

Process Patterns for Personal Practice

How to succeed in development without really trying

© Charles Weir, James Noble, version of 04/28/99 10:54 AM from Charles Weir

Introduction

There is more to software development than analysis, programming, debugging, and even organising organic organisational structures. Whether working solo, in small groups, or larger teams, the biggest problems can be our own personal misconceptions, demons, paranoia and firmly held beliefs.

Development is always done by people; clients and users are people; and under strict genetic testing most managers can be shown to share at least 50% of their genetic code with *homo montipythonus*.

People, of course, are famously fallible, when not downright stupid or obstructive¹. It's a miracle anyone manages to get anything developed ever. Of course, people do develop software every day, after many years of rigorous training and then by dint of weeks of hard work, bad skin tone, low lighting conditions, and the consumption of mind-altering quantities of pizza. Now, this is obviously far too much work, not very much fun, and bad for your physical and mental health. *So, how can you succeed in development without really trying?*

Overview

In this paper we describe what we call *Process Patterns for Personal Practice*. Each of these patterns describes a *personal process practice* that is, something which you can actually *do* (alone, or in a group) which might help your software development — or at least help you to feel better about your software development. These patterns do not attempt to describe novel techniques; rather they describe techniques and strategies that are well known in the industry, such as brainstorming, using consultants, focusing on solutions, and talking to people. The patterns should be useful for developers, analysts, and project managers who are working on or mangling real world projects for the first time.

We can't promise that these patterns will let you succeed in software, or improve your skin tone. They might improve your mental health, and that of the people who have to put up with you at work.

These patterns differ from the rest of the patterns literature is that they describe practices or actions, which have to be enacted in order for them to be useful. This is in contrast to the vast majority of the patterns literature which describe concrete solution *structures* — either in software, organisations, or processes — which can be built or embodied as a result of using the pattern. Design Patterns [Gamma], for one example, create concrete artifacts (software designs), but other kinds of patterns also describe concrete structures. Jim Coplien's Development-Process Patterns [Coplien] describe specific organisational structures (such as arrangements of roles in a development group) and Kent Back's Smalltalk Best Practice Patterns [Beck] provide advice (such as to use function names which mean something) which can be made manifest in a program text. In contrast, these personal process practice patterns produce no concrete results — no structures or products or deliverables. Rather, they describe actions you can take or tasks you can perform.

The Pattern Form

The patterns are written in a condensed pattern form, so that we can succeed in pattern writing without really trying. As in Christopher Alexander's pattern format [Alexander], each pattern highlights and discusses a problem, then concludes: '**Therefore:**'. This is followed by a specific recommendation and a discussion of how to implement this recommendation. Further sections then describe consequences, known uses', and related patterns and literature. Throughout the pattern descriptions we refer to other patterns using **pattern names in bold**. Perhaps the most vital discussion for a pattern is of the *forces*, or considerations and consequences to help readers to decide when to use that pattern rather than another. We've highlighted the main forces in the problem selection by printing them in *italics* (we also use italics for emphasis). Then in the Consequences section, we identify other the other implications, both positive and negative. To distinguish the positive from the negative consequences of

¹ Excepting, of course, yourself, ourselves and your friends and ours!

each pattern, we describe the positive consequences first, the negative ones second, and partition the two with the word '**However:**'.

Blame

This paper is the result of a friendly wager with Alistair Cockburn. It was about 12:30 am in the Ritterstube on the last day of EuroPLoP'98, and Alistair had lost several games of Patterns Mornington Crescent. To change the topic, Alistair suggested that patterns that described processes were bound to fail, while patterns that described concrete solutions were much more likely to succeed. We disagreed, and this paper is a practical result of that challenge.

Linda Rising has been our shepherd.

We have not wanted for guidance and helpful comments.

*She has lead us into the green pastures of correcet spelling,
and the quiet waters of irony.*



The authors would like to thank Klosterbrau Irsee Breweries and Williams Peach Schnapps for their inspiration.

Pattern: Five Minute Scream

What should you do when you've been working on a problem ... and you're stuck?

You are working on a programming project or some other *technically demanding task*. There may be *other people around* – some that are more experienced, some that are less experienced, some haven't a clue about what you are working on.

