A Distributed Cognition Account of Mature XP Teams

Helen Sharp and Hugh Robinson

Centre for Research in Computing
The Open University
Walton Hall
Milton Keynes MK7 6AA UK
{h.m.robinson, h.c.sharp}@open.ac.uk

Abstract. Distributed cognition is a framework for analysing collaborative work. It focuses on interactions between people, between people and their environment and between people and artefacts that are created and manipulated in the course of doing work, and it emphasises information flow and information transformation. Analyses conducted using the distributed cognition framework highlight breakdowns and potential problem areas in the collaborative work being studied; distributed cognition has been used to study a wide variety of collaborative work situations. XP teams are highly collaborative, relying heavily on interactions between team members and their environment. In this paper we present accounts of four mature XP teams based on the distributed cognition framework.

1 Introduction

Distributed cognition is an approach for conceptualising human work activities that considers the people, their environment and the artefacts that are created and manipulated as one large cognitive system. The approach originated with Ed Hutchins' work on ship navigation [1] in which he explored the complex system that results in the current position and target position being identified and transformed into the required course to steer. This system involves a series of information transformations through a variety of media including the spoken word, control panel lights and dials, instruments, landmarks and so on. The approach has been used in the analysis of computer-supported co-operative work (CSCW) in order to identify the impact of new or intended technologies on collaborative work such as call centers (e.g. [2, 3]) and communities of practice (e.g. [4]), among other areas. It has also been adapted for use in HCI analyses to support the development of interactive systems (e.g. [5, 6]).

It has been argued [7] that the distributed cognition framework provides a unifying approach to studying socially complex work situations that pulls together different disciplines that have traditionally studied such phenomena, e.g. the cognitive, social and organisational sciences. The framework therefore supports analysis of a situation that takes a more holistic view of the work and its progress.

Although software engineering is recognised as a social activity by many, there have been few reported studies of software development activity using a distributed cognition approach. Flor and Hutchins' [8] study of two programmers working

together during a maintenance activity is the most widely cited application of this theory to the study of software development activity. They observed and recorded two programmers working together on a maintenance task in order to characterise some of the system variables that were important for the success of the task. They did this by analysing the interactive distribution of information used in the task.

The program itself was a graphic adventure game and consisted of about 3000 lines of C code. The change to be made involved adding a 'whisper' command to the program – this command would take a string as input and send that string only to the player indicated. The programmer's interactions were recorded using videotape and all keystrokes and output at the computer terminal were logged. The analysis was performed on a written transcription of the videotape, the commands entered and the times they were entered, and interactions with documentation that were captured on the videotape. Therefore the analysis focused on the detail of the programmer's interactions, but did not consider the wider team or system context.

At the end, they had identified a set of seven properties of the cognitive system that consisted of the two programmers and their immediate environment. These properties were: reuse of system knowledge, sharing of goals and plans, efficient communication, searching through large spaces of alternatives, joint productions of ambiguous plan segments, shared memory for old plans and division of labour. Some of these properties, such as reuse of system knowledge and searching through large spaces of alternatives, have been observed before in studies of software development (e.g. [9]), and some of them have similarities to XP's practices. However Flor and Hutchins considered only one episode of collaborative programming, and they did not attempt to extend their analysis beyond this restricted view.

In this paper we broaden the scope of analysis to consider the whole XP team and its interactions over the course of a week or so rather than focus tightly on the details of one programming episode. The cognitive system under scrutiny therefore is the XP team and its environment. To do this, we discuss the results of observational studies with four mature XP teams working on different applications and in varying environments. In the next section we characterise the information flows through and around each of the four teams. Then we describe in more detail the approach to distributed cognition that we adopt in this paper. In section 4 we present distributed cognition accounts of these teams, and in section 5 we highlight breakdowns that we have observed. In the final section, we conclude that looking at teamwork through the lens of distributed cognition allows us to identify potential issues regarding information flow and transformation within and between an XP team and its environment.

