Adopting Agile Practices: An Incipient Pattern Language

Amr Elssamadisy
Gemba Systems

David West
Transcendence Corporation

Amherst, MA
++1-435-207-1225

amr@elssamadisy.com

davew@transcendencecorporation.com

ABSTRACT

The increasing popularity of Agile approaches to software development forces an increasing number of organizations to deal with issues of Agile adoption (and adaptation). This paper lays some groundwork for a pattern language that will facilitate the transition to agility. We introduce patterns that focus on the dynamics of adoption rather than the structure that results from adoption. To establish the desired foundation it is necessary to "push the pattern envelope" in terms of traditional pattern documentation format and relationships among patterns that form a pattern language.

Categories and Subject Descriptors

D.3.3 [**Programming Languages**]: Language Contructs and Features – *abstract data types, polymorphism, control structures.* This is just an example, please use the correct category and subject descriptors for your submission. The ACM Computing Classification Scheme: http://www.acm.org/class/1998/

General Terms

Your general terms must be any of the following 16 designated terms: Algorithms, Management, Measurement, Documentation, Performance, Design, Economics, Reliability, Experimentation, Security, Human Factors, Standardization, Languages, Theory, Legal Aspects, Verification.

Keywords

Keywords are your own designated keywords.

1. Why This Topic and Why Now?

The first agile development processes, such as Scrum and XP, have roots extending back to the mid-1990s. Agile achieved significant momentum and widespread adoption beginning in the years 1999-2003. The Agile community continues to grow but still represents minority of software development organizations. Effort has shifted from rote adoption of a set of published practices (an attempt to treat agility as a method) to attempts to understand the ideas, values and philosophy supporting agility in order to optimize the adoption process. One of the forces behind this shift in emphasis is a perception that agility must be "adapted" to fit specific circumstances before it can be successfully adopted. Adaptations include the combination of practices from several different "methods," selecting (an even discarding) among the set of advocated practices, and finding ways to incrementally introduce new practices. In effect: learning how to be agile about adopting agility.

A set of patterns, preferably a pattern language, that would assist organizations adopting agility is clearly needed and we believe that the body of experiences from which such a language can be distilled is finally sufficient in size to assure some productive pattern mining. Introducing some new patterns and relationships among them (necessary to begin establishing a pattern language) is part of what we hope to accomplish in this paper.

Another goal for this paper is to make the patterns and pattern language accessible to those who will most benefit. The patterns community, *per se*, is not our target audience in this regard. Individuals and organizations new to agility and experienced practitioners assisting in the transition comprise our primary intended audience. To serve their needs we have deviated, in some ways, from the standard treatment of patterns.

Sometimes simply in terms of vocabulary - e.g. we will speak of "smells" more often than "forces." At other times we will follow the lead of Rising, and Manns by presenting patterns that have "tangible but not necessarily structural" consequences and form.

> *A topic for considerable debate among pattern writers is whether or not a solution that has worked well in a particular context to solve a particular problem is really a pattern or whether it is just a general heuristic or guideline. The fallback response to this question has historically been: What does the pattern "build?" The origin of this response is the work of Christopher Alexander, an architect who builds structures. His patterns describe concrete changes in the real world. Since our patterns describe organizational solutions the result is tangible but not usually structural. Therefore we have taken extra care to include both an opening story to provide an image for the intent of the pattern and also to include text that describes what we feel the pattern "builds" – that is what the positive and negative results of applying this pattern will be.*

Our patterns generally address dynamics more than structure and so we copy the format introduced by Rising and Manns.

Another point of divergence is a conscious attempt to produce a pattern language that mirrors, in large part structurally, the domain, Agility, that is addressed by that pattern language. It is our intent that the patterns we introduce reflect the granularity of agile *practices.* Each pattern, therefore, will address the 'how?' and 'why?' of adopting an individual practice, or related cluster of practices. This contrasts significantly with most treatments of agility which focus on describing 'what?' the practices are and the combination of practices into methods.

Based on our collective experience in teaching, leading development teams, and participating in numerous patterns and

A preliminary version of this paper was workshopped at Pattern Languages of Programming (PLoP) '06 October 21-23, 2006, Portland, OR, USA.
ISBN: 978-1-50448-372-3

agile forums (most recently at ChiliPlop 2006 and XP2006) we believe that these patterns have significant potential to help guide the next generation of adopters – allowing them to learn from our mistakes and successes.

2. Foundation and Starting Points

Our patterns derive from the specific way we view three terms, pattern, agility, and adoption. It is useful, therefore, to make this understanding explicit.

