Chapter 5
Collaboration, Communication and Co-ordination in Agile Software Development Practice

Hugh Robinson and Helen Sharp

Abstract This chapter analyses the results of a series of observational studies of agile software development teams, identifying commonalities in collaboration, co-ordination and communication activities. Pairing and customer collaboration are focussed on to illustrate the nature of collaboration and communication, as are two simple physical artefacts that emerged through analysis as being an information-rich focal point for the co-ordination of collaboration and communication activities. The analysis shows that pairing has common characteristics across all teams, while customer collaboration differs between the teams depending on the application and organisational context of development.

5.1 Introduction

Agile software development is a group of software engineering methodologies, e.g., eXtreme programming (XP) [4] Scrum [26] Crystal [11] that became popular in the early 2000s. Agile advocates claim to increase overall software developer productivity, deliver working software on time, and minimise the risk of failure in software projects. Whilst its effectiveness and applicability remain uncertain, (e.g., [1, 19]) it is attracting increasing interest from the software engineering community, (e.g., [6, 24]). A summary of what is involved in agile software development is given in this description by Cockburn [10: 29].

> 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 <u>always</u> to run at 100% of all code that is checked in, and for code to be developed

H. Robinson (✉)
Centre for Research in Computing, The Open University, Milton Keynes, MK7 6AA, UK
e-mail: h.m.robinson@open.ac.uk

> in nano-increments, checked in and integrated several times a day. The result is delivered to real users every 2–4 weeks.[1]
>
> 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.

Agile software development produces working software by *technical* practice that also creates, and depends upon, intimate *social* activity which emphasises close collaboration, co-ordination and communication within the development team. This chapter explores the detailed nature of this social activity and its relationship to and embodiment in the technical practice. The analysis is based on the results of empirical studies we have carried out with six co-located mature XP software development teams, covering a range of organisational settings, application domains and development environments. Our approach to both data collection and analysis is ethnographically-informed [25] which results in a validated account of the detailed collaboration, co-ordination and communication mechanisms employed and their relationships to each other and to technical practice. The approach is not hypothesis-driven, but data-driven.

The analysis is in two parts. First, in section 5.3, we discuss and demonstrate how the reality of agile technical practice involves *collaborative* and *communicative* social activity. This is illustrated with consideration of two aspects of technical activity which have key social characteristics: pairing and customer collaboration. Second, in section 5.4, we analyse the critical work of *co-ordination* of collaborative and communicative activity via the mechanisms associated with key physical artefacts: story cards and the Wall. As background to this analysis, we introduce XP as a social activity (section 5.1.1), and describe the fieldwork on which the analysis is based (section 5.2). Following on from the analysis, we discuss the significance of our findings in Section 5.5, and end with our conclusions in Section 5.6.

5.1.1 XP as a Social Activity

XP is commonly perceived in terms of technical practice. XP articulates its technical practice as a set of mutually supportive components – *practices* – that include, for example, small releases, simple design, testing, refactoring, pair programming and continuous integration. In [3] 12 practices are listed, which are refined and extended into 13 primary practices and 11 corollary practices in [4]. Beck states that the practices interact to mutually support one other: “Any one practice doesn’t stand well on its own (with the possible exception of testing). They require the other practices to keep them in balance.” [3: 69]. Consequently, any analysis and evaluation of one of

[1] Time-boxed units of development lasting 1–4 weeks are called “iteration’s” in XP; time-boxed units of development around four weeks are called “sprints” in Scrum.

the XP practices has to take into account the manner in which it works in concert with other practices.

As well as being technical practice, XP is also fundamentally a social activity, with explicit values, such as communication and respect, and explicit principles, such as humanity and reflection [4]. Interviewing Beck, Highsmith observes that his “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.” [16: 53]. Beck states that: “Just as values bring purpose to practices, practices bring accountability to values.” [4: 14]. Such claims by XP advocates as to the importance of social activity are sustained by several researchers, (e.g., [9, 20, 31]), and practitioners, (e.g., [11, 21]).

The reliance of software engineering practice on purposeful social activity has been recognised elsewhere, (e.g., [14, 30]), and so XP is not unique in this respect. However the detailed nature of this social activity and its relationship to and embodiment in technical practice has not been investigated and analysed. In this chapter we focus specifically on exploring and analysing XP’s collaborative, communicative and co-ordinating dimensions. Our account of social activity will meet two important requirements. First, it will be an account that attends to the technical as well as the social. Second, it will be rooted in the reality of what practitioners do – XP in the wild,[2] so to speak – and that demands empirical fieldwork.

