XP Culture: Why the twelve practices both are and are not the most significant thing

Hugh Robinson & Helen Sharp
Centre for Empirical Studies of Software Development
Department of Computing, Open University, Milton Keynes, MK7 6AA, United Kingdom.
+44 (0)1908 652681, h.m.robinson@open.ac.uk
+44 (0)1908 653638, h.c.sharp@open.ac.uk

Abstract

XP emphasises underlying values as well as the more visible twelve practices. In this paper we explore the relationship between practices and values from two perspectives: empirical and theoretical. We present empirical evidence that the twelve practices create a community in which the XP values are supported and sustained. We also present models of culture from other domains which suggest that an alternative set of practices can produce a community with the same underlying values. We conclude that the twelve practices are both significant and not significant.

1. XP practices and values

The twelve practices of eXtreme Programming (XP) offer an accessible and apparently straightforward prescription. Each practice is simple to understand and speaks directly to the reality of working life. The very simplicity of the practices contributes to the radical nature of what is proposed. The simplicity is beguiling enough to invite unthinking objections to what is proposed, a fact that was creatively played upon by Scott Adams in the short XP episode in the Dilbert strip in January of 2003. (http://www.dilbert.com/comics/dilbert/archive/dilbert-20030109.html). Together, the twelve practices are seductive: a code by which to live working life.

At an OOPSLA 2002 panel [1], the importance of XP's twelve practices was discussed. A number of the panellists (most notably Kent Beck) gave the view that the practices were not important in their own right but rather as a manifestation of an underlying approach. Beck stated "Practices are superficial, there's something deeper". There is considerable wider debate about the twelve practices, and whether all of them are necessary, whether any are missing, etc. (e.g. [2]).

But there has always been more to XP than the twelve practices. A skim through Beck's seminal text [3] shows that the twelve practices don't even get listed until almost a third of the way through. And even then the chapter headings don't neatly align. A closer reading shows that, before listing the twelve practices, Beck quite explicitly sets out the four values of XP – communication, simplicity, feedback and courage. These four values drive XP since they give rise to a dozen or so principles and it is these principles that are more directly embodied in XP practices. He goes further by suggesting an underlying core value which informs the four, "a deeper value, one that lies below the surface of the other four – respect. If members of a team don't care about each other and what they are doing, XP is doomed." [3, p 35]. Interviewing Beck, Highsmith [4] observes that his "important vision is about changing social contracts, changing the way people treat each other and are treated in organizations" and quotes Beck's response to an article that attempted to revise XP: "I was furious that someone would strip out all of the social change and still call it XP". This emphasis on the importance of the underlying culture as well as technical practice, is not unique to XP within agile methods: Cockburn [5, 6], for example, underscores the importance of culture, as does Highsmith [4]. This wider context of values and culture also raises issues, and at XP2003 there was an explicit panel debate on whether values are more important than practices, or vice versa [7].

Against this backdrop of debate about practices versus values and the importance of culture, we address two questions in this paper: *do the twelve practices of XP give rise to a culture that embodies the explicit values of XP?* and *are these the only practices that can produce such a culture?* To answer these questions, we present empirical evidence that the *specific* twelve practices suggested by Beck do create a community that supports and sustains the four values, and we provide theoretical evidence from other disciplines that *alternative* practices

Proceedings of the Agile Development Conference (ADC'03)
0-7695-2013-8/03 $17.00 © 2003 IEEE

can also create a community based on the same values. Hence, we argue that the twelve practices both are and are not the most significant thing.

2. XP Practices Support the Values: empirical evidence

Here we present findings from an empirical study of a small company developing Web-based intelligent advertisements. The empirical study uses a participant observer approach based on ethnography. Ethnography [8] comes from anthropology, although it is now an accepted approach within sociology. It is a broad-based approach in which researchers observe members of a culture without prejudice or prior assumptions. They immerse themselves and participate in the culture and the business of those being observed — joining in conversations, attending meetings, reading documents etc. During this, all activities and conversations are considered 'strange' so as to minimise the researchers' own backgrounds affecting their observations. It is an approach that forces researchers to attend to the taken-for-granted, accepted, and un-remarked aspects of practice. Subsequently they report, rather than interpret, what they find. Ethnography insists on the primacy of empirical data and attempts to minimize the pre-conceptions and cultural baggage of the investigator.

2.1 The Company

The company was started in 1999 and the team explicitly adopted XP from the beginning. They have speculated that they may be the longest running XP team, they were mature in their use of XP and followed all twelve XP practices [9]. At the time of the study, there were eight developers in the team, one graphic designer and one person who looked after the infrastructure. The company employed four marketing people who wrote the stories for development in collaboration with clients. Marketing people were regarded as being, in effect, the customer.