2 Information Flow and Transformation Within the Four Teams

In our previous work, we have found that stories in XP are a key mechanism for capturing and propagating information throughout the XP team (e.g. [10]), and so the description of our teams and their information flows focuses on the generation and manipulation of stories. Further information about teams B, C and W can be found in [10, 11, 12, 13].

Team B produced software applications in Java to support the management of operational risk within a large bank. They were organised into two sub-teams. Stories

were generated during the planning game with developers and the customers present. They were captured on index cards. The developers estimated the story cards and wrote the estimate on the cards. The cards were then dealt onto a table, sorted through and rearranged before being placed on a portable board. Throughout the iteration, the cards were treated as tokens for work to be done, and developers would take the cards off the board and use them to generate code. Continual dialogue, focused around the story card, took place between the developers and the customers in order to clarify and expand the story content.

Team C developed web-based intelligent advertisements using Java. The customer role was carried out in this team by marketing personnel who were in regular contact with the client. The marketing personnel generated the stories following discussions with the client, wrote them onto index cards, and prioritised them in the planning game. Developers wrote estimates on the cards and those for the current iteration were displayed on a common wall. At the end of an iteration a summary of the stories completed, started and abandoned during the iteration was written to a wiki site and the cards were put into storage. Cards that were obsolete or superseded were torn up and not kept.

Team S worked in a large international bank, and programmed in Java. Their project concerned the migration of database information from several smaller databases to one large database. The work to be completed was controlled by the project manager of the team, who was not himself one of the developers. The stories were developed from the overall project plan which listed the functionality to be implemented. Stories were prioritised through consultation with the business analysts and the developers. Once written on index cards, the stories were estimated and the cards displayed for all the team to see.

Team W were part of a medium-sized company producing software products in C++ to support the use of documents in multi-authored work environments. Within each iteration, the team organised itself into sub-teams oriented around the various software products or related issues. Stories were generated by the programme managers who were hybrid figures with some technical and some business expertise. They liaised with the marketing product manager on the customer side and the developers on the software side. Stories were captured and manipulated using a purpose-built software package. Developers looked at the online story and estimated the time required to complete it. Testing information was stored alongside functionality information, and the system underwent a tiered set of tests - developers were responsible for unit tests, testers (a separate element of the team) tested the stories in context, and the QA department (quality assurance) tested the whole product.

Each team was observed for about a week; our observations focused on the interactions between team members and their environment, and the data collected included contemporaneous notes, photographs and some audio recordings.

3   Analysis Approach

The initial analysis of our data followed a rigorous approach in an ethnographic tradition. This approach involves seeking counter examples to any suggested finding. For example, where we observe that developers in a team preferred to work on problems together, we

also seek evidence in our data that would underpin the opposite result, i.e. examples where team members chose to tackle similar problems on their own.

A distributed cognition analysis takes the view that a cognitive system extends beyond what goes on in an individual's head and encompasses the wider interactions and information transformations that are required in order to achieve a goal. A distributed cognition analysis typically involves examining the following aspects of the cognitive system [14]:

- The distributed problem-solving that takes place;
- The role of verbal and non-verbal behaviour (what is said and what is not said, but simply implied, are equally important)
- The co-ordinating mechanisms that are used;
- The communicative pathways involved in a collaborative activity;
- How knowledge is shared and accessed

It also investigates the information flow through the cognitive system and identifies where 'breakdowns' may occur. Breakdowns are potential failures in communication or information flow that will impair the system's performance or prevent the system from achieving its goals. More formal breakdown analysis has been used to investigate collaborative software systems (e.g. [15]).

In the accounts that follow, we address each of these issues in turn, drawing on the observational data we have collected. In each case and before making an observation, we have carefully considered whether we have evidence to contradict the statement we want to make.

