Communication and co-ordination practices in software engineering projects

Ian R. McChesney a, *, Séamus Gallagher b

a School of Computing and Mathematics, Faculty of Engineering, University of Ulster at Jordanstown, Shore Road, Newtownabbey BT37 0QB, UK
b Division of Management and Information Systems, School of Management and Economics, The Queen’s University of Belfast, University Road, Belfast BT7 1NN, UK

Received 6 May 2003; revised 9 October 2003; accepted 14 October 2003

Abstract

In this paper we report on the investigation, description and analysis of communication and co-ordination practices in software engineering projects. We argue that existing models of the software process do not adequately address the situated, day-to-day practices in which software engineers collectively engage, yet it is through these practices that effective co-ordination is achieved. Drawing on concepts from organizational theory, we describe an approach for studying co-ordination activity in software engineering and the application of this approach to two real-world software projects. We describe key co-ordination issues in these projects and discuss their implications for software engineering practice.

© 2003 Elsevier B.V. All rights reserved.

Keywords: Software engineering; Communication; Co-ordination; Software process; Software project management; Human factors

1. Introduction

A major challenge in managing the software process is achieving the effective co-ordination of its constituent activities, products and people [7,12,17,18,37]. The field of software engineering has seen a number of strategies for improving co-ordination at the organizational, process and tool level. For example, at the organizational level, the Chief Programmer Team arrangement [3] drew on the structure of a surgical team to minimize communication overhead while maintaining the conceptual integrity of the system design. At the process level, structured methods such as Yourdon Systems Method [41] and the Unified Software Development Process [20] have sought to integrate best practice, procedures and techniques for common software engineering tasks, achieving co-ordination through knowledge sharing and standardization. At the tool level, software engineering environments e.g. [2,39] have provided technological support for co-ordinating complex software builds and version control. The ubiquity of large and complex software systems is evidence of the success of previous strategies. However, as demand for software systems increases, software technologies change, and functionality becomes more complex, so communication and co-ordination breakdowns continue to manifest themselves [13,21,37]. There remains considerable scope for improvement both in the practice of co-ordination in software engineering projects and our understanding of it.

In this paper we argue that improved co-ordination will be achieved through a better understanding of how software engineers actually conduct their day-to-day technical work. In particular, we argue that there is a gap between existing process-oriented methods for describing the software process and the situated, day-to-day activities in which software engineers engage when developing systems. A better understanding of co-ordination strategies and practices is the first step in bridging this gap. To this end we present an analytical framework for investigating co-ordinated activity in software engineering and report on its application in two real-world projects.

The remainder of the paper is organized as follows. The next section outlines the relevant literature and presents the analytical framework. Section 3 describes the research...
method used during our field work. Section 4 presents our empirical findings and Section 5 discusses these findings in relation to the gap between software process models and situated practice. In Section 6 we draw some conclusions on our work to date.

2. Literature review

2.1. Software engineering-theory and practice

Much effort in software engineering research has been devoted to the development and deployment of software process models to improve the efficiency and reliability of software engineering projects. This research has matured into the industrial use of standard process definitions, process maturity assessment models and quality management systems such as the Unified Software Development Process [20], the Capability Maturity Model for Software Engineering [33] and ISO 9001 based quality management systems [36]. However, the value of existing process models has been questioned; for example, Fitzgerald asks if development methodologies are premature generalizations from single case experiences [14]. Bach contends that some process models ignore the human dimension of software development [1]. Others accept that process models have a contribution to make, but argue for greater emphasis on the socio-organizational issues at work in software development. Sommerville and Rodden [34, p. 93] note that, in their observations of software projects, ‘there were a large number of co-operation activities which were unplanned (e.g. expert consultation)’, yet these informal, co-operative activities were critical to project success. Sommerville and Rodden also draw attention to the role of the software engineer as a professional, arguing that it is probably impossible to convince engineers that they should follow a single process model. With respect to routine problem solving and task performance, software engineers work in a flexible, locally autonomous, professional fashion, and process definitions should be designed accordingly. Proponents of agile software development emphasize the importance of people over process, arguing that software engineers are to be recognized as responsible professionals and, as such, are the best people to decide how to conduct their technical work [15].

The tension between recommended best practice in the profession and good engineering in the real-world is clearly demonstrated in Button and Sharrock’s ethnomethodological study of a software development project involving Yourdon Systems Method, CASE (Computer-Aided Software Engineering) tools, and the C programming language [6]. The software engineers studied did not follow the prescribed structured methods or organizational process definitions in their day-to-day practice. Instead, they developed adhoc ways of making their formal project structures work (the concept of ‘negotiated order’). Intuitively, as professionals, they knew how to organize their work in order to get it done, even where this meant working in ‘non-standard’ ways to achieve project goals. Central to this idea of negotiated order and situated action is that software engineers communicated, collaborated and co-ordinated within and across teams to achieve project success. Waterson et al. [37] describe how a project’s established, situated practices for coding were disturbed by the introduction of CASE technology. The practice of developing skeleton programs and fleshing these out as requirements firmed up was not compatible with the ‘structured approach’ introduced during the course of the project. These examples are not cases of ‘bad practice’. The rejection of structured methods by practitioners may be for pragmatic reasons and may represent good practice rather than ignorance [14]. In recent years, the emergence of agile process models is one response to this [4,15]. On the premise that traditional structured approaches are highly prescriptive in nature and do not adapt to changing project circumstances, agile models aim to be highly adaptive and apply only as much process as necessary to achieve project success.

