5

THE INTERPLAY BETWEEN COGNITIVE AND ORGANIZATIONAL FACTORS IN SOFTWARE DEVELOPMENT

P.E. Waterson, C.W. Clegg and C.M. Axtell

Institute of Work Psychology
University of Sheffield
Sheffield S10 2TN
(Tel +44-114- 2756600 E-Mail: P.Waterson@sheffield.ac.uk)

ABSTRACT: This paper describes a case study of large scale programming in a commercial context. In particular we chose to study the relationship between the way in which work is organized and allocated, the knowledge and expertise of project members, and the use of programming tools. Our findings point to a dynamic interplay between these factors which partly reflects evidence of opportunistic and planful behaviour, as well as the importance of patterns of collaboration which arose over time within the project. We conclude with recommendations for the training of developers as well as the need for modifications and adjustments of current models of the software process and lifecycle.

KEYWORDS: Systems development, division of cognitive labour, programming tools, knowledge and expertise

1. INTRODUCTION

Software development can be seen as a problem solving process which involves multiple agents, sometimes with competing goals and responsibilities (Curtis, Krasner, Shen and Iscoe, 1987). In the process of building a large scale program for example, a number of individuals are likely to be involved. These range from programmers and analysts involved in designing and writing the program, to managers and team leaders responsible for the day-to-day operation of the project. Added to this list are the end users of the system who may be involved at various stages in its lifecycle, customers who provide the original requirements, as well as other parties who may be called upon for their specialised knowledge (eg. database experts). Whilst the degree of involvement of individuals with particular specialisms, knowledge and expertise may vary, the essential point is that software development involves a variety of cognitive and organizational issues, for example concerning the communication and coordination of knowledge relating to the program and the achievement of mutual understanding regarding its contents.

Few studies exist of groups of programmers working together on a program, and even fewer of such activity taking place within an industrial or commercial context. The most thorough set of field investigations of the role that cognitive and organisational factors play in software development can be found in the work of Curtis and his colleagues (Curtis, Krasner and Iscoe, 1987; Walz, Elam and Curtis, 1993).

Curtis et al (1987) found that three main factors contributed to problems in building complex programs within tight deadlines. First, many projects were hampered by the fact that knowledge relating to the application was thinly spread. Second, the volatility of customer requirements meant that many changes to the program had to be made at very short notice. And third, communication and coordination breakdowns between the various parties frequently led to delays. These problems adversely affected both the productivity of the process of development, and the success of the actual product itself. Above all, Curtis et al stressed that the problems faced by software projects could not be attributed to one main psychological cause, but critically involved the combined effect and interaction between underlying cognitive, social and organizational processes.

In order to follow up the findings from the Curtis et al study, and to explore further the nature of the dynamic interplays between individual and organizational levels of analysis, we present a case study of large scale, commercial software development. In particular, we chose to study in detail a set of questions which previous studies of software development have shown to be salient. These were:

(1) How is work organized on projects, and what consequences does this have for the progress of software development?

K. Nordby et al. (eds.), *Human-Computer Interaction*

(2) How is the knowledge and expertise available on projects distributed, and how is effective sharing of this knowledge achieved or hindered within projects?
(3) What is the impact of software support tools (eg. CASE tools) on the way work is organized and on the distribution of knowledge and expertise?

2. CONTEXT AND METHODS

The study was carried out in the information technology department of a large financial services company based in the UK. The Charging project (as we have called it) involved approximately 100 people from different sections of the IT and Finance departments. About one third of these people were end users of the system who were involved at different stages in the development of the program. The Charging program is designed to cope specifically with the management of information relating to corporate (ie. business) accounts.

During our visits to the organization we conducted 30 interviews with key personnel on the Charging project including the project manager, the team leaders, senior analysts and programmers, as well as other people involved directly or indirectly with the project (eg. senior managers, user acceptance testers, database and CASE experts). In addition we conducted a number of informal interviews with project members and observed the work of project teams when time and access allowed. Participants, along with the project and organization, were guaranteed anonymity, and are disguised in this paper.

Participants were encouraged to discuss their problems, as well as events and challenges which they thought were important during the development of the project. Interviews usually took place in the work situation; this allowed us to ask questions relating to programming tools in the context of on-going work, as well as carry out informal observation of the work of the project.