Programming, or other computer tasks, can be very hard. Often, success depends critically upon *arcane knowledge* that most people just don't have, and what's worse, that they don't even realise they don't have, and don't even realise exists. Alternatively, many programming problems are just exercises in trivia, requiring mastery of a *thousand fine details* rather than any deep knowledge or insight. Maintenance or hacking is often impossible other than by trial and error.

One approach is to continually bother other people when you're making no progress – just ask someone else for help. This has the advantages that (if they can help you) you will get your work done more quickly, but it often has the disadvantage of preventing them getting on with their own work.

At the other extreme, you can continue to soldier on, making little or no progress – or worse, having the illusion of progress for an hour or two, followed by hours more of desolation when it doesn't work out.

For example, you might be unfortunate enough to be making hacks to a rather old source release version of the Java Virtual Machine. You keep running long tests with tracing enabled, because you are trying to work out what is wrong with your elegant, stylish, and hopefully minor hacks, which just happen to keep crashing the system. Nothing you can do seems to fix things.

Therefore:

Work for five minutes, then Scream.

Walk away. Talk to somebody about it. Get out of the office (or cubicle) and have a drink or a bite to eat. Work on something different for a bit. Surf the Web.

Just don't keep beating your head against that problem brick wall



Consequences

You don't get your work done, but you *feel a bit better*. For a few minutes.

When you eventually return to the problem you may have a *different outlook* – often that's all you need to solve it. You may have *new ideas*. Whomever you 'scream' to may have ideas of how to solve it. The mere act of explanation may help you solve it (see **Cardboard Consultant**).

However:

If you *interrupt* other people you may be preventing them getting on with their work (though this effect is usually small).

Some *work cultures* discourage people from interrupting or interacting with others except through formal channels. Interrupting other people (and letting them know you're not making progress) can *lower your status* within the dysfunctional organisation from which you receive your paycheck.

Examples

James says: That JVM I was hacking was crashing continually, but only after a reasonably long period of computation. After rather more than the canonical five minutes, I talked to David Holmes who suggested turning on the verbosegc flag. Eventually, this indicated that the program crashed just after the garbage collector automatically reclaimed and unloaded just one dynamically loaded class. From here, it was easy to use the debugger to find the problem, and trivial to fix it.

Another time, I noticed that the test cases of the JVM seemed to be crashing just after the next version finished compiling. After futzing about for five minutes trying to replicate this, with mixed results. I screamed (implemented by walking to the office of the local Solaris expert), who explained, that, yes, shared executables were supposed to be shared, so recompiling them and replacing new information of the top of them would probably cause all the old programs (the test cases) to crash.

Charles says: Many effective cultures encourage people to talk to each other when they're stuck. Among Symbian's EPOC development teams it is always acceptable to interrupt someone else when you've a problem. One successful Chicago options trading firm discourages even having project deadlines, since deadlines force people to work on their own work in preference to helping others.

See Also

Alistair Cockburn's "**Expert within Earshot**" is in some sense a structural counterpart to this pattern. **Expert within Earshot** describes how to structure a development organisation so that there is likely to be a **Consultant** around when you make your **Five-Minute Scream**.

The eXtreme Programming practice **Programming in Pairs** is also quite close to **Five-Minute Scream** (and **Expert within Earshot**). When working in pairs, you always have someone to whom you can scream. When your pair gets stuck, you can collectively scream, either to a single member of the development group, or involve the whole group in **Brainstorming**.

Pattern: Cardboard Consultant

AKA: Discuss it with your spouse. Tell it to your dog

What should you do when you've been working on a problem ... and you're stuck?

You've tried everything you can think of and you've *not found the solution*. In fact, you seem to be just *moving in circles*.

Maybe you've tried a **Five Minute Scream** several times, and while you're contemplating surgery for your vocal chords, you're still *moving in circles*, and you're still stuck.

Maybe someone else might have been able to help, but either they've all gone home or they're all too busy at the moment. Or maybe there's *nobody around* right now who knows anything about this kind of problem. But maybe there's *nobody in the world* but you who knows enough about this particular problem to say anything useful about it.

So you're still stuck.

Therefore:

Explain the problem out loud to someone or something.