Pattern – "a proven solution to a problem in context." Patterns are a proven format for sharing expert information about complex problems. Patterns, including the ones presented here, are validated by their successful application in multiple instances. Documentation of a pattern generally includes: name, definition, context, problem – including identification of relevant forces, solution structure, implementation, and discussion of applicability and other usage issues.

Our presentation of patterns will diverge from these generally accepted definitions in three ways:

> First, we deviate slightly from the classical format used in pattern documentation. We will use the following:
>
> **Name**
>
> **Sketch:** A story/narrative that acts like a 'sketch' in design patterns. (idea taken from *Fearless Change*}
>
> **Context:** Who and in what circumstances this pattern is useful.
>
> **{Forces:}** Used to elaborate context and give specific issues that are problems (partially) resolved by this pattern.
>
> **Therefore:** The pattern description.
>
> **{But:}** Negative consequences that can occur from applying this pattern.
>
> **{How:}** Steps, ordering, guides to adopting this pattern. Sometimes interleaved with **Smells:** to indicate where the adoption can go wrong.
>
> **{a.k.a.}** Similar published patterns.
>
> Second, we introduce the concept of a, "smell," to the patterns community. The discovery of a pattern is driven by the isolation of a problem. But how was the problem itself discovered or identified? The agile community has come to use the idea of a "smell" as a means for identifying potential problems. Smells are not defined, they are recognized. A smell reflects a relatively vague sense of unease or discomfort. Reflection on a smell may result in the identification of a resolvable problem.
>
> A smell is a kind of inverse QWAN. QWAN is the Alexandrian notion, "Quality Without A Name;" something that is readily recognized even though it cannot be defined or explicated. A lot can be said 'about' QWAN even though QWAN itself cannot be expressed. There is no algorithmic means for creating the property of QWAN but there are patterns that can be applied to increase the likelihood of the emergence of QWAN in a construct. Smells too are recognized. Smells can be talked about but not defined, and lack any kind of algorithmic resolution.
>
> Third, we introduce the explicit use of "abstract patterns" - analogous to an abstract class. An abstract pattern might also be seen as a category of patterns. An abstract pattern lacks an implementation independent of those of its concrete sub-patterns.

Agility – a state of being.

Like QWAN it is possible to discern the existence of agility – in individuals and organizations – but agility is not definable. Specific behaviors, or practices, are consistent with agility and their presence or absence is an indicator but not a measure of agility. There is no formulaic path leading from non-agility to agility nor is there an aggregation of practices that ensure agility.

Agile practices are themselves patterns – frequently observed organizational and behavioral solutions to commonly encountered problems. The practices are present, in various combinations, in individuals and organizations that are recognized as agile. However, the mere presence of those practices is insufficient to transform individuals and organizations to the agile state of being.

Agile "methods" and agile "processes" are misnomers precisely because there is no formula or specifiable combination of practices for attaining agility. The patterns and pattern language we introduce in the following pages reflects, we believe, discernable patterns in the dynamics of adoption expressed by individuals and organizations attempting the transition to agility. Necessarily they will reflect some of the same ineffability as agility, QWAN, and smells.

Adoption - the act of taking as one's own a thing or behavior. Too often, agile adoption is seen as little more than exhibiting, by rote, the enumerated practices and exhibiting, via mimicry, the described artifacts in one or more of the agile method books. It was held, and some still hold, that agile adoption was an all or nothing proposition. You did everything listed exactly as described or you were not adopting agile. This view of adoption is actually a smell – specifically the "Cargo Cult[1] Syndrome."

---

[1] During World War II a number of airbases were built on tropical islands inhabited by pre-industrial societies. Soldiers built airfields and control towers and engaged in activities that resulted in airplanes full of cargo discharging their contents. The native inhabitants shared this cargo. After the war the soldiers departed and no more cargo was available. The natives adopted superficial forms of airstrips, control towers, and ritual behaviors to induce the return of planes full of cargo. A cargo cult is any group that adopts form instead of substance and believes that doing so will bring about a desired result.

We see adoption as much more than aping as closely as possible each and every described practice advocated by an agile advocate. We believe that successful adoption requires asking additional questions, such as:

- "Which practice do I adopt first?"
- "Which practices relate to (in terms of support or dependency) others?"
- "Can I incrementally adopt a given practice or incrementally adopt from a set of practices?"
- "Can I adapt the form of a practice without altering its substance?"
- "Can I add to or delete from a specified set of practices?"
- "What values and assumptions are presupposed by a given practice?
- And, consistent with the spirit of agility, "what business value does each practice deliver?"

The patterns we introduce are intended to provide guidance for those embarking on the journey to agility. Our focus is on that problem space and how each pattern addresses both the mechanics of adoption and the expected technical and business value derived from that adoption.