4   Accounts of XP Teams

Developing a software system requires access to a lot of information. Fundamentally, there is the set of requirements for the software, but in order to produce the required software, information regarding deadlines, estimates for completion, responsibilities, the status of code under development, criteria for assessing when code is complete, priorities regarding which pieces of software to work on when, technical details of a language or infrastructure, and so on. To follow the detailed information flow paths for each of these would require more space than this paper allows, and indeed our data is not detailed enough to support such an analysis. Instead, our accounts give a broader view of information flow, transformation and application. A more detailed study is left to another day.

4.1   Distributed Problem-Solving

Problem-solving was highly distributed in all of our teams, both across people and across time.

One example of this is the use of pair programming to develop code, and each of our teams saw pairing as an essential part of their normal working practice. Having said this, observed behaviour did vary. For example, Team C paired for all of their tasks with very little change from this routine, while Team S paired for the majority of tasks when there was an even number of developers available, and the smaller sub-team in Team B had an odd number of developers and hence could not work

exclusively in pairs. Team W's working moved seamlessly from singletons to pairs and three-somes and occasionally into a larger group of team members. So problem-solving is usually distributed between at least two programmers, and between more than two if the situation demands it.

The distribution of problem-solving responsibility would extend to the team's customers, as needed. Each team's involvement with the customer varied, depending in part on the nature of the application and their availability. For example, Team S had very little contact with the customers of the system, but had a lot of interaction with the business analysts who knew the database structures and their uses. In this team, a discussion between a pair of developers to understand the problem was often extended to include the analysts. However analysts were not observed referring to the developers for help with solving the problems they encountered. Team W interacted mostly with the programme managers, and programme managers would frequently work with the marketing product managers.

One characteristic of problem-solving within our XP teams was that information was available from a number of sources including other members of the team, the customer, text books, intranet wikis and internet developer sites. For example, members of Team S would regularly consult a reference book, an online developer site, database documentation and the project manager in order to solve a problem.

Each person within a team was actively engaged in solving the problem as appropriate for their expertise. This manifested itself through the problem-solver actively seeking out the individual with the required expertise, but also individuals offering their help where appropriate. For example, in none of our teams did we observe a team leader nominating developers to help with an identified problem – people organised themselves to obtain and offer the appropriate advice. Where this involved one individual interrupting the work of another, both the interrupter and the interrupted respected each other's needs and worked together to find a suitable answer to the issue at hand.

This kind of distributed problem-solving, i.e. distributed across people, was observed on a daily routine basis. In addition to this, problem-solving is distributed over time. Test-first development means that the design of some code is considered before coding begins. Then, as the code evolves, we observed that issues may be raised during stand-up meetings, at iteration planning meetings and during lunch, coffee breaks and informal get-togethers. During our study of Team B, a particularly complex story exceeded its estimate and extended over more than one iteration, but the team persevered as they recognised it as a key part of the functionality.

4.2 Verbal and Non-verbal Behaviour

The character of each team we have studied is very different in terms of size, programming language, organisational setting, team composition and outlook. However each team had a keen sense of purpose and enthusiasm for their working. Team W for example appeared to work in a very solemn and serious atmosphere, while the atmosphere around Team S was much more relaxed. However all teams relied heavily on verbal communication. They were very sensitive to the need to talk with each other within the team and with customers or with others who had the right expertise. For example, when Team C faced a technical problem that they could not

solve internally, they had no qualms about contacting an outside consultant for advice.

Individuals vary in how they best communicate ideas and thoughts. For example in Team S there was one individual who liked to write down notes, draw diagrams and generally doodle while exploring an issue. Any pairing session he was involved in produced and relied on a large collection of these notes and diagrams. On the other hand, another of the team members wrote only short notes on index cards, while another was not observed writing any notes at all. When these latter two team mates were pairing they did not talk very much, but often would turn to each other and appear to be seeing and manipulating an artefact in between them (which presumably represented the code, or the problem they faced). At these times, they spoke little except to make comments about the common artefact that they were both working on.

