Interaction Patterns of Agile Development

Jens Coldewey
Coldewey Consulting (BDU)
Curd-Jürgens-Str. 4
D-81739 München
Germany
Tel: +49-700-26533939
Fax: +49-89-74995703
email: jens_coldewey@acm.org
http://www.coldewey.com

According to the Agile Manifesto, agile projects value „Individuals and interactions over processes and tools“. This sounds nice. However, what does it mean to value interaction? The patterns of this paper form the start of a pattern language that might help a team to find reasonable ways to balance the need of interaction and collaboration with the overhead of meetings as opposed to „real work“.

These patterns do not apply to agile projects alone – you find them in almost any organization that understands software development as a human activity of creation rather than as a production process. However, it is agile development that uses collaboration strategically as one of the major tools of project management.

Of course, this is not the first work that has been done in the area. There is a significant inventory of pattern languages on method design in the literature. Some of the patterns already published are very similar to patterns of this paper, some are variants or applications of them in a different context. I have listed these patterns as related with their specific counterparts. In particular you may want to check the following sources too:

- Jim Coplien's web-site for organizational patterns at http://www.bell-labs.com/cgi-user/OrgPatterns/OrgPatterns?ProjectIndex
- Mike Beedle's and Linda Rising's "Scrum patterns" in PLoPD3
- Jim Coplien's "Generative Development Process Pattern Language", contained in the „Pattern Language of Program Design“ that was edited by Jim Coplien and Doug Schmidt and published by Addison-Wesley in 1995.
- Ward Cunningham's: "EPISODES" pattern language published in „Pattern Language of Program Design 2“, edited by John Vlissides, Jim Coplien and Norm Kerth and published by Addison-Wesley in 1996.
- Neil Harrison's "Organizational Patterns for Teams" in the same book is perhaps closest to this language. However, while Neil's patterns concentrate on social implications, these pattern set up a structure of meetings and thus present a slightly different focus.
- Don Olson's and Carol Stimmel's "The Manager Pool – Patterns for Radical Leadership" published by Addison Wesley in 2002.

I've decided to use the „Alexandrian Form“ here to stress the point that these patterns are „fuzzy“. They do not save you from trying what works best for your team and your project.

The Structure of these Patterns

The patterns of this paper describe the collaboration infrastructure of a project. They belong to four different groups:

Communcation and Collaboration:

- Coffee Kitchen (1)
- Private Space (2)
- Pairing Up (3)
- Master and Apprentice (4)
- Peer Review (5)

Tracking and Reporting:

- Adhoc Meeting (6)
- Wanderer (7)
- Status Meeting (8)
- Design Meeting (9)
- Mutual Review (10)

Speculation:

- Planning Workshop (11)
- Retrospective (12)
- Retreat (13)

Infrastructure:

- Mailing List (14)
- Project Wiki (15)
- Information Radiator (16)

Half of the patterns are completed by now, the others are still as thumbnails. However, I decided to include them, because there are important interconnections within all of them that did not deserve being cut.

Acknowledgements

Thanks to my EuroPLoP shepherd Neil Harrison for his tips and his patience with an almost drowned sheep, thanks to Andreas Rüping for his special support, and thanks the EuroPLoP 2002 writers workshop.

1 COFFEE KITCHEN

The daily work gives little room for social interaction beside the current problem. Still, a project needs space for the team members to interact just as humans, not as professionals, and creative work needs slack.

„Sometimes the most effective thing you can do is to do nothing“ used to be Jim Coplien’s e-mail signature. It reflects an experience almost every programmer knows all to well. You work on a very tricky problem until late in the evening without any significant progress. Finally you give up and go home. A few minutes after you have left your office or when you take a shower in the morning the solution strikes you like lightning. When you restart your work again the next day you solve the problem in half an hour.

Like most complex systems the human brain needs distraction and lateral thinking to cope with problems it has eaten into. Jerry Weinberg calls these distractions „jiggles“ after a technical device used in the first radar circuits to deal with occasional deadlocks. These „Jigglers“ inserted random impulses into the system and thus broke hang-ups that might have occurred.

Most creative thinking happens when you give your brain the time to think about different things than the current problem. Chatting to others about their problems or private affairs are a good way to get distraction and address your problem from a new angle you haven’t tried before.