Explaining your problem in detail can be very valuable, even if there's little chance that the 'listener' will be able to find a solution. Your listener may be human, animal, plant or artifact. Some people like to explain things to their dog, or to a character in a picture. Others tell their spouse or a passing friend. Describe the problem in detail, assuming an intelligent listener with only a basic background in the problem domain. Or alternatively, write down the problem and the things you've already tried — say for readers on the Usenet or on an electronic conferencing service.

Consequences:

The process of explaining the problem forces you to look at the issues from a *different point of view*. You'll have to:

- Explain your basic assumptions.
- Explain the chain of logic that has led to your being stuck.
- State each implicit conclusion you've come to.

Typically one of these generates ideas for new approaches to the problem, or suggests one or more 'obvious' misconception in your logic. Some further work and the problem is solved.

So you *solve the problem* and can get on with the next thing. Or at least you've thought of some profitable lines of enquiry. Even if the process doesn't provide a solution, at least you've got to take a break from the frustration of having no ideas.



You haven't wasted any *valuable time* from other experts in your particular field.

However:

You can *look a right prat*² explaining stuff to your dog. Or if you explain everything to a person (maybe your spouse), they may be bored and exact *retribution* in some form - such as explaining their own problems back!

Some people, particularly the *unimaginative* or verbally challenged, may not benefit from the process. Others (particularly those most 'upwardly motivated') may be *unwilling to explain* a problem to someone or something they don't respect.

Variant: Lab Notebook

A more formal approach is the Lab Notebook. Rather than explain the problem out loud, write down the problem in a lab notebook. Start with a section titled "Problem", then have a number of sections starting "Hypothesis", "Test", "Results". The "Problem" section describes the problem; each "Hypothesis" section describes a possible explanation or solution to the problem; "Test" describes a way to test the hypothesis, and "Result" describes the result of the test once you've done it.

It's often easy to write down a lot of hypotheses, then to add their corresponding tests, and only to do the tests afterwards. In that way sometimes you can combine several tests in one.

Be sure to start from scratch when describing the problem. It's better to write down "Why does the program crash?" rather than "What is wrong with the Networking Component" – unless you're sure that the problem is in the networking component. There's a section in the novel "Zen and the Art of Motorcycle Maintenance", which describes the Lab Notebook procedure very well.

This approach is pretty much guaranteed to solve any (solvable) problem. Because you're writing things down, if you're still stuck later then other people may check your thinking and perhaps provide suggestions.

However:

This approach is very expensive in terms of time. To use it effectively, you must write down *everything*, to avoid the risk of missing a vital statement. So for most real-life problems it's easier to use a less formal technique.

Variant: Real Consultant

A common variant for large-scale problem affecting more than one person is to employ an external consultant. The main qualifications for an effective consultant are:

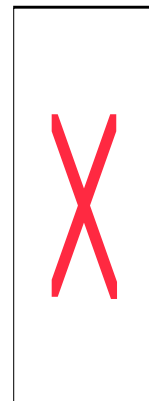
- They must be *skilled at listening* effectively and at communicating what they've learned back to the group. Thus they can challenge the implicit assumptions and chains of logic that led to the problem.
- They must *command respect* from the people involved in the problem. For this reason (if for no other), consultants usually charge significant fees for their services.
- They must *understand enough* of the subject area of the problem to be able to understand the problem and to reflect conclusions to the rest of the group, so, consultants are usually specialists in their chosen fields,

A good consultant will consolidate ideas, help find solutions, and introduce new ideas and techniques into the team.

However:

Consultants tend to be *expensive* compared with equally qualified staff working for your own organization. *Poor consultants* can waste significant amounts of time for the group without achieving anything. There's usually significant *administrative and political overhead* to employing a consultant.

Examples:



² *prat*. n. (UK). fool. from *prate* v, to explain something to your dog.

Cardboard Consultant

Charles says: I frequently discuss office, management and occasionally technical problems with my partner. I find the process of explanation almost always suggests solutions - if she hasn't already suggested one!

When faced with intractable debugging problems, I sometimes describe them in a message for an electronic bulletin board (or Lotus Notes or Usenet) used by other programmers. More often than not I find I've thought of several solutions before I complete the description – and I never post the message.

I have several times found myself asked to review programs in computer languages that I'm not familiar with myself, nor have ever claimed to be. In such a situation, it's remarkably effective to ask the programmer to explain what each section does, and to ask questions along the lines of "... and what does *that* do" during the explanation. More often than not, the programmers will find their own mistakes during this process.