5.2 Fieldwork

Our findings represent a synthesis of results from a series of six empirical studies of software practice. Our empirical studies were all fieldwork studies of teams based in industry, engaged in software development, and using XP. Each team was mature at the time of the fieldwork; that is, they had successfully transitioned to XP[3] and had been using all of Beck’s original 12 practices [3] for at least a year. Each team consisted of software developers and other team members carrying out various roles providing business, project management and specialist technical skills. The number of developers in the team varied from 23 to 5 and the overall team size varied from 7 to 26 (see Table 5.1).

For example, Team C had two business development staff and a project manager; another – Team E – had a project manager, two business analysts, a database administrator and a technical database user. The business settings of the six teams varied

[2] cf. Edwin Hutchins’ Cognition in the Wild, MIT Press, Cambridge, MA, 1995.

[3] Transitioning to XP is a process that can take place over a weekend or can require several months, depending on a range of factors such as team size, organizational culture and team member attitude, for example.

Table 5.1 Team composition and business setting

Team	Overall team size	Number of developers	Business setting
A	12	8	Web-based intelligent adverts
B	23	16	Document use in multi-author work environments
C	26	23	Travel information web pages & alerts
D	15	12	Large international bank
E	10	5	Large international bank
F	7	5	Large telecommunications company

(see Table 5.1). Each team was physically co-located, essentially in a large, open room.

Each team was studied for a period of a week (sometimes with additional spells of observation, so that, in effect, iterations of more than a week were accommodated), with further follow-up meetings to discuss findings. An ethnographically-informed approach [25] was taken with the researcher immersing themselves in the day-to-day business of XP development, documenting practice by a variety of means that included contemporaneous field notes, photographs/sketches of the physical layout, copies of various documents and artefacts, and records of meetings, discussions and informal interviews with practitioners. Data was analysed ethnographically and thematically, emphasising validation through the seeking of confirming and disconfirming instances. The thematic, ethnographic analysis of the data was complemented with an analysis from a cognitive dimensions [15] theoretical perspective for some of the data [28]. An analysis informed by a distributed cognition theoretical perspective, based on DiCOT (Distributed Cognition for Teamwork) [5] was also employed for the data collected with three of the teams [27].

5.3 The Social in the Technical: Collaboration and Communication

The Agile Manifesto [2] emphasises collaboration and interactions, and the reality of XP software development offers evidence that this emphasis is borne out in practice. Observing practice makes it clear to the researcher that the work of an XP team visibly and continually involves collaboration and communication – and that collaboration and communication are part of the technical business of creating working software. In this section we explore and analyse this intimate relationship between the social and technical via two key XP practices which illustrate this relationship: *pairing* and *customer collaboration.* We find that pairing has considerable commonalities across the six teams, while the detail of customer collaboration varies, dependent on the team's specific situation.

5.3.1 Pairing

By *pairing* we refer to the social activity of two team members (usually developers[4]) sitting together and working. Pairing work encompasses several of the mutually supportive components of technical practice: pair programming, test-first coding, refactoring, simple design and continuous integration. That is, pairing does not just involve two programmers together writing production code: it also involves test-driven development, the refining of code structure, the removal of complexity as soon as it is discovered, and the integration of new, or changed, code into the existing code base via the 100% passing of automated tests.

The collaborative activity of pairing is dominated by communication: talk between the two programmers, as they discuss, investigate, reason, understand and develop the task at hand. Understanding is shared and affirmed (*"So, are you saying there's an AddAllocation? Yes."*)[5] and action is negotiated and carried out (*"Why don't we do the simplest thing and put in a test... that's easy to test.", "It's the simplest thing and it's compatible with refactoring."*), lack of progress is acknowledged (*"So, detecting everything else wasn't a very good idea"*) and completion signalled (*"I'll commit that!").* Silence is also an accepted feature of the talk, as code is being run through a series of tests, when an unexpected "red bar" (failing test) is encountered or simply when thought is required.

In our fieldwork, the talks, and the talkers' roles, were fluid depending on the nature of the task, the developers involved and the progress being made. For example, an experienced developer would pair with a less-experienced colleague so that the experienced developer could gain familiarity with portions of the code base that the less-experienced colleague had been working on. Alternatively, experienced developers may pair where the portion of the code base being modified is particularly complex or the required change is tricky. In particular, contrary to claims by XP advocates, (e.g., Beck [3: 58]), there was no evidence of any clear split in roles, with one developer controlling the keyboard and mouse to produce code while the other was thinking more strategically. Rather, both developers would adopt these roles interchangeably as the talk progressed and the possession and use of the keyboard and mouse oriented to the talk (and not the other way around); this is confirmed by others and a more detailed study of this phenomenon is reported in [7]. The talk sometimes involved more than the two developers who were pairing, when someone in another pair would overhear the talk and offer their clarification or understanding (if it were part of the code base in which they had expertise). Indeed, the ability of pairs to peripherally overhear each other was taken for granted as desirable and was exploited to make progress for the team.

