A Medical Catalog of Project Management Patterns

Alistair Cockburn, Humans and Technology
7691 Dell Rd, Salt Lake City, UT 84121 arc@acm.org 801/943-7772

Abstract

It is difficult to identify and locate the pattern that you need. This paper presents a locating scheme based on sensations and symptoms that cause you to look for a pattern. It is tried with project management patterns because they carry symptoms and sensations suited to the medical metaphor. The reader is invited to consider whether this locating scheme works for other patterns.

The symptom-pattern form

Even just with the Design Patterns book (Gamma, Helm, Johnson and Vlissides, Addison-Wesley, 1994), people comment they don't know how to find the pattern they should be reading. With the increasing number of patterns available, that task is getting harder. At an informal gathering, Martin Fowler described the medical system of the British National Health Service, and it seemed remarkable how similar the medical problem is to the patterns problem. A doctor needs to *diagnose* a situation, run a *procedure* to check the diagnosis, and then make a *recommendation*. The problem facing doctors, as pattern readers, is the large number of possible diagnoses, and the need to find the right one. The decision of what to use depends on sensations and symptoms (a sensation is what the patient is feeling, a symptom is what the doctor detects).

A catalog of project management risk-reduction patterns is a good place to apply the medical metaphor, since the situation implies there is already a sense of discomfort. The catalog would be several hundred pages long. How would a person find the right pattern to apply? I propose: by issue, by sensation and by symptoms. The catalog contents are first divided into chapters by primary issue: **productivity, communication, distractions, efficiency, quality**, etc. A diagnosis table of sensations and symptoms helps the reader find proximate and crossover patterns. This paper shows the diagnosis table and five representative patterns. More patterns are listed in the table than there is space for in the paper. The ones in the paper are marked in the table with a '*'.

Different symptoms may lead to the same pattern recommendation. This is reasonable since a pattern represents a balancing of forces, symptoms revealing a lack of balance. Also, more than one pattern might fit some symptoms. This is partly because of the overlapping writing of different authors, and partly because some patterns are specializations of others.

The basic pattern template used is:

Chapter - The primary issue addressed
Name - The name of the pattern
Sensation - How you might feel at this time
Symptoms - Relevant characteristics of the project at this time
Forces - Forces pushing you in particular directions
Try this - A recommendation based on experience
Counterforce - What causes you to stop applying the pattern
Examples - Situations where the recommendation proved useful
Principles Involved - Why the pattern appears to work
Related Patterns... Up: Higher-level, **Down:** Lower-level and **Alternate:** patterns
Reading - Further reading as referenced in the text

The sources for this excerpt of the catalog are:

(Cockburn 95) Cockburn, A, "Prioritizing forces in software design", Pattern Languages of Program Design 2, Vlissides, Coplien, eds., Addison Wesley, 1995.
(Cockburn 96) Cockburn, A, "A medical catalog of project management patterns", Pattern Languages of Program Design 3, this article.
(Coplien 94) Coplien, J., "A development process generative pattern language", Pattern Languages of Program Design, Coplien, Schmidt, eds., Addison Wesley, 1994.
(Harrison 95) Harrison, N, "Organizational Patterns for Teams", Pattern Languages of Program Design 2, Vlissides, Coplien, eds., Addison Wesley, 1995.
(McCarthy 95) McCarthy, J., Dynamics of Software Development, Microsoft Press, 1995.