The following is an edited extract from the field notes describing the physical layout of the study: We describe it in some detail because this provides an overview of the context of our study, and much of the observed behaviour orients to the physical setting. In a sense the physical setting is symbolic of the culture.

The office was situated on the first floor of an old building in north London, UK. The organisation of the physical space within the office was significant since the practitioners' work oriented to this organisation. The office was a long rectangular shape with a walkway through the middle. At the far end of the walkway was the area where the developers spent most of their time. On the right hand side of the walkway was a large area where pair programming took place, with desks shaped specifically for programmers to sit two to a machine. (See Figure 1.) This pair programming area was divided informally into two sections. The area showed few signs of personalising the workspace. The wall had a notice board which contained, amongst other things, a large organised area devoted to 6" by 4" index cards – the active story cards. Each card had a coloured sticker on it denoting its status, e.g. red to denote unfinished story, yellow to denote developer finished story, and so on. There were also four 'to-do' lists which occupied a prominent position and two of the lists had at its top a picture – one of Anton Chekhov and the other of Pavel Chekhov[1]. These 'to do' lists were checked-off to signal certain events. This part of the wall was a 'busy' area in contrast to the rest of the walls which were generally quite bare. This area is also where a daily meeting took place – known as the stand-up, since people did not sit down at such meetings. These meetings took place at the start of each day and were very short. One of the 'to-do' lists (the 'Standup Chekov') stated that 15 minutes was the maximum length. These meetings were the occasion when people chose what work they would do that day: who they would pair program with and the stories they would work on.

Figure 1 A pair programming station. The Chekhov board is immediately to the right of this picture.

To the left of the walkway was an open area. About half of this was taken up with some tables and chairs, and a large sofa. The tables were used for an intensive and relatively long (hours rather than minutes) group activity that involved both developers and the company's marketing team. This activity was the Planning Game. Behind these tables, up against the wall, was a shelf full of 'tacky gifts' which people had brought back from their holidays. It was a tradition that each member of staff would bring the tackiest gift they could find, to add to the collection. To the right of these tables was the 'release' machine. This machine was used to release modified and tested code into the main

[1] Anton Chekhov – the Russian playwright and Commander Pavel Chekhov – the navigator of Star Trek's USS Enterprise.

system. On top of this machine was a small box with a picture of a cow on it. Picking up the box and tilting it produced a 'moo' sound, like a cow. Each time some code was released, a developer picked up the box and made the box 'moo'.

Opposite the developers' area was a stand-up bar of monitors. These were used for checking the status of the live servers, running tests on different target platforms and other activities such as personal email and Web surfing. These machines were not used for pair programming. Behind this wall of screens was a well-equipped kitchen. The kitchen had a homely and personalised feel to it, and people would happily bring their own food and leave it in the kitchen. One member of staff regularly ate his breakfast in the kitchen when he arrived. Sharing food such as cakes and biscuits was a very important part of working life. Between the developers' area and the meeting rooms was workspace for the technical support person and the graphic designer. Although these two people did sometimes pair program with developers, they had a separate desk space outside the enclosed developer area.

The walls of the developers' area were generally quite bare; they were only adorned with material of a practical nature, for example phone numbers and a list of the twelve XP practices.

Towards the other end (the main door end) were the company's marketing team. They sat at standard rectangular desks, facing each other.

2.2 Themes within XP culture

All twelve practices were observed but we don't intend to give a blow-by-blow account of each practice. Rather, we'll give our findings in terms of five themes which emerged from our analysis of the data – from our attempt to regard what was observed as strange and ask the question 'what's going on here?' These themes cover a diversity of activity and cut across practices.

2.2.1 Shared purpose, understanding and responsibility. A key feature of all the activity which we observed was that it exhibited evidence for a shared purpose, understanding and responsibility within the team. By that we mean that what work needed to be done was discussed, decided and agreed in a shared fashion, the detail of how that work might be executed was similarly a shared negotiation by the team, and responsibility for ensuring that the execution was satisfactorily carried out was collective.

The Planning Game embodied all the elements of this theme, with a number of interwoven activities: designing, estimating and planning. A huge amount of discussion happened and a lot of 'what if's' were considered. The Planning Game we observed took up a large proportion of the first three days – an unusually long time, as commented on by many of the developers. By the end of it, a set of story cards to be implemented during the next iteration had been identified, rough designs for how these might be implemented had been decided upon, and each card had been estimated.