Linda Rising says this pattern got her through grad school. "My advisor was very difficult to track down, so I always worked through tough problems with the help of my husband. Since he didn't understand a lot of the problem, he would always ask "off the wall" questions. This always exasperated me but invariably as I would patiently try to simplify my explanation, I would see the solution. I suspected him of playing "dumb" just to bring out a resolution. I now use this pattern routinely — poor guy!"

Alistair Cockburn says: Where I am working at the moment, Bob told the story of the programming wizard at their old place who put a stuffed bear on his table. When people came in for advice, Wiz would say, without looking up, "Explain your problem to the bear first." Bob then recalled seeing that Wiz, in earlier years, had always had a small stuffed bear place on top of his monitor. Only recently had Bob come to understand why this bear was always there [k1].

Freya North's book "Chloë" describes a delightful (if fictional) use of this technique. Chloe the heroine describes her problems to two figures in a copy of a Gainsborough portrait she carries with her. In her imagination they reply and help her.

Lab Notebook

I sometimes use the "Lab Notebook" myself to tackle particularly difficult program bugs. When I'm debugging, my notebooks tend to be full of "Problem:" and "Hypothesis:" statements. Few if any of them ever get completed with: "Result: yes, that was the problem"; it's not necessary.

Robert Ward describes using a Lab Notebook for debugging in his 'Scientific Debugging' presentation.

Robert Pursig's character 'Phaedrus' uses the Lab Notebook to solve motorcycle maintenance problems.

Lab Notebooks are a standard tool for Physics and Biology researchers investigating unexplained phenomena.

Real Consultant

As consultants, we [k2]often encounter situations where the group employing us have the information required for a solution, but need the facilitation skills of an external agent.

Charles says: One typical example recently involved an engineering company faced with a pilot project done by a 'tiger team' working independently of the mainstream development. The team leaders in the rest of the development group were concerned about the apparent lack of progress or benefit to the company from the pilot project. As an external agent, I was able to bring the people concerned together, to help the tiger team leader to explain his schedules and deliverables in terms the rest of the team understood. I was also able to reassure the management that the pilot project was no more off schedule than reasonable for such a task (!).

Note that the consultant (Charles) brought little new information; even the knowledge of typical schedules for such pilot projects was already available to the managers. The consultant's communication skills and authority solved the problem of anxiety in the rest of the development group, which was threatening to upset the pilot project.

Gerald (Gerry) Weinberg's book "Secrets of consulting" describes other examples of this kind of consultancy from his own experience.

See also:

Five Minute Scream can be helpful when there are people around who are familiar with your problem; **Brainstorming** is an alternative to get help from several people.

There are many books describing how to be a consultant, or to use consultants in a business³. Two of the best are Gerald Weinberg's "Secrets of Consulting", a comprehensive and readable discussion of the techniques and psychology of consulting, and "Bluff your way in Consultancy" [Viney] which is short and frivolous but contains valuable information.

Robert Ward's presentation also describes further debugging techniques.

Pattern: Brainstorming^[k3]

How can you collect a bunch of possible solutions for a difficult problem?

You're working on a problem that *affects several people*. You, or maybe others in the team, have explored all the obvious solutions and you don't appear to be getting anywhere.

Perhaps people have suggested some possible solutions, but all the *suggestions have been discarded* as unworkable, irrelevant or politically unacceptable (not invented here, or we tried that ten years ago and it didn't work then). So, far from it being a benefit to have lots of people working on it, team working has tended to make it *more difficult* to come to a solution.



Therefore:

Get the team members together for a **brainstorming session**⁴.



Brainstorming is 'A concerted attack on a problem usually by amassing a collection of similar ideas, which are then discussed' [OED]

The techniques for running a brainstorming session are now well documented [Buzan, de Bono]. Briefly:

- Identify and appoint a facilitator to run the meeting. It is best if someone skilled at the task takes this role.
- Appoint a record taker. Ideally there are two: one who will write ideas up for everyone to see, another who takes down an electronic version. 
- Get a representative collection of those concerned to solve the problem together for a single session. The time required depends on the nature of the problem.
- Working as a group, identify exactly what the problem is. 
- Identify ideas – but without criticizing them in any way. The facilitator must ensure that:
 1. Everyone gets to make contributions rather than certain characters dominating the meeting.
 2. Negative comments on an idea are discouraged.

