Organizational Patterns

Borrowing ideas from
"agile software development"

David Pritchard
October 10, 2013

Introduction

- Improving organizations: building, healing, repair and growth
  - Not by processes etc. (the hell of ISO 9000)
  - More situational – capture the good things that organizations do, write it down as a “pattern”
  - “If you have this problem… try this idea”
  - All of them are small, local ideas – not a huge “this is how to run the perfect company”
- The book is about the software industry, but the ideas are more general
  - In fact – software borrowed the term “pattern” from a famous architecture / urban planning series
    - Alexander, Ishikawa & Silverstein, “A Pattern Language: Towns, Buildings, Construction”, 1977

Introduction

- What did I get from this?
  - Mostly food for thought, ideas to apply
  - Helped me diagnose several problems that I've seen on the job that were done better at previous employers
    - An "aha!" moment when you recognize a problem in your organization
  - Also, good and evocative "labels" for conversations
    - "let's encourage Alice to be the 'Wise Fool' for the team"
    - "get to a decision – even if it has to happen in a 'Smoke-Filled Room'"
- I'm not going to over-explain this. On to the content…

Introduction

- Four groups of patterns:
  - Project Management patterns
  - Piecemeal Growth patterns
  - Organizational Style patterns
  - People and Code patterns (specific to software industry)
- I’ll discuss each group, focusing mostly on the patterns I found interesting
- This is just an introduction and overview – see references if you want to read more
  - Note that I’ve translated software-specific language in a few places

Project Management Patterns

Good Patterns We Already Do

- Community of Trust
- Size the Schedule
- Take no Small Slips
- Completion Headroom

Good Patterns

- Named Stable Bases
- Recommitment Meeting
- Informal Labour Plan
- Work Flows Inward
- Work Episode
- Someone Always Makes Progress
- Day Care

More Patterns

- Private World
- Don’t Interrupt an Interrupt

Less Relevant Patterns 1

- Get on With It
- Incremental Integration
- Build Prototypes
- Work Split
- Work Queue
- Development Episode

Less Relevant Patterns 2

- Implied Requirements
- Developer Controls Process
- Team Per Task
- Sacrifice One Person
- Mercenary Analyst
- Interrupts Unjam Blocking

Patterns We Already Do
(somewhat)

Community of Trust

- *If:* you are building any human organization
- *Then:* you must have a foundation of trust and respect for effective communications at levels deep enough to sustain growth

Size the Schedule

- *If:* the schedule is too long, staff become complacent; but, if it is too short, they become overtaxed
- *Therefore:* reward meeting the schedule, but keep two sets of books

Take No Small Slips

- *If:* you are getting behind schedule and need additional time resources
- *Then:* take one large planned slip instead of allowing yourself to nickel and dime yourself to death with small, unanticipated slips

Completion Headroom

- *If*: work is progressing against a set of hard dates
- *Then*: make sure there is completion headroom between the completion dates of the largest task and the hard delivery dates

Good Patterns

Named Stable Bases

- *If:* you want to balance stability with progress
- *Then:* have a hierarchy of named stable bases that people can work against

Recommitment Meeting

- *If:* the schedule can't be met with simple adjustments to the work queue and staffing
- *Then:* assemble staff and interested managers to recommit to a new strategy based on doing the minimal amount of work to reach a satisfactory conclusion

Informal Labour Plan

- *If:* staff need to do the most important thing now
- *Then:* let staff negotiate among themselves or "just figure out the right thing to do" as regards short term plans, instead of master planning

Work Flows Inward

- If: you want information to flow to producing roles in an organization
- Then: put the (Developer?) role in control of the succession of activities

Good Patterns

Work Episode

- *If:* you need to split up work across time
- *Then:* do the work in discrete episodes with mind share to commit to concrete deliverables

Someone Always Makes Progress

- *If:* distractions constantly interrupt your team's progress
- *Then:* whatever happens, ensure someone keeps moving towards your primary goal

Day Care

- *If:* your experts are spending all of their time mentoring novices
- *Then:* put one expert in charge of all novices, let the others do the main work

More Patterns

Private World

- *If:* you want to isolate technical staff from the effects of changes
- *Then:* allow technical staff to have private (virtual) work spaces containing the entire working (data & software) environment

Don't Interrupt an Interrupt

- *If:* you're in the middle of handling an interrupt to keep the project from getting stuck, and a new urgent need arises
- *Then:* continue handling the current issue before moving on to the new one

Deeper Dive

Someone Always Makes Progress

- …non-primary tasks are dominating the team's time, keeping it from moving forward with their primary goal. There are common complaints of distraction.