Attempts to improve software development must begin by focusing on how the people involved organise themselves and communicate their intentions [5]. Yet this is a situated phenomenon of achieving negotiated order not addressed by existing software engineering process models. So how can we narrow this gap between procedure and practice? Studies such as Button and Sharrock’s begin to answer this question, as they describe the way in which engineers make sense of this gap by engaging in co-ordinated activity which works around the day-to-day problems encountered. Such studies provide essential insight, but not necessarily a mechanism for describing such behaviour. The motivation of our work, therefore, is to identify and describe this co-ordinated activity in such a way that it can be analyzed and transferred from a local project setting for the benefit of other software engineering practitioners.

2.2. Organizational theory

We employ three concepts to describe and analyse the situated co-ordination practices of software engineering teams. First, Malone and Crowston’s [23] primitives of coordination theory are used to identify the co-ordination dependencies and mechanisms at work in software engineering teamwork [25]. Co-ordination theory provides a vocabulary and simple taxonomy for describing co-ordinated activity in a consistent way, and work elsewhere has shown its relevance in the context of software engineering [11,16,25]. Second, we use the notion of communication genres as a device for identifying and describing communicative activity in teams. Communication genres complement co-ordination theory by providing a more detailed template for the analysis of semi-structured communication processes. They also have previously been applied in the context of software engineering [29]. Third, we adopt Weick and Roberts’ [38] collective mind theory to account for the enactment of...
co-ordinated activity in a way which contributes to project success. Shared understanding between participants in a software project is critical e.g. [28]; collective mind theory articulates the mechanisms by which a shared understanding can be achieved and has been used for this purpose in previous software engineering research [11]. We propose that collectively, these concepts provide a framework through which we can investigate, describe and analyse co-ordinated activity in software engineering projects.

2.2.1. Co-ordination theory

The literature on organizational theory in general, and computer supported co-operative work (CSCW) in particular, provides a number of approaches to the study of co-ordination and co-operation in complex work settings. For example, Mintzberg identifies five mechanisms fundamental to the co-ordination of work in organizations, namely mutual adjustment, direct supervision, standardization of work processes, standardization of work outputs, and standardization of worker skills [27, pp 4–9]. Malone and Crowston’s [23] notion of co-ordination theory has emerged from a survey of co-ordination processes, not just in organizations, but across a range of complex systems such as economic, biological and computer systems.

Malone and Crowston define co-ordination as ‘managing dependencies between activities’. From this perspective, they present a co-ordination framework or ‘theory’ which characterizes different kinds of dependency between activities and identifies co-ordination mechanisms that can manage these dependencies. Refer to Table 1 for examples of co-ordination dependencies and mechanisms and Fig. 1 for key co-ordination theory concepts. Through the generalization and specialization of co-ordination dependencies and their mechanisms, co-ordination theory seeks to enable comparison of co-ordination strategies in disparate situations and processes, and provide a set of building blocks for developing new organizational processes [24].

2.2.2. Communication genres

Our approach to examining software processes requires more than an ability to identify different types of co-ordination mechanism at work. It is also necessary to describe specific features of communicative activity within a project team. Techniques such as Schmidt and Simone’s Co-ordination Protocol [32] are useful where the activity is to be specified with a view to automated support by a CSCW system. However, for our purposes, a less formal mechanism is sufficient, such as Yates and Orlokowski’s notion of communication genres [40].

Genres of organizational communication are socially recognized forms of communicative actions - such as memos, meetings, ballots, design reviews, bug descriptions - that are habitually enacted by members of a community to realize particular social purposes. A genre established within a particular community serves as an institutionalized template for social action. A communicative genre is characterized by a socially recognized communicative purpose and common aspects of form. Genres may be considered at different levels of abstraction; for example, the formal review process may be defined for defect detection, but a requirements review may be specialised to the needs of a software sub-contractor. Genres may be linked or networked into genre systems.