Consequences

Because no holds are barred, *incomplete suggestions* don't get crushed because of details that are unworkable. Even the silliest ideas may prompt some reasonable suggestion from the group.

Even people with *no technical knowledge* of the problem area may contribute to solving the problem – indeed it's a good idea to have at least one such person in the brainstorming meeting.

Because no holds are barred, the normal *inhibitions* that cause you to reject possible approaches don't apply. You end up with a list of possible approaches; one of which may solve the problem.

Brainstorming meetings are often *good fun* for all concerned!

However:

³ And in most major cities you can find churches of Satanism, and even IBM offices – places where you can sell your soul to complete the secret ritual and become a consultant.

⁴ Strictly this should be written 'brain-storm', but common usage now combines the two words.

The meeting is *costly* in people's time. It may be difficult or expensive to find a suitable *facilitator*. Badly managed brainstorming meetings can *create tensions* in the team if people's ideas are dismissed or people feel insulted or crushed

Some people may resist the *informality* of the brainstorming approach, describing it as *pointless* or *childish*, similarly, brainstorming may be unacceptable in organisations with very *formal cultures*. It may take considerable effort from the *facilitator* to get their involvement in the process.

Variant: Solo Brainstorming

It's possible, if less effective, to do the brainstorming operation on one's own. Most of the same principles apply; in particular of not criticizing your ideas, and of taking an informal approach. There are even commercial tools are even available to help. Try a web search for the word 'brainstorming'.

Tony Buzan's book 'Use your head' describes many techniques for this kind of solo idea generation.

Variant: Don't Shoot the Messenger

At any time, if someone tells you an *idea that sounds silly*, it can be remarkably effective to use the approach from brainstorming session. Rather than reject the idea, explore it together. Open questions are particularly useful for this: "What do you mean by that?", "How can we take that idea further", etc.



Charles says: More than once recently a silly idea suggested to me has eventually led to a workable, or even excellent, solution to a problem. Or, equally constructively, such a suggestion has highlighted a misunderstanding by me or the suggester, which could have led eventually to disaster.

This approach fosters *respect* between team members and makes the team more effective. The original proposer has a stake in the idea, and will champion it further. In addition, other people will be more likely to come to you with suggestions if you never reject them out of hand.

However be careful not to spend *too much time* examining ideas that won't lead anywhere.

Examples

Brainstorming sessions can be a good way to structure a workshop exploring a topic. For example, at a recent one-day workshop (OT98 working group on Software Architecture and People), we started by brainstorming a selection of issues and problems, then used voting to see which ones people considered most important. Smaller breakout groups then discussed the individual problems.

Brainstorming as a technique is used in many conferences – the variety of delegates from different backgrounds can generate good ideas. Here are just three examples we found recently:

- A session in a conference organised by TelOhio, a telecommunications consultancy, explored ways to use teleconferencing for education (<http://www.lsc.net/telohio/groupc21.html>)
- The Cedar Rapids Public Library's Pathfinders (an interest group for school students) used brainstorming to explore ways to use the library more effectively (<http://staff.lib.utexas.edu/~beth/TLA/>)
- A conference of state education associations (SEAITC) used brainstorming to find effective ways to use the WWW (<http://www.create-it.com/virtual/new/brain.html>)

See also:

The best introduction on how to run a brainstorming session is a four-page section in the excellent book on negotiation, 'Getting to yes' [Fisher]. A longer treatment of the subject is in '101 ways to generate great ideas', which has many other suggestions of ways to generate solutions to problems. [Foster]

Sometimes it may be appropriate to use an external **Real Consultant** (see **Cardboard Consultant**) as a facilitator for the brainstorming session.

A web search under 'brainstorming' gives a large selection of relevant information – too much to detail here. In particular there are several commercial tools to help with group brainstorming (e.g. www.brainstorming.org).

Solo brainstorming uses what Edward De Bono calls ‘lateral thinking’. His books explain why lateral thinking is difficult and important, and explores techniques for doing it.

Pattern: Solution First

How can you persuade engineers to do something in a particular way?