We are not presenting a finished or polished work. At ChiliPlop 2006 we quickly discovered that a full pattern language describing adoption is a monumental undertaking. So the few simple patterns here are only a beginning. By flushing these out and gaining invaluable feedback from the Pattern Language community we will make these more valuable to the readers. In addition to presenting this paper at PLoP 2006, we are making our work and results available on a Wiki at www.agileprocessadoption.com.

3. Mega-Smell – Us vs. Them

An organization or a development team has decided to adopt an agile software development methodology. Unfortunately, that organization exhibits an all too common problem – clearly, though often non-consciously, delineated sub-groups whose interaction is characterized by "CYA" contractual communication. Failure is expected and each sub-group is intent on making sure the "others" rather than themselves are to "blame" for that failure. The organization smells like "Us versus Them."

A number of forces and factors have converged to generate this smell. Among them:

- History – contention between IT and business, analyst and coder, coder and tester, developer and user has been an industry plague since the advent of commercial computing.
- Physical separation which contributes to social separation.
- Any organization in the United States (we cannot indict the rest of the world on this point) is embedded in a very litigious culture where individuals seem loath to accept personal responsibility.
- Industrial cultures have generally abandoned group consensus and informal resolution mechanisms in favor of formal courts and black/white decision making.
- Much of the industrialized world shares a heritage that teaches people are innately flawed but capable of redemption instead of innately good but subject to corruption.
- A long-standing conflict between perceptions of Art vs. Science.
- Documentation is seen by most as a means to represent reality instead of an ephemeral evocation of current group understanding.
- Functional separation (silos) - testing team/ coding team/ analyst team.
- Human Resources (and litigious culture) imposed false equivalency that categorizes people instead of recognizing individual differences.
- A culture of debate instead of dialog.

Although we believe we are the first to identify this problem area as UvT, the problem itself has been recognized by other pattern writers including Coplien, Rising, and Manns.

A significant motivating factor in the development of XP[2] was the need to address and resolve UvT. Each of the original 12 XP practices can be understood as a means of resolution. (This, of course, implies that the Practices are themselves patterns.) Any organization seeking to become agile must adopt a set of practices sufficient to eliminate all (or at least most) vestiges of UvT.

Although the adoption of the practices seems simple (as Nike said, "Just Do It!") organizations and teams have experienced significant difficulties and too many have failed in the attempt. Our involvement with (and observation of) teams and organizations that have been successful suggest that there is another set of patterns – adoption patterns – that increase the likelihood of success when transitioning to agility.

4. The Patterns

In the remainder of this paper we will introduce a selection of the patterns that illustrate the various directions in which we can continue our search for adoption patterns.

4.1 Reciprocal Visibility

Sketch: *Upon joining an agile team Waterfall Will discovered that stand-up meetings not only let him know what developer's were doing but also what Scott the ScrumMaster was doing to remove each and every road-block that came up. When an issue was brought up it was put up on the Impediment Chart (an Information Radiator) and everyday Scott was responsible to report to the team the progress he has made towards removing the impediment. Will didn't think of Scott as just 'overhead' on the team anymore. Will understood that Scott paid attention to what he and the rest of the development team were doing and also understood that Scott was diligently removing roadblocks for the team.*

Context: An organization exhibiting multiple symptoms of the us-versus-them smell, for example: tightly

[2] We are using XP as an exemplar of Agile approaches in this instance. We believe that all agile approaches equally value communication and all have practices specifically addressing the issue of communication.

constrained communication channels; documentation that is viewed as legal contract rather than a communication tool; infrequent contacts among users, managers, and developers; and, clearly delineated specialty areas – e.g. analysts, coders, testers – reporting to different managers.

Numerous forces have combined to create this state of affairs in the organization, but the most significant is 'expectation of failure.' Everyone involved in the project, including management, immediately adopts a posture of self-protection and blame avoidance. The less exposure of one's activities, the lower the portion of blame than can accrue.

Therefore: Success in the adoption of agile practices like the *planning game*, *pair programming*, and *sustainable pace* mandates deployment of countering actions. Reciprocal Visibility provides a pattern that can shape appropriate counter actions. The essence of the pattern derives from a belief that the state of a project and the contributions of all parties to that state should be overtly manifest to all.

Reciprocal Visibility is an abstract pattern, one that describes traits reflected in several related patterns. These traits are not structural (as is typically the case with Alexander inspired patterns); instead they are prescriptive. They define constraints that must be satisfied if the pattern is to exist. Reciprocal Visibility is sufficiently abstract that there are only three constraints:

- All actions that affect the state of the project are expressed in a manner that includes the whole team.
- All information about the state of the project is public and omnipresent in the whole team environment.
- All communication (actions and information sharing) take place in a "safety zone" – i.e. no one can be punished (or rewarded) for their participation in the conversation.

