An ethnography of XP practice

Helen Sharp and Hugh Robinson, Department of Computing, The Open University, Walton Hall, Milton Keynes, MK7 6AA, United Kingdom (h.c.sharp@open.ac.uk +44 (0)1908 653638, h.m.robinson@open.ac.uk +44 (0)1908 652681).

Abstract

Agile methods are a response to more rigorous and traditional approaches to software development which are perceived to have failed both customers and software development practitioners. eXtreme Programming (XP) is an example an agile method and we report on an ethnographic study of XP practice carried out in a small company developing Web-based intelligent advertisements. We identify five characterising themes within XP practice and discuss these findings in terms of the culture and community of XP.

1 Introduction: agile methods & eXtreme Programming (XP)

This paper reports on an ethnographic study of mature eXtreme Programming (XP) practice which offers insight into the culture and community which underlies the XP approach – an example of an agile method. We first give some background to agile methods in general and XP in particular, followed by some remarks on our motivation for an ethnographic study of XP practice. We then describe the study, report our findings, and discuss the implications of what we have found.

In the last few years, a related series of new approaches to software development has emerged, mainly as a result of practitioner-led concerns from within the object-oriented community, under the general heading of agile methods e.g., Cockburn, (2001), Highsmith, (2002). Agile methods are a response to more rigorous and traditional approaches to software development which emphasise the (perceived) importance of predictive planning, the use of appropriate processes and tools, and the need for documentation. Advocates of agile methods – *agilists* – hold that such rigorous and traditional approaches have not delivered timely, effective software that meets the needs of customers in the reality of the problems encountered by practitioners – problems characterized by change, speed and uncertainty. Agilists offer approaches which stress collaborative practices, face-to-face communication, collaboration with the customer and the importance of the individual and the team. The *Manifesto for Agile Software Development* (http://www.agilemanifesto.org/) gives some of the flavour of all agile approaches. For example, its opening statement is:

> "We are uncovering better ways of developing software by doing it and helping others do it. Through this work we have come to value:
>
> Individuals and interactions over processes and tools
> Working software over comprehensive documentation
> Customer collaboration over contract negotiation
> Responding to change over following a plan"

There are a growing number of practitioner-led conferences devoted to agile methods: the XP series (see http://www.xp2002.org/), XP Agile Universe (see http://www.xpuniverse.com/home) and the Agile Development Conference (see http://agiledevelopmentconference.com/), for example. Agile methods are attracting increasing academic interest from the software engineering community (Boehm, 2002, DeMarco &Boehm, 2002), with forthcoming issues (June 2003) of IEEE *Computer* and IEEE *Software* being devoted to agile methods and XP, respectively.

Whilst the various agile approaches (Scrum, Adaptive Software Development (ASD), Crystal, eXtreme Programming (XP), etc.) differ in detail, they all exhibit certain things in common. They all emphasise that the particular approach taken is motivated by an underlying set of values. In the case of XP Beck (2000) explicitly lists four underlying values: communication, simplicity, feedback, and courage, with a deeper value – respect – underlying the four. As well as this emphasis on underlying values, each particular approach also places emphasis on practice. The notion of practice is juxtaposed against that of process: 'Process deals with prescription and formality, whereas practice deals with all the variations and disorderliness of getting work done.' (Highsmith, 2002, pp 121-122). In a sense here, agilists are

occupying the same terrain as that charted by Schön (1983) with his analogy of the swamp as the place where messy but crucially important problems are tackled by the accomplished practitioner. In the case of XP, practice is oriented around twelve detailed *practices*. We give Beck's (2000, p 54) original annotated list of the twelve practices as Table 1, below.

Table 1 XP Practices

The Planning Game – Quickly determine the scope of the next release by combining business priorities and technical estimates. As reality overtakes the plan, update the plan

Small releases – Put a simple system into production quickly, then release new versions on a very short cycle.

Metaphor – Guide all development with a simple shared story of how the whole system works.

Simple design – The system should be designed as simply as possible at any given moment. Extra complexity is removed

Testing – Programmers continually write unit tests, which must run flawlessly for development to continue. Customers write tests demonstrating that features are finished.

Refactoring – Programmers restructure the system without changing its behavior to remove duplication, improve communication, simplify, or add flexibility.

Pair programming – All production code is written with two people at one machine.

Collective ownership – Anyone can change code anywhere in the system at any time.

Continuous integration – Integrate and build the system many times a day, every time a task is completed.

40-hour week – Work no more than 40 hours a week as a rule. Never work overtime a second week in a row.