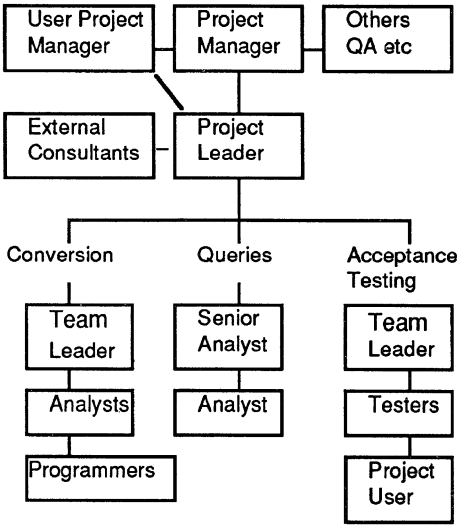
3. FINDINGS

3.1. Organization of project work

At the time we began our study approximately 30 people were involved in working on the program. Three activities occupied the main work of the project members: (1) conversion of code from the existing charging system so that it matched the requirements of the new system -- this was carried out by a group of analysts and programmers; (2) testing of the code by user acceptance testers and users from the finance area seconded to the project; (3) dealing with e-mail queries from the external contractors regarding the functional specification of the program and questions relating to the writing of the program code -- these were dealt with by the senior analyst and his assistant.

Figure 1 is a diagram of the management structure of the project. Each of the above three teams was allocated a team leader who was responsible to the overall project leader, who in turn, reported to the project manager. During the course of the project a number of other groups within the IT department were also involved, for example in quality assurance, database administration and CASE tool support. In addition, the project manager and project leader liaised with the user project manager located in the finance department; he was responsible for ensuring that the program met its original requirements specifications.

Figure 1: Diagram of Management Structure of Charging Project

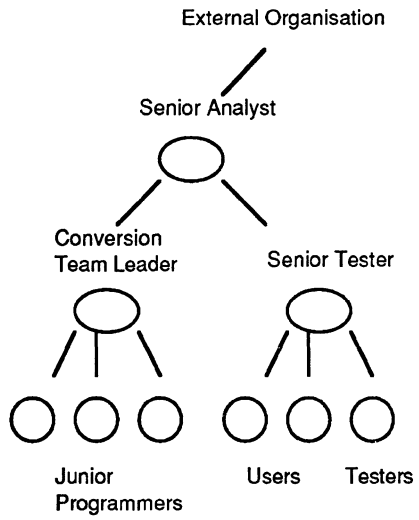


Many project members admitted that they had to conform to strict deadlines, and found their work stressful. An incoming request for a change to the program or clarification of existing code had to be dealt with very quickly in order to avoid penalties. At other times however, the project was relatively relaxed, particularly when a major task (eg. code conversion) had been completed. In order to cope with these demands, the structure of the teams fluctuated. Thus although the essential composition of teams remained more or less constant, analysts were seconded to help out with the work of other teams when and where necessary. Typically a team leader would report to the project manager that more staff were needed. As a result staff would be drafted in from other teams. In rare instances personnel were drafted in from other projects in order to help cope with the demands of the work. It was also common for project members to "rally round" to cover fluctuations in workload. In this way the project was able to adapt to circumstances and at the same time allow project members to gain an understanding of the work of other teams. The temporary restructuring of teams also facilitated

the achievement of a shared understanding of the program as well as ensuring that project milestones and deadlines were met.

One of the limitations of organizational charts is that they are abstractions; in particular, figure 1 is not very informative with regard to the communication links and information flows within the project. Figure 2 represents the team structure and communication channels within the project.

Figure 2: Structure of Teams within the Charging Project.



The Charging project fits the pattern of a "controlled decentralized" team structure (Mantei, 1981); that is, programming groups are partitioned according to the type of role they play in the project. The precise allocation of roles within the project was the responsibility of the project manager who acted with the team leader responsible for queries to recruit members of the project from the "pool" of developers which existed within the IT department. In practice this team leader was senior to the others.

Communication between the various project teams was on the whole limited to a few key members of these groups. The majority of individuals spent their time working on a specific aspect of the program. A typical member of the acceptance/testing team, for example, would use examples of inputs to the program as test materials, whilst a programmer working on the conversion team would take a module within the existing system and write code which would enable it to interface with the new system. Boundaries between teams were relatively impermeable for these project members.