But: Satisfying the constraints imposed by this pattern will almost certainly raise the anxiety level of everyone involved in the project. Managers will be uncomfortable addressing the entire team instead of communicating only with the team's coach. The presence of the manager at team meetings will inhibit – at first – frank conversation regarding issues.

Documenting individual and team performance on the walls of the team work area will expose individual differences that will make some uncomfortable. (It will also serve as a source of motivation for change.)

Some will attempt to abuse the communication safety zone with *ad hominem* remarks. This error needs to be pointed out and corrected – but not punished unless it persists after several reminders.

Reciprocal Visibility is parent to two abstract sub-patterns: Static Information Radiator and Dynamic Information Radiator.

4.2 Static Information Radiator

Sketch: *Dave the Developer and Aparna the Analyst were both on a project that has been incrementally adopting agile development practices. Test-driven-development had made a real improvement in the quality of the code so far, but there was still a lot of improvement to be done. Aparna still saw many issues fall through the cracks, and by the time she got to reviewing the work that Dave had completed for the iteration they would go into overdrive to get the user story completed or just miss completing the card all together. Recently Scott the ScrumMaster put up a StoryBoard chart in the common area. Aparna saw that Dave's work was completed and ready for testing half-way through the iteration and was reminded of it every time she passed the chart. She picked up the work Dave had completed, found the one or two issues and notified Dave. Dave had the issues fixed in a couple of days and the card went through to completion smoothly. It was funny how such a simple chart placed so that both she and Dave saw many times a day helped them both finish the card on time…*

Context: The whole team needs consistent and continuous information about factors that inhibit/enhance the process of delivering business-valued software[3]. Traditional forms of project documentation are (rightfully) perceived as overhead, generally inaccurate, and incomplete. Information must be accurate, exceedingly easy to update, and understandable "at a glance." A Static Information Radiator captures and displays information. It is the medium (e.g. posters, white board diagrams, charts) that is static not the information.

As an abstract pattern, Static Information Radiator extends Reciprocal Visibility by adding additional constraints:

- Every aspect of a project's state of interest to any member of the whole team is an appropriate subject for a static information radiator
- The contents of any given static information radiator are limited to 1-3 distinct aspects of the project state.
- Updates to the content must require nominal effort (e.g. moving a story card from one column of a progress chart to another during a daily meeting) or be automatically generated as a byproduct of development efforts (e.g. tests written, tests passed).
- Radiators should be "large format," i.e. large posters, white board diagrams, or any other equally visible and interpretable form.

[3] Here business value means software that delivers value to your business. This does not mean that this is only good for corporate applications. Scientific programs have a 'business' of that particular science.

- Radiators should be consistent with the pattern, "Evocative Document" as discussed below.

Almost all agile methods discuss and advocate the use of various forms of static information radiators. The idea of "big visible charts" is common vocabulary and addresses, but does not explicate, the same issue as Static Information Radiator.

But: Simply posting a series of "facts" about a projects state is insufficient to satisfy the constraints imposed by this abstract pattern. "Smelly" Static Information Radiators can exhibit one or more of the following symptoms:

- Voids in the information exist indicating that individuals or roles are persisting in their predisposition to secrecy. Some members of the team still believe that they can avoid or transfer to others accountability for failure.
- Team members fail to see any value in one or more of the information radiators in use – e.g. progress is being measured but not in a meaningful way. The information does not lead to improvements in the development process.
- Updates are not performed in the same time frame as they occur and they are of use. Usually means that the updates require far more than nominal effort to make and/or are not automatically generated. Overtime, erratic productivity, and declining team morale are secondary symptoms of this problem.

4.3 Dynamic Information Radiators (Meetings)

Sketch: *Ahmed the Analyst, Tammy the Tester, and Debbie the Developer had all worked together before on many successful (and unsuccessful) projects over the years. Getting a product out the door was hard, and even though they were good friends and frequently went out to lunch together, they each had a feeling that the others "just didn't get it" completely. There were always misunderstandings and complications in the requirements, and of course the code was less than perfect when developed, and the testing team did their best to find all the bugs but somehow never got them all. The project they are currently on however seems to be quite a bit different from the start. There are iteration kickoffs every two weeks where the entire team gets together to make decisions and commitments to what will be done in the next iteration. Daily stand up meetings keep everyone informed and roadblocks are removed ASAP. And finally, at the end of each iteration there is a Retrospective that allows the team to tweak the process itself and get things on the same track. All of this frequent communication has helped the team move functionality through very fast and has reduced many of the errors that would have previously been missed or would have taken several months to catch because of the slow cycle time. They may not have as many 'documents' describing exactly where they are, but as a team they* ***really*** *know where they are and where they are going much more accurately than ever before.*

Context: The whole team needs consistent and continuous information about factors that inhibit/enhance the process of delivering business-valued software. Some of that needed information is not reducible to static form – it is embodied in the actions and decisions of individuals interacting with the team. Traditional forms of inter-personal communication are too limited to meet the needs of an agile team. Circumstances must be crafted that allow multi-channel, simultaneous, and verifiable communications to take place among the whole team.