Diagnosis Table

Chapter	Sensations	Symptoms	Pattern(s)
Communication	– Teams not communicating – "Throw it over the wall" development – Process too bureaucratic	– Teams structured by specialty or phase deliverables, passing work to each other by written deliverables, not able to get their discoveries incorporated into connecting teams' work patterns – Lack of respect across teams	*Holistic diversity (Cockburn 96) Diversity of membership (Harrison 95) Feature Teams (McCarthy 95)
Communication	– Each person trying to do everything	– No specialization – Each person assigned to do everything, resulting in waste from changing gears.	*Holistic diversity (Cockburn 96)
Communication	– No unity of purpose in technical effort	– People pulling in different technical directions	Unity of purpose (Harrison 95) Architect controls product (Coplien 94)
Communication	– Cliques and splinter groups	– Communication distances too large, lack of a core group	Buffalo mountain (Coplien 94)
Distractions	– Team getting distracted – Distractions causing loss of design cycles	– Non-primary tasks dominating time, keeping team from moving forward with their primary goal – Possibly, spending too much energy switching contexts	*Team per task (Cockburn 96)
Documentation	– Too many deliverables to maintain properly	– Too many deliverables make for a heavy bureaucratic load – Deliverables cannot be reduced	Owner per deliverable (Cockburn 95)
Distractions	– This is a distraction to our primary purpose – Distraction causing loss of design cycles	– Some important interruption taking time from all the team members – Interruption can be handled by one person	*Sacrificial lamb (Cockburn 96) Mercenary analyst (Coplien 94) Firewalls (Coplien 94) Gatekeeper (Coplien 94) *Day care (Cockburn 96)
Efficiency	– We don't have time to wait for the previous (upstream) teams to get done!	– Team members sitting idle since upstream tasks not completed. – <u>Holistic diversity</u> already in place.	*Gold rush (Cockburn 96)
Ownership	– No one seems to own this deliverable. – No one seems to work on it.	– "*Ignored area*" - nobody working on maintaining some deliverable. – Ownership by classes only, making functionaliy an ignored area. – Documentation and test cases easily become ignored areas.	Owner per deliverable (Cockburn 95) Code ownership (Coplien 94) Mercenary analyst (Coplien 94)
Ownership	– No one seems to own this deliverable. – Several people work on it. – Ownership unclear.	– "*Common area*" - multiple people working on it, nobody knows what can be discarded. The area cannot be cleaned up. – Ownership by function only, making the classes common areas.	Owner per deliverable (Cockburn 95) Code ownership (Coplien 94)
Training	– Experts are being diverted teaching. – We are losing precious expertise.	– Experts not proceeding at the rate they would expect. The training is draining their energy, time, concentration.	*Day care (Cockburn 96)

Communication

Name **Holistic Diversity**

Sensations

1. Overspecialization:
   - We are doing "throw it over the wall" development.
   - Inter-team communication is not functioning properly
   - The process is overly bureaucratic.
2. Overgeneralization:
   - Each person is trying to do everything

Symptoms

1. Overspecialization:
   - Teams are structured by specialty or by phase deliverables.
   - They pass work to each other by written deliverables.
   - Teams are not able to get their discoveries incorporated into connecting teams' work patterns.
   - Lack of respect of one team for another.
2. Overgeneralization:
   - Each person assigned to do everything, resulting in waste from changing mental gears.

Forces

On the one hand, you need different specialties in the project; hence complete generalization will not work. On the other hand, teams of specialists develop unproductive attitudes toward each other.

Try This

Assemble cross-specialty teams (e.g., end user needs, system interfaces, program design techniques, UI design techniques, programming, database, test, technical writing) so each specialty is represented on each team. Make the cross-specialty team responsible for delivering a subset of system functionality, jointly responsible for the quality of the requirements, design, test, documentation, etc. They have no internal documentation requirements, although they do have documentation requirements responsibility to the rest of the project. However they choose to split up their work is their choice.

Counterforce

A person cannot be on more than a few teams at one time. If there are too few people with a particular specialty, they cannot effectively be on all the teams that need them, and another solution will be needed for that specialty.

If there are too many specialties listed, the teams may grow too large to work effectively, and a variation on this pattern may be needed.

Examples

Project Alpha started with no specialties, then built teams by specialty, then by no specialties again, finally by Holistic Diversity.

We thought each person could learn to do everything, but it became clear that that is an unreasonable expectation. We organized into teams by specialty, but after one release the teams were bickering at each other, complaining about each other, and there was no feedback from design to requirements and analysis. We reorganized into a second attempt at "everyone does everything", but by now it was clear that people had come to specialize on particular areas.

We applied <u>Holistic Diversity</u> to the system function teams to get group ownership of all the steps in the process and close communication. We listed all the skills and knowledge needed to deliver a piece of system functionality, and grouped people so that all skills were present. It became each teams responsibility to see that the functionality was delivered.

After a short time, each team developed specialists who were devoted to working their specialty and communicating to their colleagues. Since they were on the same team, they spoke several times a day, and did not need written deliverables between themselves. This was referred to several times as "Siamese twin" teaming, since the communication was so high between people. It was as though there was one brain with several bodies.

Principles Involved

The principles involved here are to create joint ownership, and to break down team communication barriers. The former, of course, improves the latter, but the idea of not needing internal document hand-offs is also needed to produce the desirable, “Siamese twin” effect. The reduced communication barrier permits programmers (for example) to tell requirements gatherers, UI designers, and analysts where they are causing difficulties, and for the test person to indicate where extra downstream costs are coming from. The multiple heads working closely on a single problem from multiple aspects allows them to discover alternatives and flaws earlier.