Towards the end of the Planning Game, there was a clear sense of 'wanting to get on with the real work'. This was not frustration. It was a reflection of the fact that everyone understood what was agreed and needed – and was ready to get on with it.

Choosing the cards and estimating them was a group activity in which everyone had an equal chance to comment, and everyone understood the final decisions. This communal, collective approach has advantages for maintaining a shared purpose and understanding, but also has disadvantages, as commented on by one of the developers: *"The way we do estimation here is to do it all as a team which has some benefits in that we all have the same vision but it also can go on for a very long time."*

This theme was also evident in the daily stand-up meeting.

The 'stand-up' was the daily developer meeting. This was the first activity of the day. On days when the focus was on the Planning Game, the stand-up was very short. On other days developers picked up the story cards they wanted to work on, chose a partner to work with, and went off to get on with the real work. People went away with the responsibility to provide a working software solution for a story which they would self-manage and self-organise into detailed technical tasks.

During this meeting, developers shared their experiences of the day before, and decided who the pairs would be and what cards each pair would work on. Everyone was kept up-to-date with changes, challenges, and progress. There was no sense of one person 'handing out' the work and no-one was treated as though they had specialist skills that demanded they work on a particular story or pair with a specific other person. Each individual volunteered to work on a particular piece of the system, and each trusted the others to work on their chosen story card. Pairs would rotate so that it was unusual for a pair to stay together for longer than one day. This meant that everyone paired with everyone else at least once during an iteration. It also meant that everyone worked on a large portion of the code base in each iteration. This rotation helped to enhance the shared understanding and reinforce the shared purpose. Everyone in the development team pair programmed, including the graphic designer and the infrastructure person.

Shared purpose, understanding and responsibility comes with communication – lots of communication. We've already indicated some of this with the face-to-face discussions of both the Planning Game and stand-up. In fact, the developers spent most of their time communicating, one way or another. At an obvious level,

pair programming is about producing code but it is also crucially about communicating, sharing and agreeing understanding.

Shared understanding was further facilitated by the layout of the pair programming area which allowed pairs to overhear discussions in another pair. The importance of peripheral awareness has been reported elsewhere [10], and there were examples during our study where one pair overheard another pair and joined in their 'meeting'.

Communication was mostly face-to-face; to use Cockburn's phrase, there was an oral tradition [11]). However, a written tradition also figured in the pervasive use of cards. For both of the meetings described above, cards were a central part of the activity. They were used in many different ways, and for many different purposes. In line with XP principles, there was no unnecessary activity devoted to documentation. During development, the only such documentation observed was that of cards. This kind of documentation was minimal, acting as a prompt rather than being a detailed account, and often transient, i.e. having sketched an idea and discussed it, the card was either ripped up or folded and put in the bin. This was true of story cards, testing routines, estimates, anything written on index cards that had outlived its usefulness. No external record of the rationale for a decision was kept. This emphasised the importance of developers' individual memories, and the significance of oral communication. During the Planning Game, certain decisions were made after much discussion, but no written record of this discussion was kept. So, when programming starts, a pair can (and often do) decide to implement a different design. As one of the developers said "*the lack of memory does cause problems some times, and pairs can design another solution*."

One of the problems often faced by teams working together is that they use the same words to mean different things, and different words to mean the same thing. The oral tradition with minimal, transient documentation of the developers we observed faced an extreme version of this problem and they went to some trouble to ensure that the words they used were particularly significant and meant the same thing to everyone. The vehicle for facilitating this shared understanding was one of the XP practices – that of metaphor. The metaphor these developers had chosen revolved around advertising terminology. However, their business product was evolving and they felt that it was time to review the metaphor. This was not simply bowing to XP doctrine – they clearly saw it as important.

> *On the Friday the stand-up meeting was followed by a group design meeting. Two main issues were discussed: whether to freeze an existing product that is no longer being developed; and the new code metaphor. A new metaphor was needed because the business had changed and the names of portions of code no longer reflected what the software was about. The developer group spent around 15 minutes discussing the names to be given to slices of code. They clearly believed that names are important, and keeping the integrity of the code metaphor was significant. Doing so was "onerous but important".*

In the practice we observed, metaphor was regarded as a fundamental facilitator for maintaining the shared vision.