- **It is important to keep a team moving forward and to avoid getting stuck on the obstacles**. You need to pay attention to every task, including small diverting ones. But you also need to complete the primary task by an important date.
- Therefore:
  Whatever you try, ensure that someone on the team is making progress on the primary task.

- If you do not complete your primary task, nothing else will matter. Therefore, complete that at all costs.
- You can employ one of a broad range of particular solutions and tactics depending on the exact forces to be resolved. The following specializations are example refinements of this pattern:
  - DevelopingInPairs - one person can always take the keyboard.
  - TeamPerTask - separate tasks into sympathetic sets.
  - SacrificeOnePerson - assign only one person to the distraction.
  - DayCare - separate the training task from that of producing software.
- But, in any case, you will always be closer to your final goal -- which is not always the case when dealing with distractions.
- The psychological effect of this pattern should not be underestimated. If the project is hit with many distractions, it can be demoralizing to see work grind to a halt. However, any visible progress will help the entire team stay focused, and will encourage them to get through their particular crisis, so that they too can once again make progress.
- Carried too far, this pattern might lead you into trouble for not adequately addressing the distractions. But too many distractions are usually a symptom of some other problem; see, for example,FireWalls.

Deeper Dive

Day Care

- … the project has just brought on several new people.

- **Your experts are spending all their time mentoring novices.**
- You begin to hear things like "We are wasting our experts," or "A few experts could do the whole project faster." Indeed, the experts are not proceeding at the rate you or they would expect, because training the new people is draining their energy, time and concentration. But the new people must be trained, by experts, of course.
- At the same time, you must make progress on the project itself.
- Therefore:
  Put one expert in charge of all the novices, let the others develop the system.

- Separate an experts-only "progress" team from a training team under the tutelage of one or more mentors. Select the mentors for their ability to teach design and programming (object-oriented design and programming, for example) to novices. Let the progress team design 85-95% of the system, let the training team focus on quality training, delivering only 5-15% part of the system. Transfer people to the progress team as they become able to contribute meaningfully.
- Make sure that the training team does not simply do training exercises, but actually contributes to the final system in an ever-increasing way.
- If you have many people to train (more than, say, six), you will have to design a series of tasks for them to attempt. Otherwise you may give them a small, real part of the main system to design.
- If the people in the training team are the ones who know the domain, you will have to make some further adjustment, or else the division may cause conflict.

Piecemeal Growth Patterns

Patterns we already do

- Phasing it in
- Diverse Groups
- Matron Role

Good Patterns

- Size the Organization
- Firewalls
- Gatekeeper
- Patron Role
- Moderate Truck Number

More Patterns

- Wise Fool
- Working in Pairs

Less Relevant Patterns 1

- Apprenticeship
- Solo Virtuoso
- Engage Customers
- Surrogate Customer
- Scenarios Define Problem
- Self-Selecting Team
- Unity of Purpose
- Team Pride

Less Relevant Patterns 2

- Skunkworks
- Public Character
- Holistic Diversity
- Legend Role
- Domain Expertise in Roles
- Subsystem by Skill
- Compensate Success
- Engage Quality Assurance

Patterns We Already Do

Phasing It In

- *If:* you can't always get the experts you need
- *Then:* grow new experts from new hires

Diverse Groups

- *If:* everyone has similar views, you have a good team, *but* too much normalization leaves important problem areas unaddressed
- *Therefore:* assemble a diverse team, based on different experiences, cultures and genders

Matron Role (Nurturer)

- *If:* your team needs ongoing care and feeding
- *Then:* include a Matron in the team who will naturally take care of social needs of the team

Good Patterns 1

Size the Organization

- *If:* an organization is too large, communication breaks down, and if it is too small, it can’t achieve its goals or easily overcome the difficulties of adding new people.
- *Therefore:* start projects with a critical mass of about 10 people

Firewalls

- *If:* you want your staff from being interrupted by extraneous influences and special interest groups,
- *Then:* impose a Fire Wall, such as a manager, who “keeps the pests away.”