This pattern might seem to be at odds with Team Per Task, but in fact, it can as easily be seen as an application of Sacrificial Lamb. The goal is to deliver functionality. That consists of tasks, some of which distract from others. Rather than have one person do all of them, one person is dedicated to each specialty or distracting task.

Related **Up: -**none- **Down:** Gold Rush **Alternate**: Diversity of Membership, Feature Teams

Diversity of Membership is a variant of Holistic Diversity applied to requirements gathering, to include users (Harrison 95).

Feature Teams is the variant described by McCarthy about Microsoft’s teams, in which teams are based around particular features of the end product (McCarthy 95).

Gold Rush presupposes Holistic Diversity or another communication-bandwidth improvement has been put into place.

Reading

Harrison, N, “Organizational Patterns for Teams”, Pattern Languages of Program Design 2, Vlissides, Coplien, eds., Addison Wesley, 1995, pp. 345-352.

McCarthy, J., Dynamics of Software Development, #7: Use Feature Teams, Microsoft Press, 1995.

Distractions

Name **Team Per Task**

Sensation

There are too many tasks, causing us to lose precious design cycles.
We are getting distracted from our primary purpose.

Symptoms

Non-primary tasks are dominating the team's time, keeping it from moving forward with their primary goal. Possibly, they are spending so much energy switching contexts and they cannot get a clear mind to do their main task.

Forces

On the one hand, if you let them drop any of their tasks, your project will miss an important date. So you want several tasks moved forward at one time. On the other hand, having people active on these multiple tasks is not working well.

Try This

Split the team. Sort the activities so that each team has a primary task with additional activities that are not mutually distracting. Sitting in meetings, answering phone calls, writing reports, for example, are distracting to designing software. Arrange it so that each team can focus on its primary task, and each task has a dedicated team member.

Counterforce

You would eventually have one-person teams. Before then, you may discover that it is not worth splitting up the task set the team has because the working synergy between people that is lost is more harmful than the dedicated time gained per task.

Examples

Situation 1: Concurrent next release and getting this release through QA.

Project Beta had one increment entering test at the same time design was starting on the next. We thought (optimistically) the bug fixes would take a relatively small amount of time, and so assigned the whole team to both fixing bugs and doing new design.

Each fix broke a designer's train of thought for a period of time (on the order of an hour) beyond just the fix. Three or four of these caused the designer to lose most of the day. Eventually, the designers gave up on the new release, because they knew the next bug fix would arrive before they would had recovered their thought and progressed on the new design.

We applied Team Per Task, and split the team into two, one for new design and one for bug fixes. The fixing designers rejoined the new design team as soon as the release went through test. The result was that the designers could maintain concentration on the new release, and the bug-fixers were kept busy (there were, by definition, as many fixers as needed so that they would be kept busy and able to make fixes at the needed rate).

Situation 2: Gathering requirements concurrently with designing software.

Project Gamma had each person do requirements, analysis, design and programming. We thought the developers would enjoy the change of activity, that this would reduce the meetings and bureaucratic documentation exchanged between people.

What happened was that the first two activities were so different from the latter two that people were uncomfortable switching between them. After having attended and documented meetings for much of the day, it was difficult to start working on the design and programming. As in the Project Beta, every time a designer was pulled away from her or his work, it cost an additional hour to recover the train of thought.

We applied Team Per Task, and split the teams along task lines. Requirements gathering and analysis went with one or two people in each team, and design and programming went with the others. The result was that the requirements/analysis people sat in meetings, read and wrote specs, examined interfaces and the like. They communicated their findings to the designer/programmers - orally, for the most part, since they were closely linked on the same team (Holistic Diversity). The designer/programmers stayed in their train of thought, getting fresh input from their requirements colleagues. Some of the people put onto requirements really wanted to program, so this was quite a sacrifice for them (Sacrificial Lamb).

Two things we did not do. We did not put the requirements/analysis people into a separate team (Holistic Diversity again). A team was jointly responsible for a section of the system, from requirements to delivery. The splitting was within each team. We also did not require the requirements group to document their decisions for the designers benefit (they did document for the project's benefit). The requirements and design people were in close contact at all times, and most information passed orally. There was, therefore, no "throw it over the wall" effect. These were both important teaming decisions made earlier, which we were intent on preserving.

