), from personal experience, or from Henney and other textbook authors' Twitter complaints about contemporary events, such as this:

> *Suffering the tyranny of the "TODO" comment. People: "do" your "todo"s. (Goodliffe, 2012)*

A draft set of cards was piloted with 3 experienced programmers to check that the process and content made sense and to determine the amount of time needed for the exercise. As a result an initial set of 88 was reduced to 54 cards by consolidating onto a single card wherever very similar content had elicited the same response. This set can be sorted within 20 to 30 minutes.

Participants sorted the cards on a 5-point scale according to how much their own work is impacted by the peer action:

- **Bad. Noticeable impact.** Makes my job harder/slower.
- **Bad. Slight impact.** Makes my job a little harder/slower.
- **Neutral.** Does not much affect my own tasks.
- **Good. Slight impact.** Makes my job a little quicker/easier.
- **Good. Noticeable impact.** Makes my job quicker/easier.

This was not intended to make a quantitative measurement cataloguing the topics in order of impact but rather to help participants assess the relative impact of peer behaviour and cue them to consider the whole breadth of activities, not just coding, that their work entails. The discussion then continued, talking about any new examples that had come to mind during the exercise and exploring the reasons for rating behaviours as ones that have good or bad 'noticeable impact'.

2.2. Analysis

The purpose of the analysis was to identify common themes in the interview data, addressing the research question by looking for areas of consensus among programmers about what matters in peer behaviour. A template analysis (King, 2012) was used because this method flexibly accommodates both an open discussion in which participants can raise any topic, and topics anticipated prior to the study (a priori themes).

The a priori themes used in the initial coding of the data were the card topics. Such codes are provisional; as each of an initial sub-set of transcripts was examined, the coding was refined to reflect participants' own take on the cards and new codes were added for relevant material that did not fit an existing theme. Once the number of distinctly different new themes tailed off (this occured after 3 transcripts), the codes were organised hierarchically under a smaller number of higher order codes describing broader themes to create the *initial template*.

The template was then developed by applying it to remaining transcripts and modifying when necessary: occasionally new topics were added, but changes were mostly refinements to existing codes (e.g. refining the description of a priori codes to reflect the scope of discussion that the related cards inspired). This process continued until, after eight transcripts, no further new and relevant information was emerging (i.e. saturation was reached). The template at the end of this process became the *final template* that informs the interpretation below. The remainder of the recordings were reviewed in audio format to identify particularly illustrative comments and check that they did not throw up common topics not covered by the template.

2.3. Findings

2.3.1. Overview

There were three parts to each interview: the initial part in which participants spontaneously recalled their experiences; the card sorting exercise in which they rated the impact of peer behaviours; and finally, a discussion structured by the cards rated to be in the most significant impact categories.

Feedback from participants suggested that the cards achieved the goal of presenting quite a comprehensive picture of a programmer's work. Thus many of the spontaneous contributions naturally pre-empted card topics, and none of the themes explored below were drawn solely from discussions of the cards. When spontaneous topics *not* anticipated by the cards came up they were typically specific to the participant's role. Sometimes they reflected team leader concerns rather than a programmer perspective as sought by the research question - although still identifying the same kind of issues. Other topics such as product deployment and support were relevant but not common because they depended on the nature of the business and the participant's role within it.

2.3.2. Fine-grained code detail issues: the importance of naming

Fine-grained details about usage of programming language features rarely excited strong enough feelings to arise as spontaneous topics. This characteristic was also reflected in the card sort. While detailed code topics represented 50% of the cards, only two were rated as examples of 'noticeable impact' by the majority of participants: "Uses code comments in ways that aid understanding" and "Chooses identifiers which are not succinct, meaningful and distinct". Both of these feature among the topics which were raised spontaneously.

Although participants agreed that good comments have significant positive impact there was a distinct absence of consensus in their detailed accounts of this topic. Opinions ranged from a programmer who felt that any comment has potential use as a clue to the code's purpose, even if not completely accurate, to

the programmer who sets syntax highlighting to render comments in the same colour as the background, and thus invisible. Comments were the only divisive topic. Differences were in the perceived scale of the impact or the contingent nature of some responses. For example the boy scout rule (Martin, 2010), represented by the card "Tries to leave a module a bit better than when they checked it out", emerged as a good ideal to have in theory but not always a desirable behaviour in practice, depending on the context.