People don’t like being *told what to do*. This is particularly true of software engineers, and truer still of US and UK software engineers⁵. If you tell them to do something their first instinct is to say ‘no’. And no matter how much managerial or technical *authority* you may bring to bear, a competent engineer can almost always think of a good way to *avoid acquiescence*.

Engineers appear to see any command and, more subtly, any use of the words ‘must’ and ‘ought’ as a challenge. *Subconsciously*, their immediate reaction is to say ‘no’. And because the reaction is subconscious it’s all the more effective.



So simply ordering an individual or a group to do something is *unlikely to work*. And the approach of *instructing* them to do it in a document is virtually without hope.

You could *enforce* things with document inspections, code inspections, threats of dismissal or rewards for compliance. But all of these are *costly* in terms of effort and *ineffective* in terms of result. They *don’t encourage* a team to work together effectively.

No – you need the engineers to *want* to do this thing. But instructing them simply doesn’t work.

Therefore:

Before describing what the team will need to do to get to the solution, how long it will take, how if they were better programmers the company wouldn’t be in this mess, how getting out of the mess will take vast amounts of work, leading them to canceling their holidays, missing their children’s birthday parties, getting divorced, and putting on too much weight thanks to reinvigorated pizza and cola habits, (oh — and also digressing pejoratively on alternative courses of action (or inaction) and their consequent outcomes, and the ultimate effect on the company’s share price) hint about what you guess you hope the advantages will be for the end product.

I’m sorry, I’ll read that again⁶:

Describe the solution and its benefits first. Motivate people by telling them what they will get out of the solution, and what it will mean for the project. Once you’ve described the solution, and everyone has more or less agreed on it, then the way is clear for you to describe the steps leading to it. If there are a variety of possible solutions, you can then describe all the possible solutions and highlight the benefits and drawbacks of each. If you’re done it right, the team will happily trash their personal lives to increase your promotion prospects.

See...?

When people can see the benefits of the solution you are proposing, they will be motivated to find out how to accomplish it, and much more likely to realise it in practice. Even if people are not completely convinced, they are more likely to hear you out, rather than tune you out. With practice, you can learn to describe solutions rather than processes, and to structure your spiel to make the best solution (your preferred solution) clear.



Consequences

Because your reasoning is clear, other people can *follow the reasoning* or *point out faults* in it. If there are faults, people can point them out, so you benefit because you’re *not making a stupid decision*.

If there are no faults in the reasoning, then other people will *accept the solution* that you are proposing. In Jung’s terms, engineers tend to be *thinkers* rather than *feelers*, and make their decisions based on

⁵ Other cultures are more authoritarian.

⁶ A reference to one of the our favorite bygone radio shows ...

logic [Keirse]. So if engineers can visualise a solution and agree with the arguments that led to it, then they usually decide to do it.

When you describe the steps to take to get to the solution, these steps may still come as commands to the engineers involved. But they will now be *motivated* to follow them, because they'll understand that they would be giving the same commands if they were in your position. If you're lucky, they will treat you as a guide giving directions to a place they want to go, rather than as a drill sergeant shouting at the troops to go *over the top* from the safety of your BMW.

However:

You need *good communication skills*, whether in writing or presentation, to communicate the solutions effectively to the target audience. It can be *very difficult* to rephrase a command as a 'consummation devoutly to be wished'.

It's important to be *concise*; verbose presentations bore the audience, losing their goodwill and *reducing your credibility*. It's also important to be precise; wooliness⁷ can lead to errors and misunderstandings. Lastly, it's important not to cheat – if, for example, you omit a known possible solution or some obvious benefit of a solution you don't like, you'll *lose the trust* of your audience and therefore their commitment.

Solution first is much less effective with people of the Feeling personality type, who make their decisions emotionally rather than by reason; this contrasts the 'Thinking' type – see [Keirse]. Among other groups than engineers, Feeling personality type dominates. So this approach is likely to be *less successful* with them. Instead an emotive approach is likely to work better. The most effective motivational presentations combine both approaches. Harry Truman's words "We will put a man on the moon" motivated the entire Apollo project. For the thinkers it showed an end result they could relate to; for the feelers it invoked the glory and romance of something previously unthinkable.