As Tom DeMarco points out, the stars in organizations are those who have the best network inside the organization. They know more people in the company than the average colleague, they need less time to figure out who could help with a problem, and they experience shorter response times. These networks are built on personal acquaintance, constructed during joint work or during social interaction.

However, chatting may distract your peers too, if you just burst into their work while they concentrate on their problem. With several groups in the same room, talking and laughing may also disturb others and eventually lead to conflicts within the team. In addition chatting costs time and each team member may need a different balance between working time and slack to do their job optimally.

Therefore:

Provide a separate coffee kitchen for social interaction. Offer opportunities to have coffee, tea, or other beverages there and provide a place to store private mugs. Make the place comfortable enough so people like to go there, but be careful with seating facilities unless you have pregnant women in the team or others who need to sit down.

➢ ➢ ➢

A comfortable coffee kitchen is one of the most powerful tools to encourage personal interaction and networking. It is separate from the workspace, so everyone who feels the need for a break leaves the space where others might be disturbed and enters a place of lateral thinking. On one side, this physical change already is a jiggle, on the other it creates a tension to go back to your work. The time you need to drink a cup of coffee or tea is a good length for a break and enough time to build connections outside your current work. Still, it naturally restricts the length of the break, especially if you have no seating facilities. If you provide seating facilities you may also want to hang up whiteboards or an internet terminal so a coffee break may slowly morph into a working session.

Bistro tables and maybe barstools are good furniture for a coffee kitchen. Providing free coffee, tea, juice, and soft drinks is a nice token of the importance the organization attaches to social interaction, lateral thinking and networking. A fridge should be provided to store milk and other perishable goods. A stove and a microwave may serve well for a birthday celebration or a welcome party.

. . . .While the Coffee Kitchen promotes lateral thinking, WANDERER (7), PAIRING UP (3), MASTER AND APPRENTICE (4), and ADHOC MEETING (6) deal with interaction that is directly related to the project. Christopher Alexander discusses a similar problem in the public space when he promotes FOOD STANDS and a STREET CAFE. Organizations that want to provide additional space for social interaction may also be interested in Alexander’s LOCAL SPORTS pattern.

Jerry Weinberg discusses Jiggles in his book „The Secrets of Consulting“, published by Dorset House, New York in 1985. The importance of Slack is stressed in Tom DeMarco’s book „Slack“, published by Broadway Books, New York in 2000. The Patterns of Christopher Alexander can be found in his 1977 book „A Pattern Language“, published by Oxford University Press, New York.

2 PRIVATE SPACE

. . . In a collaborative workspace everything is set up to ease communication. However, sometimes people need some privacy to think for themselves, talk to another team member privately or deal with private affairs they don't want to have discussed in the publicity of the whole team.

➢ ➢ ➢

Concentration is one of the most valuable assets in software development. It may cost up to 20 minutes for a brainworker to recover from a single interruption. In addition personal comfort calls for a minimum of privacy that may be endangered in a highly collaborative environment.

Most of us spend more than half of the time they are awake in the office. However, private life does not stand still. Phone calls may be necessary to arrange some child care, to talk to your physician, to appoint with your partner, or to check your stock account. If these activities are banned from the workplace, the associated worries may consume so much of your attention that you have difficulties to concentrate on your work. Still, if they are not banned you may want them to stay your private affair and you may not want to discuss them with the whole team. Most persons also like to have a home at work where they can put up pictures of their family and leave their private belongings.

Work-related issues may call for private talks, too. Solving a conflict with a team-mate is something you usually don't do in public. Or you may figure out that one of your team-mates has made a particular stupid mistake you want to correct together with him or her privately.

All in all, the respect for individuals demands to provide them with some privacy whenever they need some.

Therefore:

Make sure every team member has a private space he or she can retreat to when it is needed. This may be the simple opportunity to shut the door in a single or double office or a separate office to everyone's discretion.

➢ ➢ ➢

Some managers think of private space as something suspicious, as a place where their underlings do something out of control that needs to be kept in the dark, like writing resumes. Agile projects are built on mutual trust and respect, which also includes respect for the privacy of every individual.