We were struck by the commitment and enthusiasm of the team. The shared purpose, understanding and responsibility that we've characterised applied to individual and team. There was no sense of conflict or tension between individual and team; neither seemed ever to need to subvert their wishes to the other. The behaviour we've described was not forced but had become part of everyday life, although the need to preserve it was also recognised when one of the developers was asked the skills he valued in this way of working:

> *The willingness to take responsibility, just to get on with things, not to be told by somebody else that you have to do certain things. It's very much about getting on with it. People see gaps and go to fill them.*

2.2.2 Coding and Quality of code matters. Code and coding mattered to the XP team we observed. The following quote appears on an XP developer's site dedicated to the company in our study:

> *ConneXtra is the place where you ring up and say "can I speak to X please?" and they say "sorry, he's in a meeting" and then you say "do you mean he's programming?" and they say "yes" AND HAVE NO SHAME.*

Coding was regarded as a supremely important activity that should not be interrupted. This was reflected in the layout of the developer area which was partitioned from the main 'public' areas such as the kitchen and the Planning Game communal area. During the course of the study, developers were observed to move freely around other areas of the office, including visiting the marketing people, but we did not observe the marketing people entering the pair programming area. If clients required attention during an iteration, the responsible sales person could approach a designated contact pair (the exposed pair, as we discuss below), but no other interruptions were observed.

Keeping the company's clients happy was a pragmatic necessity, but one which was taken seriously. Each day an 'exposed pair' was identified: a pair of developers who could be interrupted if a client had an urgent request.

> *At the second meeting ,,, It was reported that generally customers are supportive of the XP approach because they had fast turnaround to issues. However when urgent changes arrived in the middle of an*

iteration it became harder to react in an acceptable amount of time. There was a sense that during an iteration the developers expected to be left alone to get on with development. Interruptions from clients wanting faster changes disrupted the flow of development. To overcome this each day the developers identified an 'exposed pair' who would be assigned to handle any such interrupts.

But it's not just the pressure of pragmatics that underlies a desire to produce quality code, a desire that was alluded to in different ways throughout the study, at higher and lower levels of detail.. During one pair programming observation, the frustration shown by the developers who wanted to refactor the code, but were working on a story card that didn't include an estimate for refactoring, was significant. The problem was exacerbated because the code base supported more than one product and refactoring would have involved making decisions about the older product which involved business strategy decisions not just coding ones. The pair reluctantly agreed to write a task card for refactoring this section of the code, and to focus only on the card in front of them. Their disappointment and frustration was palpable. This suggests that sometimes there can be a tension between pragmatic considerations and the desire to refactor and have the simplest code.

During design discussions, issues and alternatives were carefully and thoroughly considered: *"there were a lot of 'what ifs' considered during these sessions"* (Planning Game). Design and re-design to improve the code base was a way of life:

Design occurred in different places and at different times. The planning game included some design, documented on story cards and then often discarded once it had been discussed and a decision made. This kind of design could be at the architectural level or at a lower code level. For example, talking about how many lines of code a certain story would require, and even mentioning the commands that could be used to achieve the desired functionality. Whatever was needed to make an assessment of the length of time required to complete the story. This design was re-thought and sometimes changed during actual coding

During programming, design took place at a low level, i.e. in the code, but pairs also took responsibility for the whole code base and wanted to re-design it when they saw complex code. There was a genuine desire to 'do the simplest thing', a phrase that was repeated during the Planning Game but was also evident in this desire to refactor code and to find simple designs that worked.

The maxim of 'test first' was executed without comment:

In the Java pair programming session, I didn't realise to start with that the developers were writing the test and the code together. They moved seamlessly from one to the other as they understood better what changes the story card required.

When a new piece of code was released, the developers responsible would announce their success by making the cow box 'moo'. This both heralded the production of quality code (it had passed all the tests), and alerted other developers to this significant event. Code was not released into the main system after about 5pm because the biggest problems have been caused by releases at that time of day – an observation that recognises pair programming is tiring.

2.2.3 Sustainability. As well as caring about the quality of code, the team cared about quality of life. This manifested itself most strongly in the calm atmosphere. Even during the Planning Game there was no heated discussion. When disagreements or conflicts were identified, decisions were taken only after careful weighing of risks and other factors. A blue stress ball sat on the Planning Game table, but was rarely used for the duration of our study. During the retrospective, which traditionally can cause people to 'lose their cool', the developers had instigated a fun 'referee' in the shape of a toy dog that barked when shaken. The protocol said that you had to hold the dog while you spoke, but the dog mustn't bark. This resulted in the speaker having to move his/her hands and arms only in a calm manner, and calmness was reinforced because when holding the dog, people tended to stroke it.

Regular and communal breaks were taken in the morning, afternoon and at lunchtime. During the week-long study, all breaks were taken together, but once coding started only lunchtime was taken together. These regular breaks were perceived as being important to one degree or another by all members of the team.

On Tuesday, the Planning Game was still going on at 5.50. People had tried to stop but Tony kept the discussion going. Then Ian walked away and put on his coat. Gradually others drifted away too… Ian was always reminding people to take regular breaks by asking them when they last had one. Non-one was ruffled by this 'nagging' which appeared to be taken positively.