As well as involving developers, the talk also actively involved the code and its various manifestations in terms of the windows and panes of the many development

[4] We have observed pairings of a developer with a graphic designer, and a developer with a business analyst.

[5] Such italicized, bracketed material, in quotes, is an illustrative extract from our field notes.

tools employed by the developers. The conversational turns of this third partner were orchestrated by the developers as they summoned and dismissed panes, launched tools, etc. The response of the third partner could – and would – shape the talk of the developers, demanding close attention to what the code was expecting of them. The code was a central focus in the talk.

Pairing is intimate and intense at both the social and technical level and this was reflected in the developers' organisation and management of their working environment in terms of time, relationships between individuals and space. The organisation of the working day ensured that pairing did not take up much more than 5–6 hours in the day – more than this was regarded as stressful and not sustainable. Similarly, the period of pairing itself was actively managed, with recognition of the need for breaks. In all our teams, pairs would swap around regularly – anything from half a day to several days may be spent in one pair, depending on the functionality being worked on. However, framed by this organisation and management, pairing was visibly a period where developers both expected and displayed great concentration and focus.

Whilst pairing sessions themselves are intense and intimate, pairing as an ongoing activity – on a daily, week-in, week-out basis – has its own intensity that requires a level of maturity and social management from developers to accommodate inevitable clashes of programming style, attitude and personality. The development teams studied recognised this in a variety of ways. The leader of one team monitored and adjusted pairing to ensure active and effective engagement. Another team likened the individual relationships of pairing to those of marriage and sought to display all the skills of compromise, sensitivity and negotiation that this required. And another team made use of a qualified social worker to help the team understand the overall social health of its relationships. On a daily basis, many of our teams kept a record of pairings, e.g. a pairing ladder that highlighted common and uncommon pairings to make sure that rotation was evenly spread among team members.

The organisation of the space of the working environment oriented to the nature of pairing. This orientation ranged from the reconfiguration of desks for pairing to the separation of space into an area for pairing, as well as areas for activities that did not involve pairing, such as meetings, email and phone use.

Collaboration and communication occurs between pairs as well as within pairs. Apart from the exploitation of peripheral awareness mentioned above, collaboration and communication also occurs between pairs in the "stand up." The stand up is a daily meeting, taking place early in the day, before pairing begins. All developers attend and the meeting is short (no more than 15 min) – and people stand for the duration. The meeting uncovers the collaboration and communication that must take place across the developers in the coming day and initiates its co-ordination. This is achieved by each developer quickly reporting in a three-part fashion: what they've done since the last stand up that others need to know about, what they will be doing next that others need to know about, and what if any obstacles are holding them back (and that others can help with). The stand up emphasises reporting, and prolonged discussion does not take place. As a result of what is reported, various discussions will take place during the day, although rarely in the setting of another meeting.

5.3.2 Customer Collaboration

By "customer collaboration" we refer to the activity associated with the on-site customer component of XP technical practice, where the customer generates requirements, answers developers' queries and provides understanding, sets priorities, and provides feedback on iterations. Beck describes the on-site customer thus: "A real customer must sit with the team, available to answer questions, resolve disputes, and set small-scale priorities. By 'real customer' I mean someone who will really use the system when it is in production." [3: 60]. That is, in the ideal XP world of Beck's advocacy, the people filling the on-site customer role would be co-located with the developers; would "speak with one voice"; would be potential users of the system; and would be collaborative, representative, authorised, committed and knowledgeable. It is an accepted fact of XP practice that this ideal is rarely realised for a variety of reasons: client organisations may be unwilling or unable to spare people to become part of the development team; different customers may have conflicting requirements; potential users of the system may not have the authority to identify and prioritise system features, whereas decision makers may not understand the needs of users; and so on. XP practitioners have recognised this fact and devised approaches and methods to deal with the gaps between the ideal and the reality, (e.g., [22, 23, 29]). These approaches and methods are contingent upon, and are shaped by, the specific context and circumstances of the development team and who is taking the role of the "customer."

To demonstrate the nature of customer collaboration we briefly describe the collaborative and communicative activity of each of our six teams, focussing on interactions between the customer and developers.