The particular set-up of a private space depends on the culture. In the US, where cubicles and open-plan offices are quite common, a few cubicles for private affairs may be a good solution, such as the private places Kent Beck points out in his suggestion for an XP workspace. In Europe, where double offices are more common, no additional measures may be necessary, because these offices often provide enough privacy. In all areas project managers often have single offices, they can offer to the other team members if they need some privacy. However, the shared private space alternatives do not really provide a space to feel at home.

. . . Sometimes not only a single person but a complete group or the whole team needs some privacy from the other affairs in the organization. RETREAT (13) shows a common way to achieve this. HOME OFFICE is another way to balance private and business needs.

3 PAIRING UP

. . . Software development is difficult knowledge-work, that sometimes exceeds the current capabilities of a single person. The person may be tired or the problem might just be too complicated to be tackled alone.

➢ ➢ ➢

Tasks, such as developing a design or complicated Refactoring, require a constant level of high concentration and assessment of different alternatives on quite an abstract level. It is hard for a single person to maintain the concentration for a longer period. Even worse, the person might not even recognize that her or his concentration is fading and the probability of errors raises.

Often the best way to make up your mind is to explain the problem to someone else. Your peer will ask for clarification, point out gaps in your thoughts, or provide additional views on the problem you haven't thought about so far.

However, not everyone is comfortable with showing her or his work to someone else. Someone might be afraid to show bad work or may be afraid to get laughed at. These fears are usually based in the experiences the person has made so far, from early childhood on.

Therefore:

Pair up with a team mate to address a difficult problem. Change the roles between the active one who explains or shows the ideas and the passive one who asks regularly. Working in pairs should be at least a common way to do the work, if not ***the*** **standard way.**

➢ ➢ ➢

Pairing up is a very powerful way to improve effectiveness. In most cases the team is stronger than the sum of the individuals. Research about pair programming, conducted by Laurie Williams, shows that the pairs do not necessarily speed up, but they produce higher quality and have more fun during their work, raising motivation.

Another advantage of working in pairs is immediate feedback: The person in control of the keyboard or the whiteboard pen gets instant feedback on every detail of the work which may save significant time you would have to invest into extensive quality assurance otherwise.

However, not everyone likes to work in pairs or even in a team. There are individuals who prefer to work on their own and they consider a pair mate as handicap — an experience that often is also felt mutually by the persons who pair up with them. Though it is quite arguable whether persons who *never* can work together with others should work in an agile project anyhow, you also have to consider, that each of us *sometimes* wants to be on his or her own. This may be because it eases concentration or because it is just one of these days when you don't want to see anyone.

There are agile methods, such as Extreme Programming, that demand pairs for certain tasks, such as programming. Other teams just use it regularly to deal with topics of special interest, for example to refactor code that two programmers have written.

To pair up both members of the pair have to sit together. Up to today no electronic communication provides the bandwidth you need to jell as a pair when you work on a difficult problem. However, broad bandwidth techniques, such as video conferences and shared screens may help a pair that is used to work together, to solve a particular problem even when they are separated. Consider one team member who gives support on the client's site while the second team member sits in the lab and gives second-level support to his colleague as an example. Still, these sessions are much more expensive – and usually less productive – than collocated pair sessions.

. . . PAIRING UP is the opposite of PRIVATE SPACE (2). A healthy project needs both to provide a humane workspace. A special case of PAIRING UP is MASTER AND APPRENTICE (4) that features unbalanced partners to convey knowledge. Another form of PAIRING UP is a PEER REVIEW (5), where you ask your pair mate to support you asynchronously.

The first one to fight against lonesome heroes in programming was Jerry Weinberg in his „Psychology of Computer Programming“. Its 25th anniversary edition has been published by Dorset House, New York, in 1998. Laurie Williams measured the effects of pair programming on the quality of the code and the motivation in her Ph.D. thesis at the University of Utah in Salt Lake City.

4 MASTER AND APPRENTICE

. . . While traditional processes mainly rely on written documents to communicate knowledge, agile projects try to write as few documents as possible.

➢ ➢ ➢

In an environment that uses documentation only if they are part of the delivery, knowledge transfer between team members becomes a matter of direct collaboration instead of document transfer. You cannot just pass a well-written document to the novice and expect her to read it. Rather you have to transfer tacit knowledge that resides in the heads of the experts only. Still, those who have the knowledge are usually occupied with the project work.