As an abstract pattern, Dynamic Information Radiator extends Reciprocal Visibility by adding additional constraints:

- Every action, decision, and communication affecting the project must be public to all involved in the project.
- Dynamic Information Radiators are tightly constrained in terms of time and space.
- Dynamic Information Radiators are regularized in terms of scheduling and format. Spontaneous creation of a dynamic information radiators is possible, and sometimes necessary, but should be an exception.
- Dynamic Information Radiators should be "high bandwidth," i.e. involving as many human senses as possible. This implies that face-to-face radiators are preferable to multi-media which are preferable to voice only.

Almost all agile methods discuss and advocate the use of various forms of dynamic information radiators. The most prevalent example of a dynamic information radiator is the "stand-up meeting." Retrospectives and planning games are other common examples.

But: Simple articulation of information about the project is insufficient to satisfy the constraints imposed by this abstract pattern. "Smelly" Dynamic Information Radiators can exhibit one or more of the following symptoms:

- Articulations are misperceived or misinterpreted. The format of the radiator or the circumstances of delivery do not allow for sufficient feedback and clarification.
- Radiators exist for purposes other than sharing commonly needed information. E.g. to reinforce the status of a particular role, often a management role.
- Inappropriate vocabulary is employed, e.g. a vocabulary of "blame."
- Attendance is sporadic and resisted indicating that the information or the format is not of any real value to the team.

4.4 Evocative Documents

Sketch: *Suva and Ademar were on their way home from a weeklong UML training course and discussing what they had learned. "UML certainly provides a rich and detailed tool for describing our software," Ademar noted. "But it can still be misleading," Suva responded, "remember our discussion about the customer class?" "I do," said Ademar, " and how we got into that discussion of what 'is' is – when people started using our UML description as if to say it was a customer." "Oh yes, and that guy in the back talking about Alfred Korzibski and 'the map is not the territory,' that was weird," Suva added. "But he was right, really," continued Ademar, "no matter how much detail you get in your UML model and templates, something is always missing. The model is never the real thing." "And our understanding of what the real thing keeps changing, and changes from one context to another," Suva said, "how can we put all of that in a model?" "Well," Ademar suggested, "we probably don't need to if we can find a way to remind ourselves of everything we know about something when we need it." "How would we do that?" Suva asked. "Remember that icon on the wall of the seminar room," Ademar enquired, "remember when we asked the the facility manager about it and she talked for half an hour about its meaning and history, and everything." "Sure do," Suva responded, "one simple symbol evoked a huge amount of memory. Maybe that is the secret …"*

Context: Literate and legalistic societies and organizations share a deeply held, though often non-conscious, belief that written documents are ***representative*** in nature. A contract **IS** the agreement among parties to the contract. The blueprint **IS** the building, albeit in a different format. The specification **IS** the software artifact desired. This belief is so strong in the arena of software that many believe that it should be possible to formally and mechanically transform specifications into an artifact with no interpretation or ambiguity.

Agile development is the epitome of group "theory building" as described by Peter Naur. Agile practices, more than any other kind of development practice require the creation of a rich and easily accessible "external memory" as described by Bo Dahlstrom. Representational documentation is notoriously limited and has a long track record of failure in this regard.

Therefore: All documentation should be evocative rather than representational. Anyone that has read a good novel is familiar with the notion of an evocative document – one that enables the reader to "recall to mind" thousands of sensations, emotions, even details of time and place that the author could not possibly have included in the text of the novel. West has previously written about the power of evocative documentation in agile development.

One force that must to be taken into account when attempting to create evocative documentation is the previously mentioned assumption: documentation is representative. Documentation that is highly stylized, that uses precisely defined and context free syntax (e.g. UML) will almost certainly be perceived as representational rather than evocative. The constraints imposed by Evocative Documents' parent, abstract, patterns (Reciprocal Visibility and Static Information Radiator) are additional forces that must be accommodated.