The first setting involved a team where the on-site customer role was carried out by marketing personnel who dealt directly with individual paying clients on a regular basis. This direct involvement with the client brought great clarity and authority to the development process. However, the role of marketing personnel demanded that they respond quickly (minutes rather than hours) to requests from clients. Usually, such requests necessitated consultation (and hence considerable interaction) with developers. Much as the developers valued customer collaboration, the frequency of such interruptions proved too distracting given the demands for focus and concentration from the intensity of pairing. The solution explored was that of an "exposed pair": each day a pair of developers was identified who could be interrupted if a client had an urgent request. Such a solution could only work because of the shared understanding and responsibility created by other XP practices including pairing.

In the second of our settings, the on-site customer role was carried out by project managers who worked with marketing but were firmly part of the development team. As such, they understood both the market requirements and positioning of the company's various products and the needs of the software development that would create those products. Project managers organised a considerable amount of the detail of software development, as well as orchestrating and managing requests from marketing. They therefore managed a complex set of interactions between various groups and individuals. It was noticeable that pairing was more "interruptible" here: *ad hoc* discussions involving pairs and a project manager would naturally occur and often would involve individuals from another pair, or testers, or the team coach. Once the particular issue was resolved, pairing would resume and there was no sense that what had occurred was an "interruption." A variation of this occurred with our third setting where the team were the basis of a small software company with a flat organisational structure. Here, the on-site customer role was carried out by the handful of individuals who were management with collaboration and communication activities that were similar to those described above.

Our fourth setting concerns a team working in a large international bank, developing the software that would support the institution's management of operational risk. The management of operational risk was a new regulatory body requirement and hence the details of the institution's methodology were taking time to emerge. The on-site customer role was carried out by two individuals with expertise in the institution's methodology but it was a new area and there were sponsors and stakeholders, senior to the two individuals, who needed to finalise and agree the methodology. As a consequence, requirements were often subject to change. In addition, the on-site customer was not the intended user of the various applications, and the institution had a strong tradition of conventional, plan-driven software development with all its expectations of how sponsors, stakeholders and users interact with software developers. The on-site customer was also not co-located with the developers although relatively close and in the same building. Importantly, the on-site customer had significant responsibility for the overall success of the applications under development. All of these factors made collaboration and communication particularly demanding for both the on-site customer and the developers. Both worked actively to manage the relationship and overcome problems, and reported positively on this aspect at a retrospective. Developers proactively involved the customer at a range of opportunities, including planning meetings, seeking them out after a stand-up, and ensuring their involvement in the team's coffee breaks. Considerable effort was expended in developing a shared understanding of the risk methodology via *adhoc* meetings.

The other team in this same bank (our fifth team) were migrating a range of existing independent databases, each with their own, different schema, to one integrated database, with its own, new schema. For them, the customer role was taken by a technical database user who had many years' experience with the existing databases. He was co-located with the team, but not always available. Communication and collaboration here were complicated by the inclusion of business analysts who were creating the new database schema, and hence needed to communicate with both the customer and the developers. This required three-way communication and co-ordination and a double stand-up meeting each morning – one only for developers and one with developers, customer and business analysts. All of this was overseen by a project manager who was responsible for liaising with the offshore database administrators and the team's immediate line management.

Our final setting concerns that of a team working in a large telecommunications company. The customer (a representative of a large department who were the main stakeholder in the work) was not on-site and was located several hundreds of miles from the developers. Interaction between the customer and the developers routinely took place once a week via a telephone conference, with other calls during the week as and when queries arose. A wiki was also used to share information. Despite the customer and developers rarely meeting each other, developers reported that this arrangement worked effectively because they had worked with the system under development for several years and believed that they had a good understanding of what was likely to be acceptable to the customer and what was not.

In summary, collaboration and communication with the customer is rich and varied but also is highly situated. As such, and unlike pairing, it is difficult to identify recurring collaboration activities and communication patterns. For example, it is highly unlikely that the approach taken in our final setting would work so effectively in the situation of our fourth setting.

5.4 The Social in the Technical: Co-ordination

We now consider how these collaborative and communicative activities in XP practice are co-ordinated. Specifically, we analyse the co-ordinating role of two key physical artefacts identified through our analysis: the Wall and story cards. Figure 5.1 is an example of the Wall and two story cards from our fieldwork. The "Wall" is our term but it is a term, and a role, that practitioners readily recognised and agreed with in feedback sessions with them on our fieldwork. The Wall is an example of the Informative workspace primary practice of Beck & Andres [4]. However, Beck & Andres describe the primary practice simply in terms of "An interested observer should be able to walk into the team space and get a general idea of how the project is going in 15 seconds". They neither explicate nor advocate the key, detailed co-ordinating role of the Wall.