Principles Involved

The principles involved are increasing flow time, decreasing distractions. "*Flow*" is the quiet time in the brain when (in this case) the problem flows through the designer (Csikszentmihalyi 90, DeMarco 76). It is when the design alternatives are weighed, and decisions are made in rapid succession as mental doors open. The problem, the alternatives and the state of the decision process are all kept in the head. It is a not only a highly productive time, it is the only time when the designer feels comfortable making decisions.

It takes about 20 minutes to reach the internal state of flow, and only a minute to lose it. Our designers found that it took about an hour to get into flow and make progress after having been stopped. If a meeting or other task arrived during this hour, the entire period was essentially lost. As it also took energy to get into the flow, a distraction cost energy as well as time.

To increase flow time, distractions have to be reduced. Certain pairs of activities are more mutually distracting that others. Fixing a bug requires flow in the old system, hence distracts from flow in the new system. Sitting in meetings, answering questions and time on the telephone are major distracters to design flow. In the situations above, we grouped tasks so that each team had relatively sympathetic tasks. Requirements and analysis involve meetings, reading, and writing. Design and programming require concentration on the implementation technology and keeping a great number of details in the head.

Related **Up:** -none- **Down:** Sacrificial Lamb, Day Care

Team per Task applies when the task is not suited to just one person.
Sacrificial Lamb and related patterns are Team per Task when the task is suited to just one person.
Day Care is Team per Task applied to training of novices.

Reading

Csikszentmihalyi, M., Flow: The Psychology of Optimal Experience, Harper Perennial, 1990.
DeMarco, T., Lister, T., Peopleware, 1976.

Distractions

Name **Sacrificial Lamb**

Sensation

This interruption is causing us to lose precious, active cycles.
We are getting distracted from our primary purpose.

Symptoms

A single non-primary task is taking noticeable time from a number of the team member, keeping them from moving forward with their primary goal.

This is a special case of Team per Task in which the secondary task is some form of interruption which can be handled by one person.

Forces

On the one hand, you cannot drop the distracting task, it is somehow very important. On the other hand, it is keeping the team from moving on its primary task.

Try This

Assign one person to the distraction. Let the rest of the team move forward at full speed. If one person is insufficient, assign as few as possible.

Counterforce

If this keeps happening, you will have no one performing the primary task, and you ought to examine why you have so many distractions in the first place.

Examples

Situation 1: Scheduling

The schedule was out of date. We thought it would be fair to let each person on each team evaluate their own work, spreading the load, experience and discomfort of work estimation. What really happened was that progress came to a total halt. When the design team got back to designing, a month had gone by with no design progress, and they had forgotten some of the design issues that had been in their head.

One of the teams used Sacrificial Lamb. They drew lots, for one person to do the whole team's estimation while the others got on with the main task. At the end of several weeks of estimation, that team had moved forward while the other teams were at a standstill. Thereafter, every team applied the pattern. The person working on the schedule really felt sacrificed. This pattern was originally called "Scylla", for reasons that will be clear to readers of The Odyssey.

Situation 2: Going through test

As in Team Per Task, one person was assigned to walk a release through test (Sacrificial Lamb is indeed Team Per Task applied to small teams). The person assigned to test felt sacrificed; the rest of the team was happy to make progress.

Principles Involved

The principle involved here is to protect the main task, where constant progress is paramount. In Situation 1, above, it was not that the design teams were complaining of the loss of flow during design - they were simply not doing design, because of the need to do scheduling.

Related **Up:** Team Per Task **Down:** Mercenary Analyst, Firewalls, Gatekeeper

Team per Task applies when the task is not suited to just one person.
Mercenary Analyst is Sacrificial Lamb applied to documentation distractions (Coplien 94).
Firewalls is Sacrificial Lamb applied to project support interruptions (Coplien 94).
Gatekeeper is Sacrificial Lamb applied to technical information flow interruptions (Coplien 94).
Day Care is Sacrifical Lamb applied to training of novices (see below).

Reading

Coplien, J., "A development process generative pattern language", Pattern Languages of Program Design, Coplien, Schmidt, eds., Addison Wesley, 1994.

Efficiency

Name **Gold Rush**

Sensation

We don't have time to wait for the upstream teams to get done!

We applied Holistic Diversity, now, what are our designers/programmers going to do while requirements and analysis is going on?