Evocative Documents are:

- Informal – 3x5 cards rather than UML diagrams.
- Natural language based – both in terms of the natural language used by the team for communication inside and outside the office, but also in terms of the domain driven vocabulary of the project itself.
- Rich in referents to people, time, and place.
- Inclusive of .jpeg rather than .gif graphics (i.e. photos rich in color and detail of the sort typically save in jpeg format instead of the line drawings and UML diagrams typically saved in .gif format).

But: Your documentation has probably slipped into representational form and is no longer evocative if:

- It takes longer to produce the documentation than it does to comprehend and use it. (Refer to constraints stated in Static Information Radiator.)
- Anyone in the organization begins to express a belief that the documentation has intrinsic value and not just utilitarian value.
- There is any kind of movement to make the documentation archival.

Specialists are employed to produce the documentation. There is an exception to this rule, technical writers (who should really be more novelist than tech writer) charged with producing manuals and books for users of the software that were prevented from participation in its creation.

4.5 Stand Up Meeting

Sketch: *When Joe joined his current project it was his first 'agile' project and he couldn't imagine a meeting every day. The previous meetings to him were almost always a waste of time where he had to sit through discussions that weren't always very related to his work. Of course, the really important meetings where decisions were made about scope and deadlines did not necessarily include him. So he went to his first Stand Up meeting which was refreshingly short and very focused on the iteration at hand. After several meetings, he also realized that impediments to meeting iteration goals were addressed by Scott the ScrumMaster quickly. In short, Stand Up meetings were **relevant** to the current iterations work and were not too much of a burden to attend.*

Context You are an organization that has started to see software development as an *Empirical* process instead of a *Deterministic* process. Therefore you need constant information about where the current project is so that you can precisely control where it is going. Your

organization is also working to establish and maintain a 'whole-team' that is does not suffer from *Us vs. Them* via *Reciprocal visibility*. Your team is in the process of improving its communications.

Forces:

- Software projects are empirical in nature and not deterministic, therefore constant readings of where the project stands is necessary.
- Meetings tend to be long and wasteful because they mix so many agendas – both explicit and hidden.
- It is necessary to structure meetings and interactions so that they focus on one specific purpose – for example, making sure everyone understands what progress is being made towards a collective goal.

Therefore introduce Stand Up meetings as feedback for management of an empirical process. The daily meetings will give the entire team relevant information to adapt to changes and new information within an iteration so that obstacles can be addressed in a timely manner and the goals of the iteration can be met. Stand Up meetings also help establish *Reciprocal Visibility* among the different members of the team as the see that the entire group (management, analysts, developers, testers) work together to meet the iteration goals.

But your meetings can easily go off track or be co-opted by management. Some meeting smells – indicating a need to pull back and correct the format and/or purpose of a meeting include:

- Meetings become travelogues, i.e. people tell what they did in detail instead of short, concise status and impediments.
- Meetings become design sessions, the need for design discussions can be recognized inside the Stand Up but should be scheduled at another time with the relevant participants.
- Meetings become planning meetings. Planning should be done on iteration boundaries and outside the Stand Up meetings.
- Meetings are not regular and are dropped because little or no value to the meetings is perceived.

How. Have one person in charge of keeping the meeting on track, that is they must be responsible for:

- Promptly starting on time
- Attendance
- Letting the pigs talk and keeping the chickens[4] as listeners
- Keeping the meeting under 15 minutes
- Interrupting burgeoning planning and/or design discussions and having them be scheduled after the meeting.
- Keep the focus on concise status and not letting it turn into a travelogue.

a.k.a.

[2, 3] Describes a stand up meeting with respect to eXtreme Programing (XP).

[7] **Stand-Up Meeting** pattern overlaps with this pattern but is more broad in that they may "be held for the purpose of reviewing the architecture" whereas our experience suggests that this type of discussion be taken off-line with respect to the daily Stand Up Meeting.

[12] presents an entire pattern language to describe the details of a Stand Up Meeting.

[11] defines a Stand Up meeting with regards to Scrum.

4.6 Reaffirmation Ritual

Sketch: *Once upon a time, at the World's Fastest Computer Company held an annual event called Ducky Days. On the surface this was a typical company picnic intended to improve interactions and professional relationships among the employees of a large organization. It transcended most similar corporate events in that it also celebrated a prized aspect of corporate culture – egalitarianism, free-spirited creativity, and grass roots management. Ducky Days was really a reaffirmation ritual – helping all involved remember, in a light hearted and social way, the core values of the organization. The story goes, that when the company began to grow it needed to add managers from the outside world. One such manager was a notorious "stuffed shirt." One day an employee put a yellow plastic duck in the fountain outside the entrance to corporate offices. The uptight manager was appalled and notified all employees that such behavior was undignified and not to be tolerated. The next day, you guessed it, the fountain was full of yellow ducks. The manager left the company shortly thereafter and Ducky Days commemorates the victory.…*

Context You are an organization that is working to establish and maintain a 'whole-team' that is does not suffer from Us vs. Them via Reciprocal visibility. Your team members tend to get 'heads-down' in their work and, especially with larger teams, the do not communicate outside of their tasks.