The other significant topic at the fine-grained level of code detail, choice of identifiers, had no such equivocation. Most cards at the "lines of code" level were rated as having *some* impact, but for the majority of participants it was only a slight impact. Syntactically correct code, however ill-written, is unambiguous and it would seem that these experienced programmers are adept at reading it; they recognise its shortcomings but these are not a particularly significant hindrance. However, no amount of experience can make meaningless identifiers meaningful. This participant's recollection of the frustration they cause was a typical example:

> *Choosing vegetable names for variables... years ago there was somebody at another company that thought it was humorous. (Programmer B)*

Commenting and naming aside, axioms of coding practice were not the dominant factors. This revealed that the topics of most concern to the participants lay at a higher lever of granularity than understanding lines of code; the challenge is to find the appropriate lines to look at in the first place.

2.3.3. The importance of communication for navigable code

In discussing the problems of navigating their way to the right lines of code, participants spoke of the value of early and frequent communication between a programmer and their peers as an ongoing and informal review process:

> *You always need to be listening to and talking to other people in the same team to make sure that you're all on the same track. (Programmer A)*

Such communication allows other team members to give timely feedback on approaches which make structure hard for others to make sense of, such as inventing a new and therefore unfamiliar way of doing things, using something that is unsuitable or unnecessary (e.g. a new and interesting framework) or simply writing code that is not needed. The feedback is essential:

> *To my mind development of computer systems is always a team oriented activity. There's very few people who will sit down by themselves in a dark room and produce a good, useful and maintainable system. (Programmer B)*

The impact of software created in the daylight by an open-minded creator constrasts sharply with that of idisosyncratic software produced by someone ploughing their own furrow:

> *Immediately understand what they're doing... without having to sort of wonder why you've come up with some other alternative. So that kind of thing, that saves time I think. (Programmer C)*

> *The more you dig deeper, the more you get exasperated by it. Because you're thinking "what are, what have you done here? what, what monster have you created?" (Programmer D)*

Along with some technical skill to write code that is not too complex, social conduct is important. Whether a peer 'works in isolation' or 'asks questions' emerged as two sides of the same coin for the impact they have. It is not obvious that quietly getting on with the job alone can adversely affect anyone else nor that asking questions of colleagues is of particular benefit to any but the questioner, but they play a key role in creating a solution that works, fits in and makes sense.

Productive communication also calls for an open attitude and an understanding that code belongs to more than just an individual:

> *You say "I don't understand this bit of code" and their response basically is "well that's because you're an idiot". And by the time six people have knocked on the door and said "I don't understand this bit of code" you really think they should start getting the hint that it doesn't matter how clever they are compared to the rest of humanity, they've gotta write for their audience. (Programmer G)*

> *They build an attachment to what they themselves have produced and a particular way of doing things. (Programmer H)*

Having an open attitude includes being able to participate in a mature discussion of a question without giving or taking offence:

> *There's someone in the current role who is very stubborn and that's not helpful if you're trying to come to something together. So he has this ability to make himself come across as affronted by any questioning. If you can do it in a way where you can separate the person from what you're talking about you can have a fantastic discussion and we actually promote that, that's fantastic. (Programmer E)*

The phrase "trying to come to something together" summarises the essence of participants' talk about fruitful technical dialogue: it is not about ego or point scoring but moving forward with an appropriate technical solution and a better understanding. Programming is an exercise in problem solving.

3. An experimental framework for professional awareness of peer impact (work in progress)

Outline plans for the next step in the research after the exploratory study originally proposed setting code modification tasks; these would focus on decision making in aspects of coding practice that emerged as important. The plans were set aside because themes concerning coding syntax were *not* the ones that had most impact on participants.

What did emerge were issues with attitude and culture, as briefly summarised in section 2. It also became evident that the methods used in the interviews had possibilities as a tool to facilitate discussion of these unstated and potentially sensitive issues. Two things suggested this. Firstly, participants frequently seemed to enjoy the card sorting process and many said as much. Secondly, one participant asked to take a set of cards to discuss in one of his team's regular lunchtime seminars. Afterwards he reported:

> *I suggested we lay out the cards, and each pick one or two that made us think 'oh yeah, that reminds me of...' and tell the story. Which went pretty well, with one or two very gentle nudges during the course of it, away from 'I think this is good practice' towards 'this is how it affected me on this occasion'. I found that really helped allow everyone to contribute, as well, including the less experienced and quieter people- it's less intimidating to tell a story of your experience (others can't argue that you're wrong if you're only reporting your experience) rather than nailing your colours to a particular good/bad practice mast.*

His approach, akin to a focus group used to elicit all members' opinions, inspired a way to apply the findings. Work is now in progress to develop this approach as framework for professional development workshops. Once it has been piloted it will be tested first at some of the original participating companies and then at some new ones. The conjecture is that the opportunity to pick topics already written on the cards offers a defusing effect which enables them to be raised more easily.