Another problem is when there is *no clear 'best' solution*. But in this case, consistency — everyone agreeing on a single solution — is more important than any particular choice. Engineers respect that need, or can be convinced to do so; and will accept an arbitrary decision or come up with a good solution themselves.

Giving nothing but direct instructions is most effective with people of the 'Stupid' personality type. In our work, we see people like this a couple of times a day, usually when they are shaving or brushing their teeth.

Examples

Charles writes: As a Software Architect working on multi-company projects, I have little direct authority over the teams doing the development. Indeed, I no longer call myself an Architect to avoid the resentment the title invokes in certain engineers. Instead I concentrate on ensuring all the relevant teams understand and discuss the problems and on clarifying the implications of any and all reasonable solutions; it's rare indeed that we can't find a clear favorite. And of course when other people propose better solutions than I can think of, that's great!

I have used a distributed electronic bulletin board system, Lotus Notes, as the main communications medium. A typical design note describes one or more proposed solutions and their consequences. Other engineers will comment to this note, maybe pointing out better solutions or errors in our understanding. This open discussion has been very effective in gaining agreement amongst the teams.

The original idea for this pattern came from an observation about another pattern being reviewed. We noticed that EuroPLoP reviewers were unanimously critical of one particular pattern in a language of about twenty-five. Though otherwise unremarkable, it was the only pattern whose title was a command (Do the right thing). Subconsciously, we believe, reviewers reacted against being told what to do. We've changed the name of the pattern to avoid this resentment. Compare the two versions of the paragraph following the heading 'Therefore' in this pattern. Though the second version is not as funny as the first, it is more comfortable to read, doesn't offend, and is more persuasive.

See Also:

Please Understand Me [Keirse] describes the theory of personality types.

⁷ Wooliness. a. lacking in clearness or sharpness of outline. [Webster].

Getting to Yes [Fisher] describes a very effective technique for negotiating.

References

- [Alexander] *A Pattern Language. Towns Buildings Construction*. Christopher Alexander et. al. Oxford University Press.
- [Beck] *Smalltalk Best Practice Patterns* by Kent Beck Prentice-Hall 1997 0-13-476904-X
- [de Bono] *Lateral Thinking*, Edward de Bono 1970 Ward Lock Educational Limited 7062 3307 7
- [Buzan] *Use Your Head*, by Tony Buzan, BBC Books 1989 0-563-20811-2
Or consult his recent work at <http://www.buzan.co.uk/>
- [Cockburn] *Expert In Earshot* by Alistair Cockburn.
<http://c2.com/cgi/wiki?ExpertInEarshot>
- [Coplien] *A Generative Development-Process Pattern Language*, by Jim Coplien. In *Pattern Languages for Program Design*. Coplien, Schmidt, Addison-Wesley 1995 0-201-60734-4
- [Fisher] *Getting to Yes*, by Roger Fisher and William Ury, Penguin 1983 0-14-015735-2
- [Foster] *101 Ways to Generate Great Ideas*, by Timothy Foster, Kogan Page 1991, 0-7494-0533-3
- [Gamma] *Design Patterns* by Gamma, Helm, Johnson, and Vlissides, Addison-Wesley 1994 0-201-63361-2
- [Keirse] *Please Understand Me* by David Keirse and Marilyn Bates, Gnosology Books 1984 0-9606954-0-0
- [North] *Chloë*, by Freya North, Random House 1998, 0-7493-2348-5
- [OED] *Oxford English Dictionary*, 2nd Edition. CD ROM Version
- [Pairs] *Programming In Pairs* <http://c2.com/cgi/wiki?ProgrammingInPairs>
- [Pursig] *Zen and the Art of Motorcycle Maintenance*, by Robert Pursig,
- [Rising] *Don't Flip the Bozo Bit*, by Linda Rising . <http://www.agcs.com/patterns/bozobit.htm>
- [Viney] *Bluff your way in Consultancy*, Nigel Viney, Ravette Publishing 1996, 0-948456-40-X
- [Ward] *Scientific Debugging*, a presentation by Robert Ward:
<http://dv101s29.lawrence.ks.us/Robert/debug/>
- [Weinberg] *The Secrets of Consulting*, by Gerald M Weinberg, Dorset House 0-932633-01-3
- [WWW] *WWW Webster Dictionary*. <http://www.m-w.com/dictionary>