In common with the findings of other research (eg. Krasner, Curtis and Iscoe, 1987) a small group of individuals, usually the team leaders, acted as "boundary spanners" between the project teams. An example of a "boundary spanner" is the team leader responsible for the queries team, working between the external contractors and the rest of the Charging project. For example, he resolved an emerging ambiguity regarding the functional specification of the program. This involved consulting with other project members to check that this problem had not occurred before or was resolvable. This communication usually took the form of informal discussions between the boundary spanner and team members.

Within teams a clear hierarchy of access and opportunity to communicate information was apparent. Problems were generally taken to a team leader and only then passed upwards to management. Within the project, the team leaders were responsible for communicating the outcomes of work in their teams to other team leaders and managers. Likewise, these individuals acted as "gatekeepers"; part of their role was to disseminate information and make sure it was acted upon when necessary, and also to liaise with others in a similar role but in charge of different parts of the work.

Smooth and successful coordination throughout the project was largely due to the activity of the boundary spanners, particularly when technical problems occurred (eg. a CASE tool wasn't working properly) or information was required regarding the program (eg. clarification of the function of a sub-procedure). No direct procedures governing the activity of these individuals had been formalised on the project; rather project members used a "support network" which had evolved over time and depended upon the sharing of expertise and knowledge regarding the program and other, mainly technical concerns.

3.2 Distribution of shared knowledge and expertise

As a number of authors have pointed out (eg. Boehm, 1981), the expertise and knowledge of project personnel is a major factor in determining the likelihood that a new system will meet its requirements. Key sources of expertise which need to be available to project members include technical expertise relating to the hardware and software aspects of the program, as well as knowledge relating to the nature and use of the intended application.

Our interviews with members of the Charging Project enabled us to gain an understanding of the flow of information within the project as well as the relative location of sources of expertise and knowledge that project members made use of. On the basis of our interviews we categorised the expertise and knowledge of project members as falling into one or more of five distinct categories. None of the project members possessed knowledge

in one category alone and, with one exception, no single individual had extensive knowledge in all of the five categories. Rather, knowledge and expertise were distributed amongst the project members.

Table 1: Types of knowledge within the project

Type of Knowledge	Components
Computational	Algorithms, data structures in the program
Application	Typical inputs and outputs of the system
Domain	Use of the system (eg. work activities)
Project Management	Budgets, timescales, project resourcing
Software Engineering	Data modelling, use of SE techniques and tools

An important difference between these categories and those used by previous researchers (eg. Curtis et al, 1988) is our distinction between application and domain knowledge. The reason for splitting knowledge relating to the use of the final system into two categories arose because some project members knew a good deal about the inputs to the program when it was being used (particularly when these inputs were used as testing material by the acceptance testing team), whilst others knew more about the working practices associated with the use of the system (eg. day to day requests made by finance users and working practices in general).

Application knowledge was quite widely spread throughout the project; most members of the project teams had some experience of dealing with inputs and outputs from the program (eg. calculations of interest and database records). By contrast, only the users located on the acceptance testing team had extensive knowledge of working practices associated with the system (ie. domain knowledge); as a result these users were relied upon not only to check and test the program within their team, but also to answer questions and give advice with regard to the way the system was likely to be used. Part of the function of users on the project was thus to "contextualise" the knowledge of other project members and facilitate a mapping between application and domain knowledge.

Project members also acted as a resource for one another with regard to other specialised knowledge which was located on the project. For example, during the course of the project a new set of CASE tools was introduced to cover aspects of the analysis, design and coding of the program. The majority of project members were unfamiliar with the use of these tools and relied upon one or two individuals who had experience in components of the tools (eg. decomposition and entity-relationship diagrams) and their use. Again, no formal procedures had been set in place for this type of help facility; instead project members utilised and built up the support network as time went by. In a sense, group members designed and established their own help systems.

An important aspect of the coordination and communication of knowledge and expertise was the role played by individuals who acted as a boundary spanners. Typically the knowledge possessed by boundary spanners overlapped the categories described in table 1. Similarly, other boundary spanners possessed detailed expertise in one of the categories in table 1 along with knowledge and experience from other categories. Within teams these individuals frequently acted as educators or disseminators of information. In addition they spent some of their time gathering information from other teams regarding their current work and problems.

The activities of key individuals facilitated the flow of information within teams as well as across the project as a whole. Information could take a variety of forms, sometimes being problem-centred (eg. an ambiguity in the program specification), or more proactive (eg. details of the progress of other teams). Fortnightly meetings between the project manager and leader and the team leaders partly served the function of discussing and updating the work of the project, but also allowed members to gain an understanding of the final system in terms of the specifics of the program, as well as its relationship to the practices and requirements of end users.