The cards will be revised to contain only common themes from the analysis of the interviews; group members will know that the cards reflect the consensus of very experienced programmers. Being offered

for discussion as materials in an independently designed workshop structure, topics are less likely to seem so personal and cannot reasonably be dismissed as "just your opinion". The opinion of a team member should not be so lightly dismissed anyway, but the knowledge that it is widely shared may help to discourage some from doing so and help others to share their views.

As well as encouraging little-discussed topics to be raised in a gentle manner, the workshop structure also allows the discussion to have local relevance. Selecting their own topics from the range available should allow teams to derive maximum benefit from the process by focusing on specific issues that affect them rather than, say, receiving a general presentation of the issues identified by the research. If the process is to bring about change it helps for the participants to have ownership of the process (O'Driscoll, Pierce, & Coghlan, 2006), so one goal of developing and studying the workshop structure is to see whether the discussion can be conducted without an external facilitator.

To evaluate the workshop procedure two distinct enquiries will be made in a questionnaire immediately afterwards: what participants thought of the process itself, and what changes they expect or hope the experience will bring about. A survey of team behaviours will be administered approximately a week beforehand to provide a snapshot of the team's current state since the conduct of the workshop and its outcome are a function not only of the workshop itself but the context in which it is applied. This survey will be repeated 4-6 weeks later to see if any change can be observed, although this would represent a rather impressive effect. To assess whether small changes have occured, participants will also be asked specifically whether they have observed any effects that they attribute to the workshop.

4. Conclusion

The participants in the initial study spoke freely about colleague behaviour in a way that is not easily achieved in the presence of colleagues without risk of disagreement, offence or embarrassment. It may be possible to have such a conversation outside the workplace but the interviews offered an unusual opportunity to do so confidentially with someone who understands the field well and is genuinely interested. They responded to the opportunity with honest reflections and insights. It was apparent from their parting words that they also found the process interesting and enlightening.

While it is impossible to create quite the same environment for a group, the workshops will allow others to benefit from the insights that the interviews helped to draw out. This research has captured a perspective that is often overlooked among more easily measured indicators of good practice. Applying it requires more than a definition of principles to follow; just as with programming axioms, effective application depends not on declarative knowledge but a deeper understanding. By facilitating communication about the impact of behaviour, the ongoing work aims to create just that.

5. Acknowledgements

Thank you to Sherry Jeary, Keith Phalp, Jacqui Taylor and Kevin Thomas for their sage advice throughout this research and for kindly reviewing this account of it. Thanks also to the PPIG reviewers, who not only improved this paper but also generously gave advice that will be invaluable in writing up the results of the research.

6. References

Baddeley, A. D., Eysenck, M. W., & Anderson, M. C. (2015). *Memory*. Hove, East Sussex; New York: Psychology Press.

Bower, G. H., & Forgas, J. P. (2000). Affect, memory, and social cognition. In E. Eich, J. F. Kihlstrom, G. H. Bower, J. P. Forgas, & P. M. Niedenthal (Eds.), *Cognition and emotion.* Oxford; New York: Oxford University Press.

Foote, B., & Yoder, J. (1997). Big ball of mud. *Pattern languages of program design*, *4*, 654–692.

Goodliffe, P. (2012, June). *Suffering the tyranny of the "TODO" comment. People: "do" your "todo"s.* [microblog]. Retrieved 2015-01-17, from `https://twitter.com/petegoodliffe/status/217622701605535744`

Henney, K. (Ed.). (2010). *97 things every programmer should know : collective wisdom from the experts*. Sebastopol, Calif. : O'Reilly Media.

Hunt, A., & Thomas, D. (2000). *The pragmatic programmer : from journeyman to master*. Reading, Mass: Addison-Wesley.

King, N. (2012). Doing template analysis. In G. Symon & C. Cassell (Eds.), *Qualitative organizational research : core methods and current challenges.* London: SAGE.

Martin, R. C. (2010). The boy scout rule. In K. Henney (Ed.), *97 things every programmer should know : collective wisdom from the experts.* Sebastopol, Calif. : O'Reilly Media.

O'Driscoll, M. P., Pierce, J. L., & Coghlan, A.-M. (2006, June). The Psychology of Ownership. *Group & Organization Management*, *31*(3), 388.

Verhoef, C. (2000). How to implement the future? In *Proceedings of the 26th Euromicro Conference, 2000* (Vol. 1, pp. 32–47). IEEE.

Zhou, M., & Mockus, A. (2010). Developer Fluency: Achieving True Mastery in Software Projects. In *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering* (pp. 137–146). New York: ACM. doi: 10.1145/1882291.1882313