Gatekeeper

- *If:* you need to keep from being inbred
- *Then:* use a gatekeeper role to tie together staff’s work with other projects, with research, and the outside world.

Good Patterns 2

Patron Role (Sponsor)

- *If*: you need to insulate staff so that Staff Control Process and provide some organizational inertia at the strategic level,
- *Then*: identify a patron to whom the project has access, who can champion the cause of the project.

Moderate Truck Number

- *If:* you can't eliminate having a single point of failure in allocating expertise in roles.
- *Then:* spread expertise as far as possible, but not more so.

More Patterns

Wise Fool

- *If*: critical issues do not get aired easily
- *Then*: nurture a Wise Fool to say the things nobody else dares say

Working in Pairs

- *If:* you want to improve the effectiveness of individual staff
- *Then:* have people work in pairs

Organizational Style Patterns

Patterns we already do

- Face to Face Before Working Remotely
- Shaping Circulation Realms
- Hallway Chatter

Good Patterns

- Few Roles
- Producer Roles
- Producers in the Middle
- Stable Roles
- Coupling Decreases Latency
- Distribute Work Evenly

More Patterns

- Form Follows Function
- Responsibilities Engage
- Three to Seven Helpers Per Role

Less Relevant Patterns

- Divide and Conquer
- Decouple Stages
- Conway's Law
- Organization Follows Location
- Organization Follows Market
- Hub, Spoke and Rim
- Move Responsibilities
- Upside-Down Matrix Management
- The Watercooler

Patterns We Already Do

Face to Face Before Working Remotely

- *If:* a project is divided geographically
- *Then:* begin the project with a meeting of everyone in a single place

Shaping Circulation Realms

- *If:* you need mechanisms to facilitate the communication structures necessary for good group formation
- *Then:* shape circulation realms

Hallway Chatter

- *If:* key staff tend to huddle around the organizational core or supporting roles are inadequately engaged with each other
- *Then:* rearrange responsibilities in a way that encourages less isolation and more interworking among roles and people

Good Patterns 1

Few Roles

- *If:* your organization has high communication overhead and latency
- *Then:* identify the roles in the organization, and keep the number of roles to sixteen or fewer

Producer Roles

- *If:* your organization has too many roles, but does not know which to eliminate
- *Then:* identify roles as Producers, Supporters or Deadbeats; eliminate the Deadbeats and combine some of the Supporters

Producers in the Middle

- *If:* your key staff are somewhat lost
- *Then:* make sure the producer roles are at the centre of all communication.

Good Patterns 2

Stable Roles

- *If:* you have to deal with project disruptions
- *Then:* keep people in their primary roles, and deal with disruptions as temporary tasks

Coupling Decreases Latency

- *If:* you need a high throughput production process.
- *Then:* increase coupling between roles to decrease latency

Distribute Work Evenly

- *If:* you want to optimize the use of human resources
- *Then:* alleviate hot spots of overload on specific groups and individuals in your organization by Distributing Work Evenly.