Symptoms

Team members sitting idle because their upstream tasks have not been completed.

They have good communications with their upstream task members (e.g., Holistic Diversity).

Some information about the downstream task is available.

Forces

You want requirements to be done carefully. On the other hand, downstream people are sitting idle. They have some requirements, and good communications with their upstream task members, so you can rely on rapid feedback and correction.

Try This

Let the downstream people start now. They have some idea of what they are to produce. Rely on the close communication within the team to take care of changes.

Counterforce

Letting a team start early means rework. There are times when you cannot tolerate that rework:

- If the task is the process bottleneck.
- If the rework will take longer than the original task.

Examples

On Project Delta, Holistic Diversity was already in place. Each team had a requirements and analysis person and several designer/programmers. A first cut at the requirements had been done earlier, so a rough set of requirements were available. Much of the system was similar.

The designer/programmers quickly got ahead of the requirements people, who were busy in meetings trying to nail down details of the requirements. If the designers waited until the requirements were solid, they would not have enough time to do their work. They were able to guess quite closely what the requirements would be like, without knowing final details, so they started design and programming right away. The requirements person gave them course corrections after each meeting. The amount of time it took to incorporate those mid-course alterations was small compared to the total design time.

On Project Delta, the database group constrained the process. They could not afford rework. They had to work in the most efficient way possible. Therefore, they did not start early, but waited until their requirements were stable (per the counterforce). The designer/ programmers had enough extra time that they could afford to prototype some test databases for themselves, which were thrown away when the database designers did their final design.

Brooks gives some other suggestions as to what can be done in parallel (Brooks 95).

Principles Involved

The principle involved is that a process not constraining the overall system can afford to be done less efficiently and in parallel (Goldratt 86). It is often the case that the analysts, designers and programmers can get started right away, without having finalized requirements. Serializing their work will take longer than doing 10-20% rework.

Related **Up:** Holistic Diversity **Down:** -none- **Alternatives**: -none-

Reading

Brooks, F., The Mythical Man-Month, Addison-Wesley, 1995.

Goldratt, E., Cox, J., The Goal, North River Press, Great Barrington, MA, 1986.

Productivity

Name **Day Care**

Sensation

Experts are being diluted by training.
We are losing precious expertise progress.

Symptoms

The experts are not proceeding at the rate you or they would expect. Training is draining their energy, time, concentration.

Forces

On the one hand, you would like to give the newcomers the closest proximity to an expert, so the most can rub off. On the other hand, novices distract and drain them.

Try This

Create an experts-only group and a training group under the tutelage of an expert. Let the experts team design most of the system, let the training team focus on quality training. Transfer people to the progress team as they become able to contribute meaningfully.

Counterforce

If there are only one or two experts, they may both have to work on training until there are at least two people for the progress team.

Examples

Situation 1: Experts and novices

When our experts worked together, they got a tremendous amount done. When there was an expert responsible for novices, the expert got little done and the novices got little training. When there was a dedicated mentor, the mentor did almost no work but the novices got good training. So we split the group into experts, and mentored novices.

Some companies have dedicated "Apprenticeship" programs, in which novices are put under the tutelage of a dedicated mentor for 2 weeks out of every 3 for 6 months.

Situation 2: Adding staff to a project

Fred Brooks, in The Mythical Man-Month, talks about the training costs of adding people to a project. They also drain productivity from the experts. The same suggestion applies: put the newcomers in a separate team to learn the system. Move them to the progress team as soon as they are up to speed.

Principles Involved

The principles are synergy vs. distraction. As modeled in (Cockburn 97), experts having to answer novice questions are reduced to a fraction of their productivity, without particularly raising the productivity of the newcomers. If (x) experts produce (E*x), those same experts mixed with (n) novices produce only (E*x/n) + N*n, where N is very small. In Day Care, they produce E*(x-1) + n*N.

The nature of the training does not matter. As described in Team Per Task, the design and teaching tasks are antagonistic, and better split into separate teams.

The principles involved in this pattern predict that the currently standard, and frequently discussed, mentoring ratio of one expert to four novices will not work very well.

Related **Up:** Task Per Team **Down:** -none- **Alternate**: **-**none**-**

Reading

Brooks, F., The Mythical Man-Month, Addison-Wesley, 1995.
Cockburn, A., Surviving Your Object-Oriented Project, Addison-Wesley, in preparation, due early 1997.