5.4.1 Story Cards

Stories are the key unit of communication between the customer and developers and are small units of functionality for which working code can be developed after a day or maybe two days' effort. Such fine granularity is facilitated by the identification and refinement of "epic stories" and larger chunks of functionality [12, 13]. Jeffries [18] suggests that there are three parts (the three "C"s') to a story: the Card, the Conversation and the Confirmation.

> The Card: Stories are usually written on... index cards. Cards are small, physically independent entities. Their size constrains the amount of information that can be written on it, while

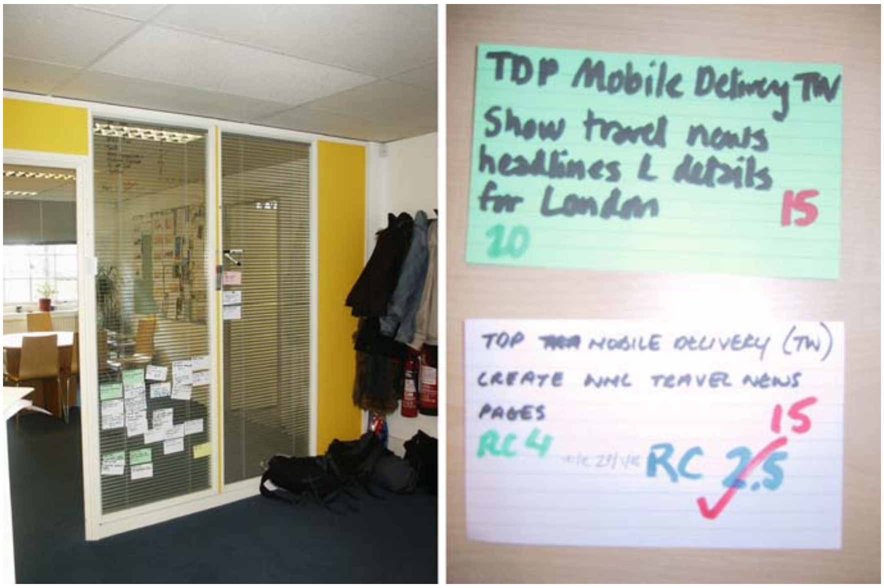
TDP Mobile Delivery TW
Show travel news headlines & details for London
15
20
TOP ~~TRA~~ MOBILE DELIVERY (TW)
CREATE WML TRAVEL NEWS PAGES
15
RC 4
RC ~~2~~.5

Fig. 5.1 An example of the wall and story cards from one fieldwork site

its independent nature means that it can be annotated and manipulated during meetings or discussions.

The Conversation: Because the card can only hold a limited amount of information, the development team has to talk to others in order to explore the detail of the story and to refine their understanding of it.

The Confirmation: Testable and measurable user acceptance tests are agreed between the customer and the development team, so that everyone concerned understands when a story has been implemented successfully.

Each of these three parts has strong social characteristics that are significant in co-ordination: the card's independent, almost ubiquitous, nature; its role as a summons for shared understanding; and its insistence on an operational definition of completion and closure.

Stories are usually thought of as being customer-initiated and as being about customer-visible functionality. Our fieldwork revealed that stories can also be developer-initiated and be about developer-required technical change such as refactoring. Furthermore, a story is often broken down into smaller units, known as tasks. For example, in Fig. 5.1, the top card is a story card ("Show travel news headlines and details for London") and the bottom card is a task card ("Create WML travel news pages") which is one of the tasks of the story. Figure 5.1 does not show that, in fact, the top story card is green and the bottom task card is white, so that the use of different coloured cards indicates the level of granularity. The use of different coloured cards here is deliberate and a common practice amongst teams. All of the teams we studied made use of stories and all, with one exception, made use of story (index) cards. The exception was a team which had moved from the use of index cards to an electronic, Word document. This Word document permitted considerably more detail about what was required than would have been possible on an index card and also included full details of the acceptance test.

At the start of an iteration, an iteration planning meeting is held to determine which stories will be developed in the coming iteration. The cards that are being considered for the iteration[6] are often physically dealt on to the meeting table. The planning meeting is collaborative with all team members and the on-site customer being involved. Customers are asked to prioritise stories for the coming iteration, and developers ensure that they have estimated how long each story will take and that the cards are annotated with this information (such an estimate appears in the bottom left corner of the (top) story card in Fig. 5.1). Working together, the team determines how many and which stories will be included in the coming iteration. Frequently, the physical space of the meeting table and the independent nature of the cards are used to group and arrange cards to aid this process.