Therefore use Reaffirmation Rituals to give the teams a laid-back atmosphere to enjoy time together. They will naturally discuss their work and keep in touch with each other. A team lunch is an excellent venue and one of the most common reaffirmation rituals.

But (Smells) Be careful to keep these rituals informal. In one particular case lunches were very productive until the project manager started showing up and tried to 'run' the lunch like a stand-up meeting. This destroyed the atmosphere and reduced the high bandwidth informal communications which were happening without him.

a.k.a *Small Successes* in *Fearless Change* has a lot of overlap.

[4] The terms "pig" and "chicken" in this context are derived from a story about a pig and chicken starting a restaurant. Given a menu of bacon and eggs, the pig is committed and the chicken is merely involved. The suggestion here is that only those who are committed – those that will suffer the repercussions of failure – should speak in stand-up meetings even though all involved may and should attend to gain an understanding of what is happening with the project.

4.7 Solidarity Ritual

Sketch: *The project was certainly important enough. Everyone seemed eager to be on the project and to utilize the new agile approach. But the team just wasn't coming together. Stand-up meetings were characterized by a kind of finger pointing, e.g., "I gave the specs to Jose yesterday, my roadblock is not getting the updated customer requirements from Julia." Angela, the coach, recognized that the team needed some rituals to create and maintain a sense of common identity so she approached the IT director with a list of requests. "We really need our own space to consolidate the team. We can start with one wall of the cubicle space where we will own all the whiteboards and bulletin boards. We will conduct our daily meeting in the aisle in front of that wall. We also need some marker of team membership. I talked with HR and they said they could issue a different color security badge to the members of this team and since everyone has to wear those badges in a visible way every day – it will help everyone instantly recognize who is and is not a member of the team." "Well those sound simple enough," the manager replied, "what else might you need?" …*

Context You are an organization that is working to establish and maintain a 'whole-team' that is does not suffer from UvT. Your team is in the process of building itself and establishing its culture.

Forces. Organizations are comprised of individuals. Even though some cultures emphasize individualism more than others almost everyone is shaped by idiosyncratic needs and history. We all return home to separate homes which are the real focus of our lives. Large organizations are multi-cultural in composition. Different people bring different food for lunch, may speak different languages around the water cooler, have different artifacts in their cubicles. Assertion of identity, gender, and ethnicity are powerful forces acting on all of us.

Therefore Solidarity Rituals to help teams establish their individuality and pride in their work. Celebrate Success when a team passes a major milestone and remember important events as a team.

How. The challenge here is creating a new subculture. Cultures are characterized by common values, world views, behaviors, and language. Creating the outward appearance of a culture is the first step.

- The importance of a common space cannot be overemphasized. Begin small, e.g. an owned whiteboard and a bit of aisle space among the cubes, if necessary but push hard for a dedicated room.
- Common attire is important. It is frequently easier to use jewelry or distinctive id badges rather than clothing (people do not want to wear the same T-shirt everyday).
- Schedule common meals. Use food to mark important milestones or points where the team needs to resolve issues among themselves. Precede retrospectives with a potluck lunch.
- Engage the team in activities other than work – a reading group for instance, a movie night, a Friday afternoon picnic.

5. Participant Observation

Sketch: *"Hey Samantha, where were you last week?" asked Henri. "Back in the trenches over at Facilities Management," she replied. "I used to work in an area like that before becoming a business analyst. They had a crisis and needed someone to help our for a few days." "I bet that was awful," Henri sympathized. "At first," Samantha admitted, "but then I noticed that I was really gaining a much better understanding of the requirements for this new system we are building for them."*

Context Your team, and your entire organization, is organized based on roles and roles within product lines. Even though you are assembling a team that includes most of the roles, customer to coder, communication is still role-to-role with lots of interpretation across the communication channel.

Therefore. Anthropologists long ago discovered that true understanding requires more than talking and observing. They invented the term, "participant observation," to describe how they gather the information and understanding necessary to write a great ethnography. Live with, eat with, work with, and empathize with the other members of your team.

- Pair programming is only the beginning – do some pair testing, pair story writing, pair database design, pair forms entry. Have all of your people work alongside all of your other people.
- In addition to working with your on-site customer, pair for a day or two in the actual business unit where your software will be used.