More Patterns

Form Follows Function

- *If:* there is little specialization, and people don't know where to turn for answers to technical questions
- *Then:* Create domains of expertise called *roles* that cluster around artifacts or specialization

Responsibilities Engage

- *If:* central roles are overloaded but you don't want to take them out of the communication loop
- *Then:* intensify communication more among non-central roles to lighten the load on the central roles

Three to Seven Helpers Per Role

- *If:* you want to even out communication
- *Then:* at least try to limit communication to Three to Seven Helpers Per Role, and to pull up the outliers to the same level of engagement

Deeper Dive

Producer Roles (1)

- ...once you have identified the roles in the organization, you are in a position to optimize the role structure. This usually involves reducing the number of roles, particularly for mature organizations.
- **The overhead and bureaucracy in the organization is excessive, as manifest by the presence of too many roles. Yet all the roles seem important. It looks like there is no way to reduce the bureaucracy.**
- An organization needs some bureaucracy to keep projects running smoothly; there is much administrative work to be done. Programmers don't want to bother with it. But left unchecked, bureaucracy tends to grow: new roles get created and the communication overhead increases.
- People tend to gravitate to those roles they are most comfortable with. This is healthy. However, some people need the recognition associated with titles (German: *Titelsucht*), and roles are obligingly created to fill that need. Such roles have no intrinsic value to the project.
- Over time, the responsibilities of roles evolve. In some cases, the real benefit of a role drains off to other roles, leaving little more than a shell behind. In one organization, the chief responsibility of a particular role was "worry." It added no value to the project. But because of the history of the role, it is easy to simply assume that the role is important.

- *Therefore*:
  Identify each role as a producer, supporter, or roles that add no value to the project (deadbeats). Eliminate the deadbeats, and in some cases, eliminate or consolidate some supporters. Nurture the producer roles; they are the ones that pay the bills.
- Producer roles are those roles that contribute directly to the end product; there is an obvious connection between their work and the revenue of the company. The canonical producer role in software organizations is "developer".

Deeper Dive

Producer Roles (2)

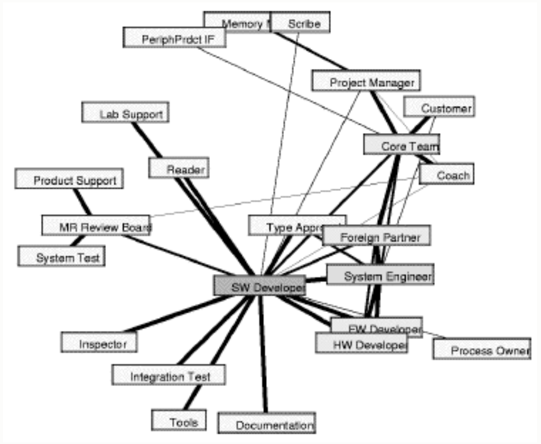
- An organization has numerous support roles. These roles contribute to the effectiveness of the producer roles, but don't directly develop the products. Many support roles are vitally important, such as FireWalls, GateKeeper, and PatronRole. Roles that provide computing support, for example, are also essential. But support roles are inherently higher in overhead than producer roles. There may be opportunities to gain efficiency by combining support roles.
- Deadbeat roles, as other types of roles, can be identified by their responsibilities. They may do nothing more than receive information and pass it on without adding any value to it. Watch for other responsibilities that add no value to the project, such as the aforementioned "worry." If a role truly adds no value to the project, it should be eliminated.
- Note that in some cases, a role that passes information adds value by doing so. For example, a person who passes information by "pushing" it to those who would normally not get the information may prevent project inconsistencies, or might even detect such inconsistencies before they get out of hand (see WiseFool). Such a role is an important support role.
- Although eliminating roles fosters greater organizational efficiency, it may lead to bruised egos, or even feelings of insecurity. In some cases, roles might be preserved, but reshaped to contribute more directly to the project. Refer to FormFollowsFunction and ShapingCirculationRealms for further help.
- It sets up ProducersInTheMiddle. There is a link to DomainExpertiseInRoles. See also FireWalls, GateKeeper, and PatronRole.

Deeper Dive

Responsibilities Engage (1)

- ...the organization has been established, and people have settled into their roles. Communication tends to be centralized.

- **If communication predominately flows through the center of the organization, two things happen: communication takes too long, and the most central roles become overburdened with communication.**
- The most central roles in an organization have the most information about the project, thus they are the most logical ones to transmit and receive information. However, they are also the key producer roles in the organization as well. So time they spend in communication directly impacts their development productivity.
- This figure shows an overburdened central role of software developer:
- But there must be central coordination (which is a weak form of control) or some other acceptable point of control. Fully distributed control tends to lead to control breakdown. Coordination helps accountability, efficiency, camaraderie, can reduce decision time for changes in the business environment (such as requirements changes), and so forth.