5.4.2 The Wall

Once the stories for the coming iteration have been determined they are taken and arranged on the Wall. The Wall may be a convenient physical wall, as in the case of Fig. 5.1, or it may be whatever is to hand. Examples from our fieldwork include the vertical front surface of a collection of filing cabinets (see Fig. 5.2), a flip chart, and a large (foldable and highly portable) piece of cardboard. That is, it matters to teams that they have a Wall and they will create one in the most difficult of settings. Even the team who held stories electronically had a cut-down version of the Wall.

The exact way in which each team arranged, and manipulated, story cards on the Wall varied and we give here a simplified, but nevertheless, essential description where the team worked in iterations of 3 weeks. The Wall is divided into three main sections, one for each week of the iteration. The section for a week is sub-divided into a "to do" area and a "done" area (see Fig. 5.3). At the start of the iteration, the team considers how the cards need to be distributed across each of the 3 weeks and carefully construct the Wall accordingly. Initially, only the "to do" area within the Wall section for each week has any cards and the "done" area is empty. Within the "to do" areas, cards are arranged so that task cards are with their associated story card.

Following the first stand up of the iteration, some cards are removed from the "to do" area of the first week – each card being taken by a pair of developers. The Wall is annotated to indicate that a card has been moved (e.g., in Fig. 5.3 by the dotted rectangle). In the case of the Wall of Fig. 5.1, a ghost of the moved card would be drawn on the glass so that the card's position on the Wall was preserved. The pair

[6] Software is released after a series of iterations, typically every few months. There is a layer of release planning, which helps scope out the functionality of an iteration that we have not touched on here.

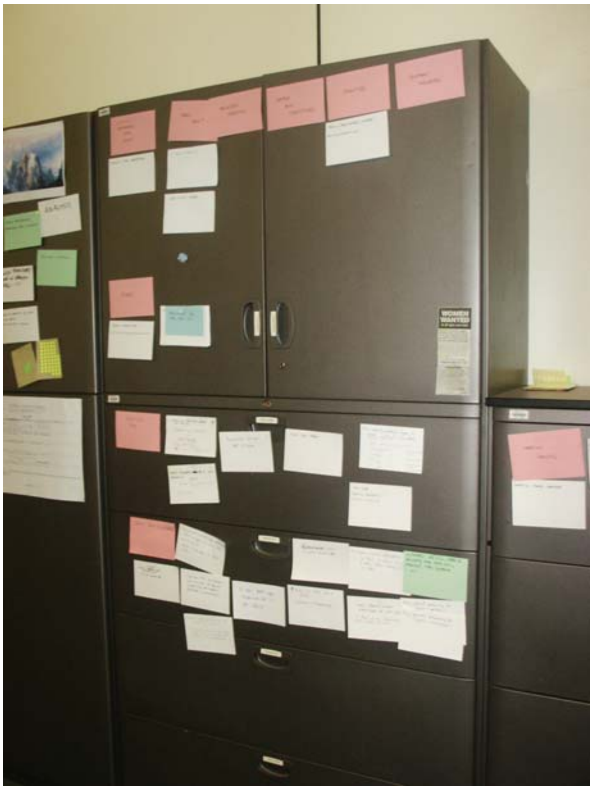

Fig. 5.2 Filing cabinets used as the wall

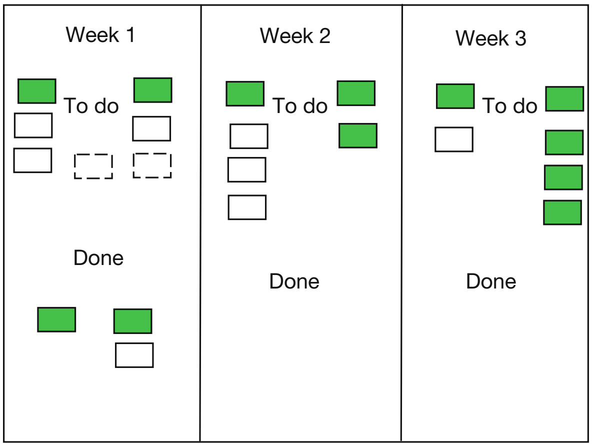

Fig. 5.3 A schematic of the wall shown in Fig. 5.1

takes the card to a workstation, stick the card to the monitor, and engage in pairing. Once they have produced tested, integrated working software, they annotate the card with their initials, the actual time taken, and a large tick to indicate that it has been completed and return the card to the Wall,[7] placing it in the "done" area for the week, erasing the annotation in the "to do" area that indicated the card was being worked on by a pair.