Although the office was always emptied by about 6 (on the observation days), there was evidence that the work sometimes continued after that time. For example people attending the eXtreme Tuesday Club (a weekly London-based meeting open to all interested in XP) and adjourning to a local bar for a chat after leaving the office. Not everyone took part in these activities, and there didn't appear to be any social pressure to do so.

One significant element of quality of life is fun! The developers' love of fun was evident in a number of ways. For example the tacky gifts that people brought home from holiday, and the use of the 'moo' box to signal code

release. The tradition of bringing in food to share also indicated an emphasis on having a good time.

During the afternoon, Peter produced a packet of Cadburys mini-eggs. He said that we could all have one for every good idea we had…. After about 15 mins without anyone having an egg, we decided that we could have an egg whether we had any good ideas or not!

We've mentioned already that there were no signs of conflict or tension between individual and team. But there was a recognition that the emphasis on shared development and quality of code can be intense and that there was a need to reflect this with some activity that focussed solely on the individual. To that end, two days a month were given to each individual – known as 'gold card' days – where one could carry out some individually-focussed work that was of value to the company.

Our observations here shouldn't be regarded as simply evidence that the team followed the 40-hour week practice. It was a more subtle and accomplished achievement than the mere following of a prescription. Each individual was in control of the divide between work and home and seamlessly crossed the divide repeatedly. This ability to both set the divide and to move across it repeatedly was evidence for their shared ownership of the work product and of control over how the work was achieved. The end result was that of making development sustainable in its human dimension.

2.2.4 Harmonious rhythm. The team atmosphere that we observed was one of calm, competence and confidence, embodied by a pervasive and harmonious rhythm. This rhythm operated at a range of levels and we'll concentrate on two that emerge from the data: one was a daily rhythm and one was a rhythm oriented around the three-week iterations. These rhythms were marked by events that signalled openings and closings and were punctuated by other events which signalled progress.

For example, the daily rhythm began slowly as people arrived, but the 'real' day did not begin until the stand-up meeting:

When developers arrived in the morning (any time up to about 9.30), they engaged in various activities such as eating breakfast, checking email, and reading the newspaper. When most people were present, there was a stand-up meeting to start the day. It felt as though this meeting heralded the real start of the day. After the stand-up everyone went off to start the agreed tasks, either to continue with the Planning Game or to program.

The daily rhythm began with the stand-up where tasks were chosen, progressed through pair programming – guided by the shared metaphor and peripheral awareness of other pairs to share understanding – and would be punctuated by breaks, lunch, and the release of code. People actively supported this rhythm. The progress made by release of code was publicly signalled by making a 'moo' – the 'moo' of continuous integration.

Even in the meetings and gatherings there was a recognisable flow. Discussions would start when enough people were present and end when enough people had left. No-one called these developer meetings to order.

The closing of the day was not marked by so explicit an event as with the opening and the stand-up. It was typically a more low-key version of what we've described above for the Planning Game: over a period of 30 minutes or so people left. There was no sense of a signal being given that this was appropriate and the order in which people left varied over the days on which it was observed.

As each day passed, the story cards progressed through their lifecycle of different coloured stickers, reinforcing the rhythm of progress under the benevolent gaze of the two Chekhovs. In a sense the two Chekhovs, with the story cards on the wall, orchestrate what's been done and what needs to be done, noticing the 'moo' and moving on through the score. This orchestration via the Chekov board was public and visible. Each developer had a responsibility, to themselves and to others, to check progress, to maintain progress, and to shout if progress wasn't happening.

Whilst the atmosphere was relaxed, much was happening: daily working life had plenty of 'busy-ness' but that 'busy-ness' was distinctly not hectic or frenetic haste. For example, the end of the day simply came (as night follows day, so to speak) and there was no sense of frustration, such as might be associated with missed targets in a rigid schedule. Within this busy-but-relaxed atmosphere, people carried out tasks that were 'do-able': no one was faced with doing something that they had not chosen to do, did not understand or did not have the resources and help to carry out. Tasks were just as challenging though, and the team showed real courage in particular decisions. For example, at one point it was decided to use the PYTHON language for a significant element of the system. Only one person in the team knew the language, but the response to this was not "well, we'd better use something else" which might be found elsewhere, but instead "we'll go and buy some books on it". And they did, and produced successful PYTHON code.