6. Conclusion

So here we are - with a collection of Agile Practice Adoption Patterns that all address the Us vs. Them smell found in many organizations. The concrete practices we have are 3 levels removed from the smell; that is we have Reciprocal Visibility which in turn references other patterns such as Dynamic Information Radiator, which references the concrete practice of Daily Stand-Up.

We envision new adopters of agile processes going to these patterns driven by the smells they need to address at their particular organization and upon locating the concrete patterns being able to get advice on how to adopt them. Currently most of the literature is about the pattern itself not how to go about adopting them.

Finally, we see that the full pattern language is very large and requires much more work to flesh out. One of the issues we are currently grappling with is taking subsets of this language that make sense as a cluster instead of just arbitrarily digging deep as we have done here.

7. Conclusion

So here we are - with a collection of Agile Practice Adoption Patterns that all address the Us vs. Them smell found in many organizations. The concrete practices we have are 3 levels

removed from the smell; that is we have Reciprocal Visibility which in turn references other patterns such as Dynamic Information Radiator, which references the concrete practice of Daily Stand-Up.

We envision new adopters of agile processes going to these patterns driven by the smells they need to address at their particular organization and upon locating the concrete patterns being able to get advice on how to adopt them. Currently most of the literature is about the pattern itself not how to go about adopting them.

Finally, we see that the full pattern language is very large and requires much more work to flesh out. One of the issues we are currently grappling with is taking subsets of this language that make sense as a cluster instead of just arbitrarily digging deep as we have done here.

8. Vision and Future Work

The vision we have for this work is to build a community where these and other patterns come together and are available for those who are on their way to becoming agile. At a more immediate scope our aim is to get a few patterns in a useful "cluster" and tie them to related business values and smells. This paper is a step towards writing this "cluster", and as we get feedback (from the Pattern Language Community) we will be able to put this information in a useful format for the community. The feedback we have received from writing this paper will be used in putting together a more readable, consistent, complete, and hopefully useful pattern language targeted towards those who are adopting one or more agile practices[5].

9. Acknowledgements

Dave and I took the ChiliPLoP 2006 work and refined it to present at PLoP 2006 where it was reviewed yet again by another group. Special thanks to Ademar Aguiar for taking the time and effort to shepherd our work for PLoP. Linda Rising and Mary Lynn Manns have also read early versions of this work. Richard Gabriel lead the workshop that reviewed this work, the reviewers were Donald Little, Rebecca Rikner, James F. Kile, Till Schümmer, Lise B. Hvatum, Joseph Bergin, and Guy Steele.

10. REFERENCES

[1] Bartlet, E., *And The Agile Survey Says…*, Agile Journal, http://www.agilejournal.com/content/view/29/43/ , 2006.

[2] Beck, K,. *Extreme Programming Explained: Embrace Change*, Boston, MA, Pearson Education, 1999.

[3] Beck, K. and Andres, C. *Extreme Programming Explained: Embrace Change (2nd Edition)*, Boston, MA, Pearson Education, 2004.

[4] Bergin, J., *Patterns for Agile Development Practice, Part 1*, presented at EuroPLoP 2005.

[5] Bergin, J., *Patterns for Agile Development Practice, Part 2*, presented at EuroPLoP 2006.

[6] Bergin, J., *Patterns for Agile Development Practice, Part 3*, to be presented at PLoP 2006.

[7] Coplien, J. and Harrison, N., *Organizational Patterns of Agile Software Development*, Upper Saddle River, NJ, Pearson Education, 2005.

[8] Elssamadisy, A., Elshamy, A., Johnson, A. and West, D., *Patterns of Adopting Agile Development Practices Workshop*, ChiliPLoP, Phoenix, AZ, http://agileprocessadoption.com/wiki/index.php?title=ChiliPlop_2006_Results , 2006.

[9] Elssamadisy, A. and Elshamy A., *Patterns of Adopting Agile Development Practices Workshop*, XP 2006, Oulu, Finland, http://agileprocessadoption.com/wiki/index.php?title=XP2006_Patterns , 2006.

[10] Manns, M.L. and Rising, L., *Fearless Change: patterns for introducing new ideas*, Boston, MA, Pearson Education, 2004.

[11] Schwaber, K. and Beedle, M., *Agile Software Development with Scrum*, Upper Saddle River, NJ, Prentice Hall, 2002.

[12] Yip, J., It's Not Just Standing Up: Patterns for Daily Stand-up Meetings, to be presented at PLoP 2006.

[5] As of July, 2008 this work has been expanded into two books, *Patterns of Agile Practice Adoption: The Technical Cluster*, and *Agile Adoption Patterns: A Roadmap to Organizational Change.*