Daily stand ups are conducted around the Wall, with individuals often pointing at the Wall or taking cards from the Wall. By taking a card from the Wall, a developer signals that they want to speak about the card and that they are exercising a form of ownership[8] over the work it represents. During the day, developers often look at the Wall when considering progress, or the work left to be done.

At the start of the next week, the Wall is carefully studied by the team and rearranged appropriately if the team has not completed all the stories initially allocated to the week that has just finished.

This essential account makes it clear that the Wall and its associated cards are not just visible signs of progress for visitors, managers and team members, as the advocacy literature of Beck [3] or Cockburn [11] would suggest. Rather, they are an information-rich focal point for the co-ordination of collaboration and communication. The Wall and its associated story cards work in a complementary manner. The card is annotated in strict ways as it progresses through the development cycle, but the card itself represents too small a chunk of development to stand alone – it is important to see the wider overall picture of progress and activity. The Wall provides this overview, and is designed spatially to carry extra information which complements the detail shown on each card. Much of the mechanics we have described – card annotation, displaying stories on a wall, taking cards to a workstation when implementation has started, etc – are focussed on co-ordination of the team members' efforts. However the way in which this co-ordination is achieved underpins the collaborative and communicative nature of the team's work and makes it possible for such close collaboration and communication to be successful.

5.5 Discussion

In order to make technical progress, code must be implemented, and in order to make that code useful, requirements must be understood through interaction with customers. In XP, pairing supports the creation of code, and customer collaboration supports understanding requirements. These two activities are clearly technical practices, but our accounts also show the key facilitating role played by social activity.

[7] All actions that involve a card are carried out with a care that transcends its deceptive simplicity and informality. Indeed, one team studied had an internal wiki entry entitled "The care and feeding of story cards."

[8] Collective ownership is part of the technical practice of XP: "Anybody who sees an opportunity to add value to any portion of the code is required to do so at any time." [3: 59].

A striking difference between pairing and customer collaboration is that pairing involves repeatable patterns of collaborative and communicative activity that transcends teams and their contexts, while interaction with the customer is very rich and highly situated. In order for regular communication to take place between customers and developers, the activity of pairing needs to be interrupted, and different teams handle such interruptions differently. Teams also vary in terms of whether and how often the customer attends the daily stand-up. As others have noted, the role of customer is rarely (in our six teams – never) taken by the ideal individual and the individual circumstances of that person affects the nature of collaboration and communication. For example, how much authority the customer has in making decisions; how much knowledge of the domain the customer has; where the customer is located relative to the developers; and so on. All of these impact the nature of the collaborative and communication activities required to support technical development.

Much of the co-ordination activity supported by the Wall and the cards captures progress information rather than functional information. The Wall, supported by annotations on the story cards, is good at showing an overview of the team's progress, but it is not good at showing an overview of the structure of the code, or the functionality being offered. Instead, the functional attributes and structure of the software is communicated, evolved and kept safe through social activities such as pairing and customer collaboration as described above.

One consequence of this is that project management tools, commonly in use within the software industry, need to link into the Wall and its mechanisms for capturing progress. A tempting solution may be to digitise story cards and the Wall to enable this linkage, but software tools based around the Wall and the cards must support the facilitation, management and visibility of working activity offered by their physical counterparts rather than just produce electronic versions of these artefacts, however sophisticated (see [8] for a compelling example of such an approach to the computerisation of a workflow system in the print industry). Developments such as that of Iterex [17] are promising. The Iterex system supports the creation of story cards in accordance with Jeffries' three "C"s', the breaking down of a story into tasks, the colour coding of stories/tasks and their arrangement and their printing for use "as technology in their own right." Importantly, the system links support for story cards into the other activities of tracking iteration and release progress, visualising project velocity, scope and burn down/up and planning future releases based on past performance.

Another consequence of the Wall's focus on progress and not functionality is that the social activity underpinning the discussion, evolution and agreement of functional development and progress is crucial to effective code development.

5.6 Conclusion

The social activity we have described and analysed – the collaboration and communication of pairing and customer collaboration, and the co-ordination of the Wall and its associated story cards – brings purpose and meaning to the technical practice of XP: to pair programming, test-driven development, refactoring, simple design, continuous integration, and the on-site customer. Similarly, the technical practice makes the activities of collaboration, communication and co-ordination accountable: it is not just any ("warm and fuzzy") collaborative, communicative and co-ordinating activity that is acceptable but the detailed work, intimately connected to the technical that our analysis has revealed. The creation of working software is a socio-technical enterprise.