This daily rhythm sat within a longer three week iteration which provided an over-arching rhythmic score: the opening was marked by the Planning Game and setting up of the iteration's wiki page; and the ending of an iteration was marked by completing its wiki page, and usually (though not always) by a retrospective. An iteration was rhythmic rather than a set of deadlines that needed to be hit.

2.2.5 Fluidity. The final theme we report on runs through a range of activities and concerns boundaries – boundaries of various sorts but boundaries which were fluid and

whose fluidity was an organisational element of activity and behaviour rather than a constraint on activity and behaviour. Boundaries were there to be crossed rather than to be adhered to.

We went to some lengths to give a detailed description of the physical setting, suggesting that it was symbolic of the culture. The physical setting is open plan: open and public to all in the team. Yet within this open plan are the boundaries of the various areas: the kitchen, the area for meetings, the pair programming area, the marketing area, and so on. The walls have boundaries too – mostly of unadorned space, but the area of the Chekhov board is special and bounded. The physical bounding of space is symbolic of the working life we observed. It is shared & open. It has an area dedicated to that which matters above all: the creation of quality code (and a special place for the culmination of that act of creation with the release of code). With the homely kitchen, it provides a separation and transition between work and non-work. The Chekhov board ensures that the space acts in concert – the rhythmic harmony of life.

These physical boundaries can be crossed as part of everyday life, although, in the case of the pair programming area, not everyone can cross a boundary with the same ease. Developers went to marketing but the reverse did not happen. Marketing people would come to the Planning Game area. They respected the 'privacy' of the pair programming area and left developers alone to concentrate on producing code. Urgent calls from clients were addressed by having the exposed pair. This boundary shouldn't be seen as some evidence for a caste system where 'developers' had some hegemony of status over 'marketing' (the customer). Rather it was a recognition of the boundaries of different roles and responsibilities. There were clear responsibilities given to marketing people to communicate with the client, to prioritise stories and to advise on a client's requirements. They did not comment on technical story cards and their prioritisation. Conversely, developers accepted marketing's decisions on a client's needs and priorities. The pair programming area was simply an area where an activity took place whose intensity and significance needed to be respected.

Boundaries other than those of physical space also were present. Within the developer team, boundaries were very fluid. Individual developers did not have permanently assigned roles and would move across all aspects of the current development – something which is reflected in (and reflected by) the fluidity of the code base. Boundaries between pairs in pair programming also existed but were fluid and could be crossed without any great negotiation as peripheral awareness led to wider discussion as necessary. Pair programming would include the graphic designer as well. Boundaries between work and home existed, but were fluid and were within the control of the individual. Similarly, technological boundaries existed in the sense that the team had awareness of their technical competence and knew when they might be moving outside that competence, but such a boundary was one to be consciously crossed – as in the PYTHON case, discussed above.

2.3 Summary of findings

We were struck by the fact that the XP developers we observed were clearly 'agile', both individually and as a team. This agility seemed intimately related to the relaxed, competent atmosphere that pervaded the developer group. This team created a self-managing, self-organising community with a culture that emphasised shared responsibility. There was a rhythm to life that enabled people to organise their work tasks in a way that gave them common ownership of the work product and control over how the work was achieved. Being in control could mean going home at 5 pm and it could mean going on to the eXtreme Tuesday Club. Their rhythm was comfortable and relaxed, yet purposeful and productive.

We would summarise our findings by saying that an XP community has the following four characteristics:

1. Both individuals and the team are respected;
2. Both individuals and the team take responsibility;
3. Both individuals and the team actively encourage the preservation of the quality of working life;
4. Both individuals and the team have faith in their own abilities to achieve the goals they have set themselves, which is constantly re-validated & re-affirmed.

The first of these reflects Beck's deeper value of respect but we report on it separately here because we want to emphasise the importance of both the individual and the team.

The account above describes a community that clearly values communication, feedback, simplicity and courage. These values were deeply embedded in the community. For example:

1. No-one on the team said to us "I value communication", but *communication* happened freely, as a matter of everyday life, and there was no resentment, reluctance or coercion for people to communicate;
2. Developers voluntarily approached the customer to obtain *feedback* and to clarify issues when it was appropriate.
3. The frustration that emerged when some refactoring to *simplify* existing code had to be postponed indicates that "do the simplest thing" was not just rhetoric.
4. The team showed *courage* by taking on an unknown language, PYTHON, and through the level of re-design they proposed.

Hence, the *specific* twelve practices suggested by Beck create a community that supports and sustains the four XP values, plus the value of respect.

3. Alternative practices can support the values: theoretical evidence

In this section we introduce some tried and tested models of culture from other domains, and we use them to view the relationship between XP practices and values from a different perspective.