In one important case decisions were dependent upon the knowledge and influence of one individual in particular. This role is described in previous research variously as that of "project guru" or "super-conceptualiser" (Curtis et al, 1988).Within the Charging project the leader of the queries team was consistently described as the "lynchpin" of the project. His main function was to deal with requests for clarification and updates of programming problems from the external contractors via an e-mail link. The knowledge of this individual was clearly vital to the work of the project, having worked at every stage of the program's development including the writing of the original functional specification. His influence on decision-making was crucial to the day-to-day activity of the project. Not surprisingly, this individual was called upon to clarify problems and to educate other project members in the details of the program.

3.3 Use of programming tools

A number of different types of computer systems were used by members of the project during the development of the program. These ranged from typical applications to be found in most offices (eg. word processing packages, spreadsheets), to tools specifically designed to deal with project management (eg. to monitor the progress and

scheduling of project deadlines and milestones). In particular a set of CASE tools were introduced during the project; these were for use in the analysis, design and coding of the final system.

In the Charging project, the allocation of tasks and responsibilities would normally take the form of analysts and programmers working collaboratively on the program and dividing up the work to be done between them. A number of project members, particularly those associated with the conversion of the existing system, viewed the introduction of a tool to handle design as problematic mainly because it "distorted" the tasks normally given to programmers and analysts. This led to confusion about who was working on a particular part of the program. A major complaint was that many tasks were being inefficiently carried out by a number of people in parallel and that this had led to needless repetition of work.

An example can be seen in the processes which individual programmers routinely carried out when working on the program. As other research has shown (eg. Guindon, 1990) the process of design often involves carrying out a number of tasks in parallel, tasks which typically include opportunistic planning and reasoning about changes to a program which are not carried out immediately but are stored as partial solutions to plans and acted upon later. Within the project a number of individuals described how they made decisions when first faced with a program. This process sometimes involved deciding to split a program module down into smaller components which fell into common units, or alternatively modules which were too large to be handled as one unit and could be made easier to design if split into two of three sub-components. A common complaint was that by automating this process these decisions had been taken out of the control of the programmer and given to others who were not within the team and whose decisions could not be communicated within the team and the project (eg. CASE specialists). In other words, work was not only being needlessly repeated but also it could not be related to the activity of the team as a whole.

Activities such as boundary spanning applied not only to routine discussions of the collective work of the project but also allowed an overview, or mental model, of the system to be developed and communicated. The introduction of CASE changed this process and disturbed the communication patterns which had evolved as the system was being developed. Together these problems could be seen as symptomatic of the technology-led approach to the implementation which the larger organization (Symon and Clegg, 1991) had decided to adopt. Most effort was being placed on re-organising the project around the demands of the tools, rather than vice versa.

4. CONCLUSIONS AND IMPLICATIONS

Within the project there evolved over time a clear division of labour regarding the tasks which had to be undertaken in order to build the program. The mechanisms by which this came about reflect on the one hand decisions made by management regarding the allocation of responsibility at the start of the project. Over time the spontaneous formation of coalitions and patterns of interaction restructured the project less along hierarchical lines and more around the skills and expertise of individuals. In many respects the social system which resulted involved a mixture of planful and opportunistic behaviour (Broadbent, 1993).

One of the major successes of the project was its ability to reallocate and re-negotiate tasks and responsibilities according to cognitive demands and constraints. Collaboration within the project was the norm, and rather than being centred solely around the technology to be used, was determined by people and their skills. The precise form of what might be termed the "division of cognitive labour" (Hutchins, 1991) changed according to the type of tasks to be completed and the availability of knowledge and expertise.

According to Hutchins' description of collaborative work, the project can be seen as a set of mutually adaptive sub-systems where the computation of programming tasks was definable in terms of the needs of the project. The term "computation" is used to refer to the set of tasks which have to be distributed amongst team members. Although the project had to deal as a whole with many uncertainties and work on the basis of incomplete and often poor information, it coped with these because it was able to restructure accordingly. Much of the success of these changes can be seen as the combined result of cognitive and organisational processes; boundary spanning allowed knowledge to be shared, whilst the presence of differential sources of expertise and knowledge made sure that problems and queries could be reconciled quickly. The introduction of CASE can be seen as altering not so much the structure and form of the "computations" which the project had to perform, but the relative distribution of these tasks across teams and individuals. On the face of it the tasks which were distributed amongst project members was the same - programmers converted code, analysts were involved in design etc, however, these activities were no longer as clearly defined as they had once been.