Communicating knowledge is an inherently hard thing to do. In his book „Agile Software Development“ Alistair Cockburn even argues that precise communication between humans is impossible and states that software projects rather need to manage the inadequacies of human communication than to strive for completeness of documentation.

Alistair refers to what the communication psychologists call the sender-receiver-model. Communication is based on messages sent, no matter whether they are written or oral. Every part of this process implies some fuzziness, so what the sender really sends is neither what he or she wanted to send, nor even what she or he meant to have sent. And the way the receiver interprets a message is not what is really included. All of these steps are biased by personal experience and the interpretation of the context every individual has. In most cases sender and receiver are close enough to enable communication at all, but misunderstandings are very common. The way we usually deal with these misunderstandings uses the complete bandwidth of human communication, starting with the words we use, using body language, up to sketches we may make. The narrower the bandwidth is, the harder it is to communicate and the more expensive it is.

This is especially painful if sender and receiver have different levels of knowledge. If both are experts in the same domain, some short hints or reference to common experience may be sufficient to communicate effectively. However, if they do not share this common ground, the more knowledgeable person first has to figure out, at which level she or he has to communicate, a process usually done with a trial-and-error strategy. If the only option is to write a document and throw it over the wall, chances are high, that the communication won't work at all.

A particular hard topic arises when you consider typical project situations in which knowledge has to be transferred. Often the experts desperately need another person to get rid of an extra workload that is swamping them. Transferring knowledge needs time and patience – both not being the major resources you have under stress.

Another issue is the efficiency of learning. Psychological studies (and personal experience) suggest that the fastest way to learn is to do something yourself. Reading about something new makes it much harder to learn it persistently.

Therefore:

Pair up anyone who needs to learn something about the project with someone who is expert in this area. Let them do the work together until the apprentice has learned enough to work on his or her own. The main work — e.g. programming — should be done by the apprentice while the master watches and gives hints on how to do the work.

➢ ➢ ➢

This is the most effective and — in the long term — most efficient learning strategy. It is different from „training on the job", interpreted as a mere management fad that tries to disguise the true attitude: „We want to sell cheap employees for the rates of experienced people but we don't want to invest in education". Letting someone do the work together with a more experienced person guarantees that the apprentices learns things relevant for the job, become productive as soon as possible and build up enough knowledge to do their job alone later.

However, this may be a hard setting for the master when there is short term time pressure. Especially in the beginning the master could move forward faster on his or her own, without the apprentice. Objectively this is not the alternative, because the apprentice has to learn the stuff anyhow. In traditional processes the master would have to write her or his knowledge down, which would chew up even more time. Still, the subjective feeling of being too slow remains and the master has to fight this feeling with professionalism.

It is important to understand that both master and apprentice are just temporary roles in a specific context. In a pair the same person could be the master regarding one topic and at the same time be apprentice regarding another topic. Consider a domain expert pairing up with a programmer to write acceptance tests: The domain expert is the master in the domain but an apprentice in writing tests, the programmer is an apprentice in the domain but hopefully skilled in writing tests.

You can also use this pattern to spread knowledge about a subsystem to team members of other subsystems. Invite them to work on your team for two or three weeks and go into their team for some time to learn about their needs. After this time both of you will have a better understanding of the problems and requirements each side brings into the game. This not only spreads knowledge but also extends your network and deepens your relationship to the users. People who have worked together tend to be much more responsive, more constructive, and tend to have shorter response times. In addition it calms down bad politics.

. . . Christopher Alexander describes MASTER AND APPRENTICES as a fundamental setting for learning and discusses room layouts to support this setting. In EXPERT IN EARSHOT Alistair Cockburn describes a similar setting, where several „apprentices“ sit in earshot of the expert to see her or him doing the daily work and thus learn without actually pairing up necessarily. Jim Coplien describes a related pattern, called ARCHITECT ALSO IMPLEMENTS where the architect of a project not only develops the architecture but also does programming work. In this setting the other programmers can learn from the architect — in particular if they practice pair programming — while the architect learns from the programmers about their problems implementing the architecture.