Although the purpose of pairing is to produce code, the process of pairing is fundamentally about communication – both verbal and non-verbal. We have observed elsewhere that this interaction is much like a three-way conversation [12], with developers occasionally talking directly with each other, sometimes interacting through the code and sometimes interacting directly with the code while the other developer watches. This intense three-way relationship introduces different ways of communicating; both developers typically engage in talking, typing, and gesturing - using the cursor and highlighting techniques to focus attention. In addition, the ability for pairs to overhear and be overheard appears to support the distributed nature of problem-solving where relevant expertise is offered when it is needed.

Other examples of the effect of non-verbal behaviour are the unannounced start of a stand-up meeting, and the use of a non-verbal noise to communicate information. In the former case, Team S did not often have to announce the fact that it was time for the daily stand-up meeting. When the time approached, team members would automatically congregate at the appropriate spot and the meeting would start. In Team C, a non-verbal noise was used to signify the release of tested code into the code base; in this case it was an artificial animal sound.

4.3   Co-ordinating Mechanisms

There were broadly two different types of co-ordination that we observed in our XP teams: regular and *ad hoc*. We first consider the regular mechanisms. Team C relied entirely on the manipulation and display of story cards, the planning game, and daily stand-ups for co-ordination. Team S also relied on these but in addition the project manager held the overall project plan from which story cards were generated. Team B used a similar approach to Team S. Team W did not have physical story cards, but kept their stories within the supporting software environment. This meant that the detailed manipulation of stories was not clearly visible, although they employed summary flip charts which showed which stories were being worked on and who was in which team.

All the teams were self-organising and hence there was little or no co-ordination imposed from the team's higher management. The key regular co-ordinating mechanisms were therefore the story cards, the planning game and the daily stand-ups.

Supporting this more regular co-ordination was the wide spectrum of *ad hoc* meetings, peripheral awareness, and fluid pairing situations, as we discussed above.

The stand-up meetings, for example, would be less effective if these other mechanisms were not in place.

4.4 Communicative Pathways

In general, communicative pathways in our XP teams were simple. In all cases, developers had direct and regular contact with the customer or the customer's representative, and they had clear and uncomplicated access to fellow team members and local, relevant expertise. The story was a key focus of all communicative pathways.

Within the team, communication happened through a network rather than along a single pathway. As we have discussed above, the teams all had effective ways of keeping each other informed of development issues, and the team members volunteered information when they felt it was relevant.

In Team C, if an issue arose with a customer, then the marketing person assigned to that customer would talk directly with the developer(s) working on the relevant part of the software. It was noticeable, however, that the marketing personnel would not walk straight into the developer 'pens', but would wait around outside until they were noticed by the developers (see [10] for more detail). In Team W, information regarding the wider product picture was communicated via marketing product managers or other senior staff on a regular basis, as and when there was something to report.

4.5 Knowledge Sharing and Access

Collective ownership is one of the practices that underpins XP. Hence it is no surprise that we found knowledge sharing and access to relevant expertise to be well-supported. For example, in all teams, pairs were formed explicitly on occasions to provide a balance between experience and novice status in order to expose novices to areas of the code that they did not know.

The most public evidence of knowledge sharing and access was the use of information radiators [16] to show the current status of the stories. We found these in all teams. Even in Team B where the organization rules regarding the sticking of items onto the walls prevented them from using a traditional board, they used a portable board or flipchart.

The above descriptions have painted a picture which implies that all developers are equal in terms of their capabilities and their 'specialisms'. Indeed we have not mentioned specialisms before. In Team C for example, there were eight developers, but also one graphic designer and one IT support manager. The graphic designer was not a Java programmer. She produced HTML and graphical images but in order to also gain an appreciation of the concerns of the developers, she would often pair with one of the Java programmers. In these circumstances her contribution to the development of code was minimal but the team all felt it important to share knowledge in this way. The graphic designer and IT support manager would often work together on tasks.

In other teams where 'specialists' did not pair with developers, all team members were actively involved in the daily stand-ups and other regular meetings. For

example, Team W included a technical author, web developers and two testers who did not pair with developers, but they attended the meetings.