One of the most common mechanisms for restructuring tasks and their allocation involved what Mintzberg (1979) refers to as "mutual adjustment", that is, key individuals adjust their patterns of interaction and communication in order to deal with situations which involve uncertainty and require support from others. For example, we

found many instances of the emergence of collaborative practices centred around the provision of help and support. Despite the importance of these processes within the project, we found little evidence that they were regarded as important amongst managers. We suggest that more resources are put into the training of individuals to fit these roles and the provision of more formally designed support networks within projects. (cf. Nardi, 1993).

A major implication of our study is that processes which involve the transformation and exchange of information are largely underplayed in traditional models of the software process. The implicit models embodied in newer tools (eg. CASE), assume a linear sequence of information flow through a project. Our experience suggests that this process involves feedback at every stage and is flexible rather than fixed. The roles which are assigned to people depend on their knowledge, as well as the demands of the on-going task. Depending on the nature of the task, rather than one sole individual, various people may be collectively involved. The problem with current interventions to re-engineer this process relates to attempts to reify the roles of individuals rather than recognise that at any one time many different, and sometimes unrelated, activities involving knowledge sharing and acquisition may be in operation (Walz, Elam and Curtis, 1993).

Much the same criticisms apply to the way in which management views the software process as it occurs within teams. Here a strict division of labour is assumed, programmers are assumed to be concerned simply with coding, whilst in reality their roles involve the integration of information from a number of sources. In common with other work which has detailed the effects of the technological innovations (eg. Orlikowski, 1993; Bowers and Rodden, 1993), the problematic implementation of CASE can be seen from a number of viewpoints each centred around a "triangulation" between the needs of the individual, the project or larger organization, and the demands involved in using the tools

Above all, we believe there is a greater need for case studies of the type we have described. Future work should address the question of how the assignment of project roles and responsibilities interact with spontaneous patterns of work organization, expertise and tool usage, and the consequences this has for overall project productivity.

REFERENCES

Boehm, B.W. (1981), *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall.

Bowers, J. and Rodden, T. (1991), Exploding the interface: experiences of a CSCW network. In *Proceedings of INTERCHI '93*, New York: ACM Press.

Broadbent, D.E. (1993), Planning and Oppportunism. *The Psychologist: Bulletin of the British Psychological Society*, **6**, 54-60.

Curtis, B., Krasner, H. and Iscoe, N. (1988), A field study of the software design process for large systems. *Communications of the ACM*, **31**, 1268-1287.

Curtis, B., Krasner, H., Shen, V. and Iscoe, N. (1987), On building software process models under the lampost. *Proceedings of the Ninth International Conference on Software Engineering*, 96-103.

Guindon, R. (1990), Knowledge exploited by experts during software system design. *International Journal of Man-Machine Studies*, **33**, 279-304.

Hutchins, E. (1991), Organizing work by adaptation. *Organization Science*, **2**, 1, 14-39.

Krasner, H., Curtis, B. and Iscoe, N. (1987), Communication breakdowns and boundary spanning activities on large programing projects. In G.M. Olson, S. Sheppard and E. Soloway (eds.), *Empirical Studies of Programmers: Second Workshop*, Norwood: Ablex.

Mantei, M. (1981), The effect of programming team structures on programming tasks. *Communications of the ACM*, **24**, 3, 106-113.

Mintzberg, H. (1979), *The structuring of organisations*. Englewood Cliffs, NJ: Prentice-Hall.

Nardi, B.A. (1993), *A small matter of programming*. Cambridge: MIT Press.

Orlikowski, W. (1993), Divisions amongst the ranks: the social implications of CASE tools for system developers. *Proceedings of the Tenth International Conference on Information Systems*, Boston, Mass.

Symon, G. and Clegg, C.W. (1991), Technology-led change: A study of the implementation of CADCAM. *Journal of Occupational Psychology*, **64**, 273-290.

Walz, D.B., Elam, J.J. and Curtis, B. (1993), Inside a software design team: Knowledge aquisition, sharing and integration. *Communications of the ACM*, **36**,10, 63-77.