Pete McBreen gives an extensive discussion on the idea of craftsmanship in software development in his book “Software Development as Craftsmanship”, published by Addison-Wesley, Reading, in 2001. The pattern by Christopher Alexander is contained in his “A Pattern Language”, already cited before. Alistair Cockburn’s “Expert in Earshot” has been published in the proceedings of the XP 2000 conference by Giancarlo Succi and Michele Marchese. The book has the title “Extreme Programming Examined” and is also published by Addison-Wesley in 2001. James Coplien’s “Architect also Implements” pattern is part of his “Generative Development Process Pattern Language” contained in the “Pattern Language of Program Design” that was edited by Jim Coplien and Doug Schmidt and published by Addison-Wesley in 1995.

5 PEER REVIEW

. . . Sometimes PAIRING UP (3) physically for the whole task is not feasible in the current situation. You may have to share the expert, such as the user on site, among several people or you may need a consultant who is on the project only for a few days per month.

➢ ➢ ➢

If you need an expert’s advice or decision when the expert is not available, stopping the work and waiting for the decision might not be the best choice. In many cases you can at least guess what the advice might be and continue to work in that direction. However, sooner or later you have to get the expert’s opinion and therefore risk to have worked in the wrong direction.

Traditional processes use documents to transfer the information between team members. For example the requirements document is meant to transfer domain knowledge from the domain experts to the designers. But, as I already pointed out, written information is very low in bandwidth. Still most programmers or other team members at the end of the “information food chain” have acquired quite a good idea of the domain problems or whatever information they need regularly.

Sometimes the information the expert provides is only needed at a few points of the development at hand. If the expert is not able to support the rest of the time it might not be a good idea to pair up with her and watching her yawning most of the time.

There are also situations where you cannot discuss the problem with the expert because she does not understand the problem at the current stage. For example a user might not be able to discuss concepts based on UML diagrams. She may need to try a working user interface to figure out whether it suits her needs or not.

Therefore:

Start your work but be sure that the expert can take a look at and review it as early as possible. Try to guess what the expert would tell you. Make sure the material the expert has to review fits his or her mental model of the problem. Take care to keep the parts of your work changeable that rely on the guesses so you can still react to the review comments of the expert later.

➢ ➢ ➢

A review makes sense only if you have the option to draw your consequences from the review results. If the expert thinks, a different result would have been better and you answer, „Oh sorry, I can't change that anymore" the review is a waste of your's and the expert's time.

The key here is not only to make your guessing as good as possible, but also to do the review as early as possible and the product as changeable as possible. For example, it is easier to change a user interface design on a white board than after programming it in native C directly on the API.

Still you might have chosen a completely wrong direction and your work was in vain. This is the highest risk you take using this pattern, because it cannot avoid wasted effort you put into the material *before* the review takes place. Whether this is acceptable, depends on the amount of work you put on stake. A few days of work might be painful but they usually do not effect the project's outcome. Using this pattern to judge on months of work is simply bad management.

It is important to understand that these are not the kind of reviews most participants fear, where ten or fifteen politically chosen persons come together to tear a hundred page document into pieces that is due in one week, so the only correction that is possible anymore is spelling. These kind of reviews are annoying and shouldn't be done at all. These are also no reviews to gather QA statistics about the number of minor and severe „defects" in a result. Rather these reviews are a way to discuss the way the project should move on if physically PAIRING UP (3) is not sensible.

One particular useful application of this pattern is when you have several clients distributed all over the country and you want to add a major feature to your product. After meeting with the users and figuring out what they need, you develop you first-cut version of the feature together with your customer on site. Then you travel around again, showing the first-cut version to all of your clients and ask them for their feedback, before developing the final version. If the first version is throw-away code, this is just prototyping. However, the better choice often is to write production quality code right from the beginning and keep it changeable.

Under any circumstance prefer a feedback in direct communication over written feedback. Sending written feedback is error prone and may give raise to misunderstandings. If you really cannot come together in person, you can exchange notes or email and then talk through the notes on the phone. It is very important that you have the chance to clarify open issues in the notes.

. . . In many cases PAIRING UP (3) physically is the better choice, because it provides a much closer communication between the two persons and a better understanding of each other's problems. The patterns of a WRITERS WORKSHOP give good hints on how to give feedback without damaging the dignity of the author. A good format for the papers to be reviewed are what Ward Cunningham describes in his TECHNICAL MEMO pattern: "Focus each memo on a single subject, and keep the text short and to the point."

6 ADHOC MEETING

. . . Often during development important issues come up a larger group has to deal with. It may be a major refactoring, external events that have effect on the project or a difficult bug that needs the know-how of several team members.