4.6 Potential Breakdowns in Information Propagation

In each team, we found evidence of breakdowns, or potential breakdowns.

Informal communication can breakdown when the parties involved don't have a common memory of the conversation and what was decided. In Team C the planning game involved estimating the cards and in order to do that it was necessary to gain a level of understanding about how to implement a solution, and that required some design. However the design used as the basis for estimating was not documented and it was not uncommon (as reported to the researcher) for a different design to be implemented and the estimate to be compromised. This illustrates the potential of collective memory to fail. This is not necessarily a problem situation provided the changed design and its rationale are communicated to others, but it is a potential breakdown.

Communication can also break down where there are 'too many' information flows. Team W did not, at the time of the study, use physical story cards, but stored information in an online internal software system. This had several advantages, including the fact that significant information could be kept alongside the main story, including acceptance tests, modifications, estimates, a history of who worked on the code, and so on. However we observed a situation where a story number was transcribed incorrectly, which led to a tester running the wrong acceptance test against a story. Due to the nature of the story and of the application it took significant time to realise that the code he had downloaded was not the code he should have been running against the given test. There are many possible ways that this could be avoided, e.g. better structuring of information online, double-checking of codes and tests, automatic linkages between code and tests, etc. However, it is interesting to note that shortly after we had completed our study in this organisation, they introduced physical story cards.

XP teams rely on self-managing and self-organising individuals who are prepared and able to take on responsibility for their work. This has significant advantages, but one consequence of this is that individuals regard their time as precious. We witnessed a situation in Team S which illustrated this. The project manager called a meeting of the developers (at the time, only four of them were in the office) in order to discuss a significant technical issue. One of the developers was unhappy as he did not understand why they were discussing this issue, nor the purpose of the meeting (i.e. what is going to be the result of this meeting). In this situation, the project manager was sharing information, but had not adequately explained the issue's significance or the meeting's purpose. Interestingly the other team members made considerable efforts to ensure that the unhappy developer was calmed.

The potential breakdown we identified in Team B revolves around the organisation's internal procedures and the team's expectation of timely feedback. Once the software had been tested internally, the software was handed over to another part of the organisation to run the acceptance tests, and this part of the organisation did not operate under XP principles. The consequence of this is that results from the acceptance tests were fed back to the developers 4 or 6 weeks after they had finished

working on the software. This caused considerable consternation as the team had been working for several weeks on software that did not pass the acceptance tests.

We mentioned above the reliance of regular co-ordination mechanisms on the more *ad hoc* mechanisms. We did not see any examples of the *ad hoc* mechanisms failing, but if they did our analysis suggests that the regular co-ordination mechanisms might also suffer.

5 Discussion

One of the consequences of the XP approach to development is that much of the knowledge and expertise required to solve problems that are encountered is available quickly and easily in a form that can be immediately applied to the existing situation. For example, because one of the XP practices is collective ownership, all team members have a good understanding of the context of any problem that arises. This means that the time needed to explain the problem is minimised, and the applicability of potential solutions can be assessed rapidly. One way of expressing this is to say that the team members have sufficient common ground to be able to communicate effectively. Common ground is a key concept in co-ordination activities and without it collaborators need to express every detail explicitly [17, 18]; the discussions above indicate that XP teams need to maintain considerable common ground. There has been much debate about how to choose programmers to join an XP team. There is wide consensus that the new programmer needs to be compatible socially with the other team members, but we would also suggest that the level of common ground between the new programmer and the existing team (in terms of technical knowledge and experience, or in the specific application domain) will affect their compatibility with existing team members.

The main transformation taking place in this cognitive system is that of transforming the story into executable code. There is very little information propagation outside the story – the story remains the central focus of development from the time it is created until the code is handed over. One reason that these simple flows are sufficient for the team's needs is that the work is divided into small manageable chunks, thus restricting the amount of information needed to complete the story.