3.1 Models of culture

'Culture' is a slippery word with a range of meanings. As [12] observes: "The trouble with culture is that no one is quite sure what culture is", describing a literature survey that produced 171 definitions of culture, sortable into thirteen categories! We're not trying to survey such complexity in this paper. Instead we want to clarify what we mean by culture and what are the components of culture.

Sociologists and anthropologists view culture as the means by which people "communicate, perpetuate and develop their knowledge about attitudes towards life. Culture is the fabric of meaning in terms of which human beings interpret their experience and guide their actions." [13]. That is, culture frames the world, what matters in the world, including how problems and solutions are perceived and how people behave and perform.

Culture in this sense is something which is learnt. The learning of a culture can take place in a semi-formal way that makes membership of the culture open and accessible to a wide range of individuals. For example the culture of software development can be learnt. In contrast, if the learning of a culture takes place during infancy, childhood and adolescence in a pervasive, prolonged and unconscious fashion then it is hard, if not impossible, for an outsider to truly learn and be part of the culture. For example, neither of the authors of this paper can learn Balinese peasant culture because they were not brought up in a Balinese peasant society.

Business, organisational and management studies share this concept of culture from sociology and anthropology, providing a more instrumental and structured account which focuses on organisational culture. Such studies (e.g. [14-16]) propose layered models of organisational culture. Whilst the detail varies, typically there are three layers: behaviour and actions, values, and basic assumptions. Behaviour and actions are directly observable, values less so, and insight into basic assumptions can only be gained indirectly. An analogy with a lily pond is given in [16] (see Figure 1), so that behaviour and actions are like the lily pads floating on the surface, and values and basic assumptions are like the more hidden stems and roots beneath the pond's surface. Importantly, the relationship between the layers is not one way (where, say, beliefs beget attitudes & values and these in turn beget behaviour) but is two-way: culture is "both the product of an action and a conditioning element of future action." [16]. Culture is created and sustained by the actions of people in that culture and culture shapes and frames the actions of people: the relationship between the layers is reflexive.

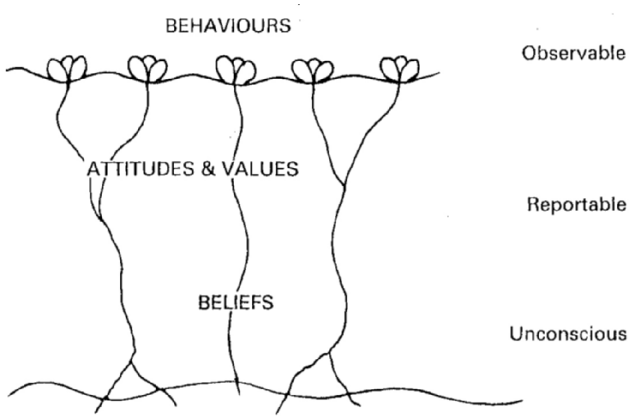

Figure 1 Culture and Behaviour depicted as a lily pond [16]

3.2 XP culture

Our examination of culture gives us a model in which to place the relationship of the twelve XP practices to the underlying values. Comparing them to the lily pond model and its terminology, we suggest that XP practices correspond to observable behaviours, while XP principles lie below the surface and are reportable, and XP values lie even further below the surface and correspond to unconscious beliefs. We can also see that XP's insistence on the importance of the underlying values is well-founded in terms of these cultural models: the values will shape the principles which will frame practices. Practices are important as well since they reflexively create and sustain the values.

The layering approach of the model could be taken as rigidly deterministic, i.e. that the observed behaviour and actions are the *only* behaviour and actions that can create and sustain the underlying culture. However, as we've emphasised, that's not what is intended: culture *shapes* and *frames* the *possibilities* of actions and behaviour; actions and behaviour *orient* to deeper values, but they're not rigidly governed by them. For example, [13] talks of culture as being that by which humans "*guide* their actions" and [16] speaks of culture as that which "*influences* the goals, means and manner of action." (our own emphasis).

This is an important point which provides a different perspective on the relationship between XP practices and values. Because the models are not rigidly deterministic,

they imply that the twelve specific practices are not significant since an alternative set of practices can produce a community that supports and sustains the same set of values.

4. Conclusions

At the beginning of this paper we asked two questions: *do the twelve practices of XP give rise to a culture that embodies the explicit values of XP?* and *are these the only practices that can produce such a culture?* To answer the first, we've reported on an empirical study of XP practice where the specific twelve practices of XP have created a community that supports and sustains a culture that includes the XP values.