References

1. Abrahamsson P, Warsta J, Siponen MT, Ronkainen J (2003) New directions on agile methods: A comparative analysis. Proceeding of ICSE'03, ACM, New York, pp. 244–254.
2. Agile Manifesto http://agilemanifesto.org/, accessed 7th October 2008.
3. Beck K (2000) eXtreme Programming Explained: Embrace Change. San Francisco: Addison-Wesley.
4. Beck K, Andres C (2004) eXtreme Programming Explained: Embrace Change. San Francisco: Addison-Wesley.
5. Blandford A, Furniss D (2005) DiCoT: A methodology for applying distributed cognition to the design of team working systems. Proceedings of DSVIS'05, Springer, Berlin, Heidelberg, New York.
6. Boehm B, Turner R (2004) Balancing Agility and Discipline. Boston, MA: Addison-Wesley.
7. Bryant S, Romero P, du Boulay B (2008) Pair programming and the mysterious role of the navigator. IJHCS 66(7): 519–529.
8. Button G (2004) Changing ways of working, seminar given at the IBM Alamaden Institute, presentation available at http://www.almaden.ibm.com/institute/2004/bio/2004/index.shtml?button.
9. Chong J (2005) Social behaviours on XP and non-XP teams: A comparative study. Proceedings of Agile 2005, IEEE, Los Alamitos, pp. 39–48.
10. Cockburn A (2000) Balancing lightness with sufficiency. Cutter IT Journal 13(11): 26–33.
11. Cockburn A (2004) Crystal Clear: A Human-Powered Methodology for Small Teams. San Francisco: Addison Wesley.
12. Cohn M (2004) User Stories Applied. San Francisco: Addison-Wesley.
13. Davies R, Sharp H (2006) Early and often: Elaborating agile requirements. Cutter IT Journal 19(7): 6–11.
14. Good J, Romero P (2008) Collaborative and social aspects of software development. IJHCS 66(7): 481–483.
15. Green TRG, Petre M (1996) Usability analysis of visual programming environments: A 'cognitive dimensions' framework. Journal of Visual Languages and Computing 7: 131–174.
16. Highsmith J (2002) Agile Software Development Ecosystems. San Francisco: Addison-Wesley.
17. Iterex (2008) http://www.planningcards.com/site/, accessed 13th October 2008.
18. Jeffries R (2001) Essential XP: Card, Conversation, Confirmation, accessed 04.11.07 http://www.xprogramming.com/xpmag/EXPCardConversationConfirmation.htm.
19. Kruchten P, Adolph S (2008) Scrutinizing agile practices or shoot-out at the agile corral. Workshop at ICSE 2008, Leipzig, Germany.
20. MacKenzie A, Monk S (2004) From cards to code: How extreme programming re-embodies programming as a collective practice. CSCW 13: 91–117.
21. Mackinnon T (2003) XP – call in the social workers. In: Marchesi M, Succi G, (Eds.) Proceedings of XP2003, Lecture Notes in Computer Science, Vol. 2675. Berlin, Heidelberg, New York: Springer, pp. 288–297.

22. Martin A, Biddle R, Noble J (2004) The XP customer role in practice: Three case studies. Proceedings of the Second Agile Development Conference, Salt Lake City, Utah, June 22–26. IEEE, Los Alamitos.
23. Martin A, Noble J, Biddle R (2003) Being Jane Malkovitch: A look into the world of an XP customer. In: Marchesi M, Succi G (Eds.) Proceedings of XP2003, Lecture Notes in Computer Science, Vol. 2675. Berlin, Heidelberg, New York: Springer, pp. 234–243.
24. Paulk MC (2001) Extreme programming from a CMM perspective. IEEE Software, November/December 2001, pp. 19–26.
25. Robinson HM, Segal J, Sharp H (2007) Ethnographically-informed empirical studies of software practice. Information and Software Technology 49(6): 540–551.
26. Schwaber K, Beedle M (2002) Agile Software Development with SCRUM. Englewood Cliffs, NJ: Prentice Hall.
27. Sharp H, Robinson HM (2008) Collaboration and co-ordination in mature eXtreme programming teams. IJHCS 66: 506–518.
28. Sharp H, Robinson HM, Petre M (2009) The role of physical artefacts in agile software development: Two complementary perspectives. Interacting with Computers 21: 108–116.
29. Sharp H, Robinson HM, Segal J (2004) eXtreme programming and user-centered design: Friend or foe? Proceeding of HCI2004, Leeds.
30. Weinberg G (1998) The Psychology of Computer Programming. New York: Dorset House.
31. Whitworth E, Biddle R (2007) The social nature of Agile teams. Proceedings of Agile 2007, Washington, DC, August, IEEE, Los Alamitos, pp. 13–17.