➢ ➢ ➢

There are events that cannot wait for the next general meeting. If an important client reported a severe crash of the system, you'd better react soon. Still, you may be reluctant to disturb everyone in her or his work or it might be unsure who can contribute to the solution.

The traditional way to deal with unforeseen events is to invite for a meeting: The project manager determines who should contribute to the solution, or who should attend for political reasons and finds a time slot in their groupware system none of the participants have blocked. An invitation and the necessary documents are sent around and many of the meetings end up as search for the responsible and blame transfer, usually to those who are not present. Setting up meetings like that also takes a long time. One or two weeks are rather the standard than the exception.

Meetings like that tend to develop their own culture. Since people get invited, some of those who are in the meeting are not really interested in the topic while others who need the results are forgotten. Especially in large projects persons thought to be important spend most of their time in meetings. I once heard a project manager complain: „I've spent the whole last week in meetings and seventy percent of the topics were repeated all over again!" In projects like that these poor persons — mostly managers — get chewed up by meetings and have no time to do their real work: facilitating the team.

Respect for the rare resource „time" of every team member is an important issue when you need a group to get together. Ideally everyone who has a stake in the issue to be discussed should be able to decide whether she or he wants to participate. The time for the meeting should then be scheduled accordingly, so that everyone who is interested can participate. But this may take time.

Therefore:

When the issue arises, call everyone for an immediate adhoc meeting. Only those who are interested in the issue are going to participate. During this adhoc meeting start with a description of the problem at hand and then decide whether to go on to solve the issue immediately or whether to schedule another meeting. Limit the meeting to ten minutes, than ask who thinks that he or she can contribute to the solution. Give a break, so that everyone who answered no can leave without embarresment.

➢ ➢ ➢

Adhoc meetings should be called in with care, since they tear everyone out of their flow and steal time. However, the group might be able to find a solution quickly or at least figure out how to go on. If everyone interested in the topic is at hand, scheduling of a subsequent DESIGN MEETING (9) can be very fast. Everyone can decide herself whether her own stake is large enough to defer the solution until she has time or whether she withdraws for the sake of a faster solution.

Since not everybody who could be interested might be around at the time the meeting is called in, someone important might be missing. Hence it is important for everyone on the project to understand, that this is not a political move but an attempt to push the project forward. If the group thinks, an important person is missing, they can either postpone the solution until this person is present or use a PEER REVIEW (5) to get his or her opinion later.

In some projects you find that a STATUS MEETING (8) is turned into an ADHOC MEETING (6). This often leads to frustration because it is hard to leave a status meeting. In contrast an ADHOC MEETING (6) should cultivate a drop-out attitude: A participant can leave at anytime if he finds out, that he cannot contribute enough to satisfy the time sacrificed.

7 WANDERER

. . . Though an agile project is built on team collaboration, rather than on a predefined separation of work, keeping track of the project status is important. According to organizational research, monitoring self-organizing teams is vital for the project's success to avoid chaotic dynamics.

➢ ➢ ➢

The project manager, the architect, or the coach can only provide the coordinating services to the team, if she or he has a good notion of what is going on in the project. To gather this information is one of the most important tasks of management.

In process-oriented environments the standard way to gather status information is the status report. Programmers have to write weekly status reports, the project manager compiles into monthly reports. This compilation process may be repeated all through the hierarchy. However, this is no guarantee for accurate or even correct information. In his 1971 book "Psychology of Computer Programming" Jerry Weinberg describes a project where the compilation process took about six weeks and lost nearly all the contained information. The team managers were forced to write their reports six weeks in advance and were quite aware that through the six stages of compilation their own contribution got lost. The result was a completely faked monthly report that had only very vague resemblance with the real status of the project. The surprise came when integration started after several years of development and there was no way to camouflage the real status anymore.

Another issue in collecting the information is the workload it produces on the developers. Writing status reports is one of the favorite sources for jokes in the Dilbert cartoons, mainly because it puts work on the developers they can't see any purpose in.

On the other hand if the coordinating members of the team, project manager, architect, coach, don't have a clear picture of what is going on in the project, they cannot do their job. In addition the stakeholders have the right to know about the status of the project – after all it's their money that is spent. Last but not least other projects may need to know about the status of deliverables they are waiting for.