To answer the second question, we've discussed models of culture from other domains that suggest that the specific twelve practices can be varied and still produce the same underlying values. So, if it is not possible – for whatever reason – to adopt all twelve practices, this evidence implies that a modified, different set of practices might produce an appropriate culture and so may be perfectly valid. We've a hint of this in the company we've studied. Arguably, they are not following the 40-hour week practice. We didn't see any evidence of individuals keeping any kind of record of hours worked or there being any sense of work beyond a certain point in the day being termed 'overtime'. Individuals had a flexible and fluid approach to hours worked, and to the need for breaks during working hours that ensured that the deeper purpose of sustainability was achieved.

Models of culture have also given insight into the nature of the relationship between practices and values: a reflexive relationship where practices create and sustain values and values support, shape and frame practices. The nature of this relationship answers any debate over which is more important. You cannot have practices without the values that support, shape and frame them – and you cannot have the values without the practices that create and sustain them. Attempts to artificially graft practices onto values in the absence of this reflexive relationship are simply dysfunctional, as with Cockburn's account of 'hostile XP' (practices with the wrong values) versus 'friendly XP' (practices with the right values), where the difference is profound [6, pp 103 - 4]).

These conclusions concur with the so-called 'XP Maturity Model' (reported by Alan Francis at XP2003, quoting Kent Beck) which has three levels:

Level 0: follow all 12 practices all the time

Level 1: modify the practices to suit the environment

Level 2: it doesn't matter what you're doing as long as it works

The practices are significant when developing and building a community with the appropriate culture (Level 0). However, once the community is established and the values are embedded in the culture, then the practices can be modified. Eventually, an alternative set of practices can be found, if required, because the community's culture is so strong and the values so deeply embedded. However, an alternative set of practices must be such that they sustain the values and they must be practices that the values can shape and frame – the relationship between the two is reflexive.

So the specific twelve practices are significant because we know that they can create a community with the right values. However, theory tells us that they are not necessarily the only set of practices that can create a community that sustains and supports those values.

Acknowledgements

We are indebted to the staff of Connextra for their co-operation and patience.

References

[1] OOPSLA, "To Be Extreme or Not to Be Extreme?", Panel session flyer, Thursday November 7, 2002, OOPSLA 2002, Seattle, WA., 2002.

[2] XP2002, "Xtreme Programming: The 13th Discipline or the Missing Practice(s)?", Panel session at XP2002, Tuesday May 28, 2002, Alghero, Sardinia, Italy, 2002.

[3] K. Beck, *eXtreme Programming Explained: embrace change*. San Francisco: Addison-Wesley, 2000.

[4] J. Highsmith, *Agile Software Development Ecosystems*. San Francisco: Addison-Wesley, 2002.

[5] A. Cockburn, "Software development as community poetry writing." Available at http://members.aol.com/acockburn/papers/swaspoem/swpo.htm: Invited talk given at 1997 annual meeting of the Central Ohio Chapter of the ACM, 1997.

[6] A. Cockburn, *Agile Software Development*. Reading, Massachusetts: Addison-Wesley, 2001.

[7] XP2003, "XP Practices versus Values?", in *Extreme Programming and Agile Processes in Software Engineering*, Lecture Notes in Computer Science 2675, M. Marchesi and G. Succi, Eds. Berlin: Springer, 2003, pp. 455 - 458.

[8] M. Hammersley and P. Atkinson, *Ethnography: principles in practice*, Second ed. London: Routledge, 1995.

[9] T. Mackinnon, "XP - call in the social workers", in *Extreme Programming and Agile Processes in Software Engineering*, Lecture Notes in Computer Science 2675, M. Marchesi and G. Succi, Eds. Berlin: Springer, 2003, pp. 288 - 297.

[10] C. Heath and P. Luff, "Collaboration and control: crisis management and multimedia technology in London Underground line control rooms", *Proceedings of CSCW'92*, 1992.

[11] A. Cockburn, "Balancing lightness with sufficiency", *Cutter IT Journal*, vol. 13, pp. 26 - 33, 2000.

[12] C. Geertz, *Available Light: Anthropological Reflections on Philosophical Topics*. Princeton: Princeton University Press, 2000.

[13] C. Geertz, *The Interpretation of Cultures*. New York: Basic Books, 1973.

[14] E. H. Schein, *Organizational culture and leadership*, 2nd ed. San Francisco: Jossey-Bass, 1992.

[15] F. Trompenaars, "Riding the waves of culture", in *Understanding Cultural Diversity in Business*, F. Trompenaars, Ed. London: Nicholas Brealey Publishing, 1994.

[16] A. Williams, P. Dobson, and M. Walters, *Changing Culture: New Organizational Approaches*, 2nd ed. London: Institute of Personnel Management, 1993.