On-site customer – Include a real, live user on the team, available full-time to answer questions.

Coding standards – Programmers write all code in accordance with rules emphasizing communication through code.

Some may contend the detail but the sense of XP practice is captured in this description by Cockburn (2000, p 29, original emphasis):

> It calls for all the developers to sit in one large room, for there to be a usage expert or 'customer' on the development staff full time, for the programmers to work in pairs and develop extensive unit tests for their code that can be run automatically at any time, for those tests always to run at 100% of all code that is checked in, and for code to be developed in nano-increments, checked in and integrated several times a day. The result is delivered to real users every two to four weeks.

> In exchange for all this rigor in the development process, the team is excused from producing any extraneous documentation. The requirements live as an outline on collections of index cards, and the running project plan is on the whiteboard. The design lives in the oral tradition among the programmers, in the unit tests, and in the oft-tidied-up code itself.

2 Motivation

Our motivation for this study is one of gaining insight into the culture and community of an agile method as part of a broader agenda of an examination of the culture and community of software engineering e.g., Low et al., (1996), Robinson, (1996), Sharp &Robinson, (2002), Sharp et al., (2000a, b), Sharp et al., (1999). That is, we seek here to understand better the values, beliefs and assumptions that inform and shape agile practice and which, in their turn, are created and sustained by practice. Similarly, we seek to explore the extent to which, for example, agile practice might constitute a community of practice with mutual engagement, joint enterprise and a shared repertoire (Wenger, 1998). Given this motivation, our methodological approach is ethnographic e.g., Hammersley

&Atkinson, (1983); an approach that forces researchers to attend to the taken-for-granted, accepted, and un-remarked aspects of practice, considering all activities as 'strange' so as to prevent the researchers' own backgrounds affecting their observations. Our data is naturally occurring insofar as it is derived from in-depth participant-observation that gave no *a priori* significance to any particular feature of practice. Importantly, our study was not intended to be any evaluation of an agile method as a software development method and the extent to which it may be viable and successful.

3 The study and its setting

3.1 The study

Our findings are based on a study of XP practice in a small company developing Web-based intelligent advertisements for paying customers. Started in May 1999, the company speculate that they may be the longest running XP team. XP has been used from the very beginning, they use all twelve practices and they are mature in their use of XP. 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 determined what was required in collaboration with clients. Marketing were regarded as being, in effect, the customer.

The study took place in January 2002 and was carried out over a week. The start of the study coincided with the beginning of an iteration. In XP terms, an iteration is the two to four weeks of development which culminates in the release of working software to the paying customer. An iteration begins with the Planning Game, a group activity where requirements are explored and prioritised and the overall thrust of development for the iteration decided upon. The Planning Game we observed lasted until the end of Wednesday. The Thursday and Friday of the week were development days. At the end of the iteration the developers held a retrospective and a follow-up visit took place for that one day. During the week, the observer took part in day-to-day activities involved in XP development. The data collected consisted of contemporaneous field notes, audio recordings of discussions and meetings, photographs of the physical layout, and copies of various artefacts.

3.2 The setting

Before reporting the findings from our ethnographic study, we describe the physical setting of the study. We do this partly as good ethnography but also because, as we shall see, the spatial organisation of the office was significant insofar as the practitioners' work oriented to this organisation.

The office was open-plan, having an overall long rectangular shape with a walkway through the middle. This open-plan layout was organised into a number of areas, chiefly on the basis of differences in furniture and its layout but sometimes by means of half-height partitions.

The pair programming area – the developers' area – was situated towards the end of the walkway and had desks shaped specifically for programmers to sit two to a machine, as illustrated in Figure 1.

Figure 1 A pair programming station.

The developers' area was also enclosed by means of half-height partitions. The wall of the area had a notice board dominated by a large organised space devoted to 6" by 4" index cards. These were the active story cards – brief details of the tasks being actively worked upon. In addition, there were four 'to do' lists which occupied a prominent position. Two of these lists each had a picture at its top: one of Anton Chekhov (the Russian playwright) and one of Pavel Chekhov (the navigator of Star Trek's USS Enterprise). Figure 2 shows 'the Chekhov board'.

Figure 2 The Chekhov board where active story cards were displayed

Adjacent to the developers' area was desk space for the infrastructure support person and for the graphic designer. Close to the pair programming area was a stand-up bar of monitors, used for checking the status of live servers, running tests on different target platforms, as well as for other activities such as personal email and Web surfing. Behind the bar of monitors was a well-equipped kitchen where people would bring and leave their own food.