Therefore:

Practice „management by walking around“. Visit each team member regularly, for example once a day and ask for the status and current problems. Have a look at progress indicators, such as test cases and be sure to give more positive feedback than criticism.

➢ ➢ ➢

Management by walking around is one of the key techniques of collaborative management. Beside gathering information it demonstrates that the managers care and that they are open to problems. Be sure you develop a good antenna for nuances that express the morale of the team. Good skills in questioning are definitely helpful.

The frequency of your visits depend on what other meetings are established in your project. If you have weekly STATUS MEETINGS (8) or the project progresses very fast, a daily walk around may be appropriate. If you find out, that you don't get too much new information on a daily basis, you may want to lower the frequency. A weekly walk around is probably the lowest you can go, without loosing the connection to the project's progress.

A major drawback of this way to gather information is that it is hard to be traced later. If you manage as a WANDERER, you may have a hard time to figure out who said what a view months later. So it decreases traceability and accountability. If you think that these are important in your project you still may have to use formal status reports. The key question here is: “If I find out that someone has reported something at a given time, what do I do with that information later?”. If your answer is “I guess people will do better work if they are accountable”, you may want to consider that fear of punishment rarely leads to better work. However, if traceability is a sign of the responsibility your team takes, they might agree that written status reports make sense.

You have to take care about your own time management. Some members of the team may ask you to help them solving their current problem and you may get stuck with them. Others may welcome the opportunity for a break and start to chat. Though there is nothing wrong with both of them, it may not fit into your own time frame. If approaches like that lead to excessive time consumption, you may want to point the colleagues to other facilities, such as ADHOC MEETINGS (6) for problems and COFFEE KITCHEN (1) for chatting.

. . . The COFFEE KITCHEN (1) also is a good place to get a feeling for the morale of the team. Ward Cunningham describes this pattern as WORK QUEUE REPORT: “Collect status reports in regular personal interviews conducted weekly. Solicit estimates of days of remaining effort, using contrasts with comparable work. … Use these estimates along with individual dilution factors…”.

In their 1975 book “The Structure of Magic: A Book About Language and Therapy” Richard Bandler and John Grindler describe an interesting linguistic approach to lead a question-and-answer dialog and clarify imprecise statements.

8 STATUS MEETING

How can the team distribute information about the current status and make sure the project keeps its focus?

Have a regular status meeting with all team members participating. Every team member should answer the three questions „What did I do since the last meeting“, „What am I going to do until the next“ and „What were my main obstacles“.

9 DESIGN MEETING

How do you find the solution to a tricky design problem?

Bring two to four team members together and let them discuss the issue with a whiteboard or whatever tool they consider useful. The discussion should last until they know how to move on, or divide the problem into smaller chunks.

10 MUTUAL REVIEW

How do you ensure that code and documents get reviewed without drifting into unreasonable quality standards or frustration?

Everyone in the team should both review and be reviewed. Review standards are not developed in advance but by the entire team based on the most often observed problems.

11 PLANNING WORKSHOP

How do you ensure that as much knowledge as possible is considered in the project plan and that most team members buy into the plan?

Have a planning workshop at the start of each increment. All team members should participate. The group starts with a prioritized list of requirements and estimates every requirement. Then the group agrees which features will most probably be done in the next increment and which features might be included.

12 RETROSPECTIVE

How do you keep the process alive and adaptable to the project’s needs?

Have a retrospective workshop regularly in which the team answers three questions: „What went well in our process?“ „What didn’t“ and „What do we want to change in our process?“

13 RETREAT

How do you organize a meeting if the project needs a boost, either technically or socially?

Rent a small hut somewhere in the wilderness and go there for three to five days to discuss the issue.

14 MAILING LIST

How do you provide a communication channel for information that could be received asynchronously?

Establish a mailing list with every team member of the project on. You can also establish different mailing lists for the core team, for clients, and so on.

15 PROJECT WIKI

How do you provide a communication medium that both organizes the information and provides a space for asynchronous discussion?

Install a WikiWiki web for your project.

16 INFORMATION RADIATOR

How do you distribute information everyone needs as subconscious knowledge?

Write or draw the most important information on a poster and stick it on the wall where every team member passes by several time a day.

I stole the name of this pattern from Alistair Cockburn. It was too good not to be stolen.