- *Therefore*:
  Shuffle responsibilities among roles in a way such that outer roles collaborate with roles other than the most central roles.

Deeper Dive

Responsibilities Engage (2)

- For example, a tester role may be isolated from the project. It would be well for the tester to learn which areas of the project are especially troublesome, so they can be tested especially rigorously. But this information is often not forthcoming. The tester could ask the key developers what the project "hot spots" are, but this would be inefficient and cause bottlenecks.
- Therefore, give the tester some project management responsibilities, where they actively participate in status meetings. They will pick up information relevant to testing through the project management responsibilities.
- Note that in some cases, moving responsibilities will actually cause roles themselves to migrate, and even merge. In most cases, that is actually a good thing.

- This infuses a level of "distributed control with central tendency" that lends overall direction and cohesion to an organization. It complements DivideAndConquer, both by providing for bonds within organization clusters and by providing linkages between sub-clusters, linkages less formal than aGateKeeper role. It adds symmetry to DivideAndConquer.
- This pattern can stand on its own, but it is nicely completed by the application of HallwayChatter.
- Laurie Williams notes that DevelopingInPairs achieves some of the same effect. When she uses this in a pedagogical setting, students learn to rely more on each other and less on the teacher for answers to common questions.

People and Code Patterns

Patterns we already do

- Standards Linking Locations
- Smoke-Filled Room

Good Patterns

- Deploy Along the Grain
- Generics and Specifics
- Code Ownership

More Patterns

- Private Versioning
- Stand-Up Meeting
- Lock 'Em Up Together

Less Relevant Patterns

- Architect Controls Product
- Architecture Team
- Architect Also Implements
- Feature Assignment
- Variation Behind Interface
- Loose Interfaces
- Subclass Per Team
- Hierarchy of Factories
- Parser Builder

Patterns We Already Do

Standards Linking Locations

- *If:* you have geographically separated work
- *Then:* use standards to link together parts of the architecture that cross geographic boundaries

Smoke-Filled Room

- *If:* you need to make a decision quickly and there are reasons to exclude others
- *Then:* make the decision covertly so that the rationale remains private, though the decision will be publicized

Good Patterns

Deploy Along the Grain

- *If:* reuse of work is suffering from a fragmentation of responsibilities for an artefact
- *Then:* give people dedicated, long term responsibility for a management piece of the system

Generics and Specifics

- *If:* you have many new people
- *Then:* put the experienced people on the generic parts of the work, and give specific assignments to the new people

Code Ownership

- *If:* you need responsibility for code and want to build on Domain Expertise in Roles,
- *Then:* give various individuals responsibility for the overall quality of the code.

More Patterns

Private Versioning

- *If:* you want to enable incremental changes without publishing them
- *Then:* set up a mechanism for developers to version code without checking it in to a public repository

Stand-Up Meeting

- *If:* there are pockets of misinformation or people out of the loop
- *Then:* hold short daily meetings to socialize emerging developments

Lock 'Em Up Together

- *If:* your team is struggling to come up with an architecture
- *Then:* isolate them physically for several days where they can work uninterrupted

Closing Thoughts

- There's no prize for using the most patterns. Just use ones that make sense

- Many of these are ideas that only managers can implement

References

- Coplien & Harrison, “Organizational Patterns of Agile Software Development”
  - Draft version online as PDF: http://web.archive.org/web/20050514110633/http://easycomp.info.uni-karlsruhe.de/~jcoplien/HarrisonCoplien.pdf
  - Earlier draft available as web page: http://orgpatterns.wikispaces.com/BookOutline
- Thoughts on how this relates to “scrums”: http://scrum.jeffsutherland.com/2008/02/scrum-and-organizational-patterns.html http://jeffsutherland.com/20071029CoplienOrgPats.pdf