At the bottom of the walkway was a large open, communal area for meetings, etc. with tables, chairs and a large sofa. Beside the tables, there was a wall shelf full of 'tacky' gifts brought back by people from their holidays, there being a tradition that each member of staff would add to the collection by bringing back the tackiest gift that they could find on each occasion. Next to one of the tables was the machine used to release modified and tested code into the main system. This machine had on top of it a small box with a picture of a cow on it and the box would produce a 'moo' sound when picked up and tilted. Each time a developer released code, he/she would pick up the box and make it 'moo'. Finally, at the other end of the walkway to that of the developers' area was an area occupied by the marketing team. They sat at rectangular desks, facing each other.

4 Findings

Our analysis turned around the identification of five characterising themes (Fielding, 2001) which emerged from our 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 the twelve XP practices.

---

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

4.1 Shared purpose, understanding and responsibility

A key characteristic of all the observed activity was that it oriented around a shared purpose, understanding and responsibility within the team. What work needed to be done was negotiated (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 shared purpose, understanding and responsibility that we detail below applied both 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 and this was part of everyday life. 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.

We illustrate and detail what we intend by this theme with observations about the Planning Game, stand-up meetings, pair programming, documentation, and the use of metaphor.

The Planning Game

The Planning Game, which took place in the open area for meetings, was a substantial meeting that embodied a number of interwoven activities: designing, estimating and planning. Here is an edited extract from the field notes on the Planning Game.

> This activity was informal in that no-one called it to order. It began when enough people were there, and continued until enough people had left. The Planning Game involved all of the developers most of the time, with the marketing people being called in when necessary. Occasionally, developers would break away from the Game in order to service requests from clients or to find statistics relevant to the planning task.

> The business of the Planning Game was to examine stories, to estimate how long it will take to implement each story, and to decide which story cards can be satisfied in the next iteration. This was achieved by talking with each other and with marketing people when necessary. Some of the stories for this iteration were developer-generated; i.e., they were concerned with infrastructure issues or changes of architecture. Marketing people were asked to make trade-offs between customer story cards, but if the developers said that a technical card had to take priority then the marketing people did not argue. Marketing people were not asked for their opinion about prioritising these technical cards. Everyone calmly attempted to find ways around any conflicts.

> 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.

Figure 3 shows a story card: a brief description of something the system needs to do.
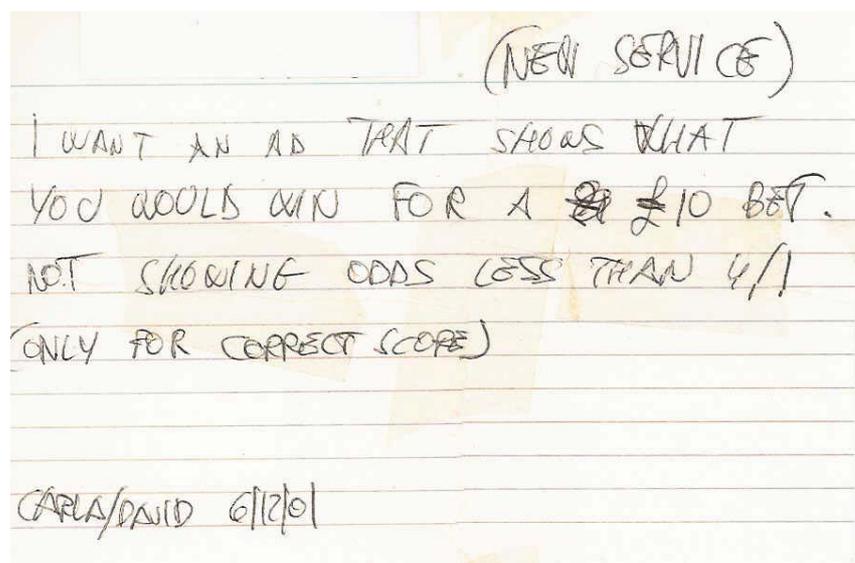
Figure 3 *A story card: "I want an ad that shows what you would win for a £10 bet, not showing odds less than 4/1 (only for correct scope)"*

A huge volume of communication and discussion took place, with many 'what if?' scenarios being explored. The Game we observed took up a large proportion of the first three days. Whilst many of the developers commented that this was an unusually long time, we saw no evidence that they regarded the meeting as being unnecessarily protracted: it simply took, on this occasion, that length of time. By the end of the meeting, 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. This was the only documentation produced from the meeting. 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 creates and sustains a shared purpose and understanding but can take time, 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.'*

Stand-up meetings

The first activity of the day for developers was the daily stand-up meeting. These meetings took place in the developers' area. No one sat down in these meetings: they were very short. One of the 'to do' lists (the 'Standup Chekov') stated that 15 minutes was the maximum time.