6 Conclusion

Looking at XP teams using the framework of distributed cognition shows us that XP teams use a simple flow of information that is underpinned by shared understanding of the software under development and sufficient common ground to support effective communication. To achieve their goals, XP teams tend to work in information-rich environments with easily accessible, easily applicable knowledge. Individual team members put effort into making sure the cognitive system performs as it should. The regular co-ordination mechanisms used, for example, would not be as effective if the more *ad hoc* system were to stop working. XP has a deep cultural attachment to close-knit, informal settings. In this analysis, we have indicated the benefits of this kind of setting for effective working. Potential breakdowns we have identified stem from a disturbance of the simple, coherent cognitive system we have described.

Conclusions from the work reported here fall into two areas: practical implications, and research implications. We suggest that practitioners study the potential breakdowns identified in Section 4, and consider whether any of these situations applies to their own circumstances. For researchers, we would argue that the analysis presented in this paper has shown the potential of distributed cognition to shed light on information propagation within an XP team from a novel perspective, but that this work has only just begun and there is clear scope for more, in-depth studies.

References

[1] Hutchins, E. (1995) *Cognition in the Wild*, Cambridge MA: MIT Press.

[2] Halverson, C. A., (2002) Activity theory and distributed cognition: Or what does CSCW need to DO with theories? *Computer Supported Cooperative Work*, 11:243-267.

[3] Jones, P.H. and Chisalita, C. (2005) Cognition and Collaboration – Analysing Distributed Community Practices for Design, *CHI 2005 Workshop*, April 2005, Portland, Oregon.

[4] Hoadley, C.M. and Kilner, P.G. (2005) Using Technology to Transform Communities of Practice into Knowledge-Building Communities, *SIGGROUP Bulletin*, 25, 1, 31-40.

[5] Hollan, J. Hutchins, E., Kirsch, D. (2000) Distributed Cognition: Toward a new foundation for human-computer interaction research, *ACM Transactions on Computer-Human Interaction*, 7(2), 174-196.

[6] Wright, P.C., Fields, R.E. and Harrison, M.D. (2000) Analyzing Human-Computer Interaction as Distributed Cognition: the resources model, *Human-Computer Interaction*, 15, 1-41.

[7] Rogers, Y. and Ellis, J. (1994) Distributed Cognition: an alternative framework for analyzing and explaining collaborative working, *Journal of Information Technology*, 9, 119-128.

[8] Flor, N.V. and Hutchins, E.L. (1992) Analyzing distributed cognition in software teams: a case study of team programming during perfective maintenance, *Proceedings of Empirical Studies of Programmers*, 1992

[9] Detienne, F. (2002) *Software Design - Cognitive Aspects*, Springer-Verlag, London.

[10] Sharp, H. and Robinson, H. (2004) An ethnographic study of XP practices, *Empirical Software Engineering, 9*(4), 353-375.

[11] Robinson, H. and Sharp, H. (2004) The characteristics of XP teams, in *Proceedings of XP2004* Germany, June, pp139-147.

[12] Robinson, H. & Sharp, H. (2005) The social side of technical practices, in *Proceedings of XP2005*, LNCS 3556, 100-108.

[13] Robinson, H. and Sharp, H. (2005) Organisational culture and XP: three case studies, in *Proceedings of Agile 2005*, IEEE Computer Press pp49-58.

[14] Preece, J., Rogers, Y. and Sharp, H. (2002) *Interaction Design: beyond human computer interaction*, John Wiley & Sons, Chichester.

[15] Scrivener, S., Urquijo, S.P. and Palmen, H.K. (1993) The Use of Breakdown Analysis in synchronous CSCW system design, *Proceedings of ECSCW*, 517-534.

[16] Beck, K and Andres, (2005) *Extreme Programming Explained: Embrace Change* (2nd edition), Addison-Wesley.

[17] Clark, H. and Schaefer, E. (1998) Contributing to discourse, *Cognitive Science* 13, 259-294.

[18] Flor, N. (1998) Side-by-side collaboration: a case study, *International Journal of Human-Computer Studies*, 49, 201-222.