> ... 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, 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 individual '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. The act of choosing pairs centred on the individuals involved – each individual self-elected to work on a particular piece of the system, and each trusted the others to work on their chosen story card in the sense that there was no discussion or dispute on the competence or suitability of the pairings. This process was repeated at the start of each day: pairs would rotate so that it was unusual for a pair to stay together for longer than one day at a time.

Pair programming

Pair programming is the XP practice whereby all production code is created by two people working in concert at one machine with one keyboard and one mouse.

We observed that pairing rotates each day, and it was reported that pairing rotates frequently and everyone pairs with everyone else at least once during an iteration. This also means that everyone works on a large portion of the code base in each iteration. This rotation enhances the shared understanding and reinforces the shared purpose.

Everyone in the development team pair programmed. This included the graphic designer and the infrastructure person. Although the graphic designer didn't know Java (indeed hadn't worked on software before joining this company) she still would occasionally pair program because it helped her to understand the system.

We have already emphasised the volume of face-to-face communication that takes place in the Planning Game and in the stand-ups. In terms of our observations, we would argue that pair programming is as much about continuing that face-to-face communication as it is about writing code. Pair programming involved two people in a discussion where the results of that discussion were written up, sometimes by one of the pair and sometimes by the other. At an obvious level, pair programming is about producing code but it is also crucially about communicating, sharing and agreeing understanding. XP practitioners seem to recognise this fact. We found the following on an XP developers' wiki site about 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

Appended to this was the response:

> - Sorry, can't really see the difference, especially in the context of paired programming... or is there a pedant in the house?

The layout of the pair programming area allowed pairs to overhear discussions in another pair, which further reinforced shared understanding. The importance of peripheral awareness has been reported elsewhere (Heath &Luff, 1992), and there were examples during our study where one pair overheard another pair and joined in their 'meeting'.

Documentation

We have emphasised face-to-face communication: to use Cockburn's phrase, there was an oral tradition (Cockburn, 2000). 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. These include:

- Customer stories were written on cards.

- Estimates were written on story cards.

- Tests were written on the back of story cards.

- Rough designs and notes were drawn on the cards and used to communicate or explain ideas.

- Coloured stickers were placed on cards to denote progress, and so they were used as progress trackers.

- When pairs choose the story to work on in the morning, they take the card from the board. This means that no-one else can be working on the same story at the same time, so they are a means of controlling work.

- During the Planning Game the cards were moved about and clustered to show related work.

Agile methods in general, and XP in particular, seeks to eliminate 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 was 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, and anything written on index cards that had outlived its usefulness. No external record of the rationale for a decision was kept. This reliance on oral communication over formal records is strong evidence that shared purpose, understanding and responsibility is actively created and sustained within XP practice. The reliance on oral communication also emphasises the importance of developers' individual memories. 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.

Metaphor

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 of the developers we observed, based on minimal transient documentation, was particularly significant and they went to some trouble to ensure that appropriate communication was maintained. The vehicle for facilitating this shared understanding was one of the XP practices: that of metaphor, a simple story or model to share amongst the team about how the software fits together. In a sense, it is the vernacular software architecture of the system – and produces, amongst other things, a naming scheme for components and an aid to comprehension and understanding. 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".

Not only did we observe developers using their metaphor to discuss ideas among themselves, and to name code, we also observed the terminology spilling over into discussions with marketing people. It was regarded as a fundamental facilitator for maintaining the shared vision.

4.2 Coding and Quality of code matters

Code and coding mattered to the XP team we observed. Indeed, as we have already noted with the wiki site, they publicly took pride in programming. Parenthetically, we note that agile methods generally, and XP particularly, seek to resurrect programming from its perceived trivialization by more traditional approaches to software development, where it was regarded almost as a mechanistic skill that would surely be soon automated. In contrast, agilists valorize programming as a subtle and sophisticated achievement that speaks of great technical mastery (see Beck's comments on technical mastery and code aesthetics in Highsmith, (2002), for example).

Coding was regarded as a supremely important activity that should not be interrupted. This was reflected in the layout of the developers' 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.

The developers we observed were passionate about producing and maintaining good quality code. There are two aspects to this: providing customer service and keeping a quality code base.

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. This system was not working very well from either perspective and the issue was discussed during an observed company meetings.

> At the second meeting there were concerns about the interface between the marketing people and the developers. This concern spilled over into the relationship between the developers and the customers. 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. There was dissatisfaction on both sides for this arrangement and it was decided that other solutions were needed.

But it is not just the pressure of pragmatics that underlies a desire to produce quality code. Our observation here turns around the XP practice of refactoring. Refactoring is the restructuring the system (without changing its behaviour) to remove redundant or unnecessary code, simplify, add flexibility and to make the code (more) understandable. During one pair programming observation, significant frustration was shown by the developers who wanted to refactor the code, but were working on a story card that did not include an estimate for refactoring. 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 could be a tension between pragmatic considerations and the desire to refactor and have the simplest code.

Producing quality code was alluded to in different ways throughout the study, at higher and lower levels of detail. For example, there was no obvious coding standards manual, yet each developer was able to pick up the code they were working on extremely quickly. Coding standards were not overtly visible but must have been used since code written in an individual style would have been harder to understand.

We have mentioned that no unnecessary activity was devoted to documentation and that is true of the implicit coding standards that we infer, in the sense that no documentation of code was produced. However, viewed from our ethnographer's stance of considering all things 'strange' we offer a slightly different perspective. We have already suggested that a key feature of pair programming is that it is communication, where understanding is developed, agreed and shared. From a 'strange' stance, pair programming is also about writing and the quality of that writing and how it expresses the shared

understanding. That 'writing' is, of course, coding – another written tradition with a permanence that complements the transience of story cards.

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, although sometimes redundancy was identified before the code was written.

> Part way through the session, the developers pulled away from the workstation and drew some pictures on cards to illustrate what they were trying to do. As this discussion was happening, the pair to the side of my pair overheard some of the conversation and chipped in that they too were looking at the same area of the system, and at needing the same functionality. Neither pair realised this to start with, and it was only due to the layout of the physical space, and the proximity that allowed this collaboration to happen. It was agreed that the other pair would take responsibility for implementing this modification.

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 by the pairs observed:

> 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.

Code was tested against the complete set of system tests and released once a story card was complete. As estimates ranged between 0 and 3 days in increments of 0.25, a release could happen four times a day. This rate was not observed during the study, but two releases took place in the two programming days of the study (one on each day). 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. This recognises that the shared understanding comes with a price – pair programming is tiring

Having written the code, and got 'a green bar', i.e. the code had passed all system tests, the developers would move to the release machine and would announce the successful completion of a story card by making the box 'moo'. A significant event had taken place and its significance was connected with a belief in quality code.

4.3 Sustainability

As well as caring about the quality of code, the team cared about quality of life. This manifested itself most strongly in an atmosphere of calmness. Even during the Planning Game there was an absence of 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 be robustly emotive, 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 was not permitted to 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 planning game, 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.

... 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 some people attended the eXtreme Tuesday Club (XtC), a weekly London-based meeting open to all interested in XP, and on other days might meet for a chat in the bar. Not everyone took part in these activities, and there didn't appear to be any social pressure to do so.

The developers' had a love of fun which 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, Pavel 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 have 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.

One of the twelve XP practices is the injunction to 'Work no more than 40 hours a week as a rule. Never work overtime a second week in a row' (Beck, 2000, p 54). Our observations here are not simply evidence that the team followed the 40-hour week practice. It was a more skilful 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.

4.4 Rhythm

The team atmosphere that we observed was one of calm, competence and confidence, embodied by a pervasive rhythm. This rhythm operated at a range of levels and we 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 moo of code releases. People actively supported this rhythm.

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 have 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.

> 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

As each day passed, the story cards progressed through their life history of different coloured stickers, reinforcing the rhythm of progress under the gaze of the two Chekhovs. In a sense the two Chekhovs, with the story cards on the wall, orchestrate what has been done and what needs to be done, noticing the 'moo', keeping the score 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 was not happening. This orchestration of activity via a board of some sort has been noticed elsewhere in different settings such as that of patient care activity by nurses in ward settings e.g., Davis (2001), Chapter 5.

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. However, this did not mean that tasks were not challenging or difficult 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.

4.5 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.

We went to some lengths to give a detailed description of the physical setting, suggesting that it was significant. 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 – in rhythm.

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. 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.

5 Conclusions

Ethnographic studies, as Fielding (2001) has remarked, often have a hidden 'debunking' theme where the reality revealed by ethnographic study turns out to be at odds with the received, 'official' view. For example, we have reported elsewhere (Sharp et al., 2000a) that the reality of the emergence of the object-oriented paradigm is considerably more complex than that of a straightforward revolution where the significance of a new paradigm is immediately recognised and adopted. Outside of ethnography, it is also sometimes a commonplace that there is often a difference between what practitioners might say they do and what they actually do in practice (Cockburn, reported in Highsmith, (2002, pp 80-81) for example).

However, in contrast, we find here that our study leads to a set of conclusions that do not have a 'debunking' theme. 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. The rhythm was comfortable and relaxed, yet purposeful and productive.

We would summarise our findings by saying that our study would suggest that XP culture 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.

We were also struck by what we did not observe. For example, we found no evidence of activities that would normally be expected in software development projects such as use of rigorous modelling techniques, minutes of meetings, requirements documents, etc. Whilst XP practice was suffused with meetings of one sort or another, the only documentation used (story cards) was destroyed once it had outlived its immediate purpose. Diagrams and rough design sketches were produced as part of the Planning Game and might appear during programming, but once communicated, the diagrams were discarded.

In the five themes that emerged from the data we found evidence of a culture that values feedback, communication, simplicity and courage, underpinned by a respect for individuals and the team. We also observed a community of practice that structured and shared knowledge, and provided practical support. Wegner's (1998) three characteristics of mutual engagement, joint enterprise and a shared repertoire were clearly present in an accomplished fashion.

References

Beck, K. (2000) *eXtreme Programming Explained: embrace change,* Addison-Wesley, San Francisco.
Boehm, B. (2002) Get ready for agile methods, with care, *IEEE Computer,* **35**(1)**,** 64-69.
Cockburn, A. (2000) Balancing lightness with sufficiency, *Cutter IT Journal,* **13**(11)**,** 26-33.
Cockburn, A. (2001) *Agile Software Development,* Addison-Wesley, Reading, Massachusetts.
Davis , H. (2001) *The social management of computing artefacts in nursing work: an ethnographic account*, University of Sheffield
DeMarco, T. and Boehm, B. (2002) The Agile Methods Fray, *IEEE Computer,* **35**(6)**,** 90 to 92.
Fielding, N. (2001) Ethnography, In *Researching Social Life*(ed, Gilbert, N.) Sage, London, pp. 145 - 163.
Hammersley, M. and Atkinson, P. (1983) *Ethnography, Principles in Practice,* Tavistock, London.
Heath, C. and Luff, P. (1992) *Collaboration and control: crisis management and multimedia technology in London Underground line control rooms*, Proceedings of CSCW'92, 69 - 94

Highsmith, J. (2002) *Agile Software Development Ecosystems,* Addison-Wesley, San Francisco.

Low, J., Johnson, J., Hall, P. A. V., Hovenden, F. M., Rachel, J., Robinson, H. M. and Woolgar, S. (1996) Read this and change the way you feel about software engineering, *Information and Software Technology,* (38)**,** 77-87.

Quintana, C., Krajcik, J. and Soloway, E. (2000) Exploring a Structured Definition for Learner-Centered Design, In *Fourth International Conference of the Learning Sciences*(eds, Fishman, B. and O'Connor-Divelbiss, S.) Erlbaum, Mahwah, NJ, pp. 256-263.

Robinson, H. M. (1996) (Re)presenting the p-word: paradigmatic discourse on programming languages, In *Programming Language Choice*(ed, Woodman, M.) International Thomson Computer Press, London, pp. 333-344.

Schön, D. A. (1983) *The Reflective Practitioner,* Temple Smith, London.

Sharp, H. and Robinson, H. M. (2002) *Object technology: community and culture*, OOPSLA '02 Companion, Seattle, 92-93

Sharp, H., Robinson, H. M. and Woodman, M. (2000a) Software engineering: community and culture, *IEEE Software,* (January/February)**,** 40 - 47.

Sharp, H., Robinson, H. M. and Woodman, M. (2000b) *Using ethnography and discourse analysis to study software engineering practices*, The 22nd International Conference on Software Engineering (ICSE 2000) Workshop Proceedings, Limerick, Ireland, 81 - 87

Sharp, H., Woodman, M., Hovenden, F. and Robinson, H. M. (1999) *The role of 'culture' in successful software process improvement*, Proc. IEEE Euromicro '99, Milan,

Wenger, E. (1998) *Communities of Practice: Learning, Meaning and Identity,* Cambridge University Press, New York.