<table>
<thead>
<tr>
<th>Dependency</th>
<th>Possible Co-ordination mechanisms for managing dependency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shared resources</td>
<td>‘First come/first serve’, priority order, budgets, managerial decision, market-like bidding (e.g. managed through configuration management tools; object public methods/properties constructs)</td>
</tr>
<tr>
<td>Producer/consumer relationships</td>
<td>Notification, sequencing, tracking (e.g. managed through application of standard change management procedures)</td>
</tr>
<tr>
<td>Prerequisite/consumer relationships</td>
<td>Inventory management (Just-in-time, economic order quantity), task allocation (e.g. managed through automated or manual task allocation system)</td>
</tr>
<tr>
<td>Accessibility (right place dependencies)</td>
<td>Standardization, participatory design (e.g. managed through provision and automated enforcement of standard template for change requests)</td>
</tr>
<tr>
<td>Fit constraints</td>
<td>Scheduling, synchronisation (e.g. managed through standardized build processes and software configuration management)</td>
</tr>
<tr>
<td>Task/Subtask dependencies (e.g. the breakdown of project tasks into subtasks for management of complexity or task allocation purposes)</td>
<td>Goal selection, task decomposition (e.g. work breakdown structures, PERT/CPM planning techniques)</td>
</tr>
</tbody>
</table>
constituting a more co-ordinated communicative process. Communication genres are particularly relevant to our work as they provide a means of describing communication processes in a software project in a way that recognizes the inherently situated and dynamic nature of these processes [29].

2.2.3. Collective mind theory

Preliminary field investigations revealed co-ordination practices in which group participants ‘just knew’ what actions to take through the application of tacit knowledge fostered through interaction with co-located team members. Weick and Roberts’ theory of collective mind [38] provides a basis for describing this activity. Similar issues are addressed in Clark’s notion of common ground [10] between participants in a joint activity, a concept which has proved useful for explaining interaction and shared understanding between computer system users and developers [28].

The concept of collective mind was originally developed to explain organizational performance in situations requiring nearly continuous operational reliability but has since been applied to more conventional organizational processes, including software engineering [11,22]. Collective mind can be considered as a pattern of heedful interrelations of actions in a social system (Fig. 2). Participants in the system construct their actions (contributions) in the understanding that the system consists of connected actions by themselves and others (representation), and interrelate their actions within the system against this background (subordination). The term ‘heedful interaction’ is used to describe the relating of one’s work processes to others’ work process carefully, critically, consistently, purposefully, attentively, vigilantly, conscientiously, and so on. Ongoing variation in the heed with which individual contributions, representations, and subordinations are interrelated influences comprehension of unfolding events and the incidence of errors in the process. As heedful interrelating and mindful comprehension increase, organizational errors decrease. The quality of collective mind may be dependent on a number of social processes, for example, how knowledge of previous experiences is perpetuated within the group, or how newcomers to the group are induced and integrated. It is useful to contrast heedful performance of a task with habitual performance:

In habitual action, each performance is a replica of its predecessor, whereas in heedful performance each action is modified by its predecessor… heedful performance is the outcome of training and experience that weave together thinking, feeling and willing. Habitual performance is the outcome of drill and repetition [38, p362].

Fig. 1. Concepts in co-ordination theory.

Fig. 2. Aspects of collective mind.
2.2.4. A framework for describing and analyzing co-ordination in software engineering projects

As we have argued in Section 2.1, existing process models do not adequately match the reality of the situated action and ways of working of practising software engineers. While previous studies have investigated such situated practice and drawn observations from it [6,37], such situated practice is difficult to describe and abstract with a view to its improvement and transfer of best practice. We have identified co-ordination theory, the concept of communicative genres and collective mind theory as devices for describing and analyzing such activity (Fig. 3). Our thesis is that co-ordinated activity, as defined by co-ordination theory, is managed through co-ordination mechanisms typically operationalized as communicative genres, and managed effectively through the heedful operation of these genres. Collectively these concepts can describe co-ordination practices in a software project. By devising means for describing and analyzing co-ordinated activity in this way, our ultimate aim is that existing software process models can be augmented with guidelines for improved communication and co-ordination.

3. Field study

3.1. Organizational context

We studied co-ordinated activity in two software engineering projects. Both projects were geographically distributed projects, requiring extensive co-ordination across sites as well as within co-located teams. The organizations, individuals and products are referred to by fictitious names to maintain the anonymity of the organizations concerned.

3.1.1. Alpha.uk—the CDB project

Alpha.uk was a UK-based software organization developing a range of information systems for a North American partner company. One of their projects (hereafter referred to as the CDB project) concerned the re-engineering and enhancement of a sophisticated customer database, enabling the North American company to construct detailed profiles of its commercial customers for business analysis purposes. Key features of the project were as follows:

- The CDB system was three-tier client-server in nature, consisting of a Win32 client built using C++ and a Unix-based relational database server.
- Almost all software development was conducted by the UK project team, consisting of a project manager (Tom), a senior software engineer (Paula), three software engineers (SEs) and four graduate software engineers (GSEs) (see Fig. 4). SEs typically had two or more years software experience, GSEs typically one to two years experience.
- The project’s business manager, users, technical manager and database administrator were based at the North American site.
- All nine UK team members worked full-time on the project, though Tom was also managing one other project throughout most of the CDB project.
The project was developed in three releases over an eighteen month period. We had access to the project during the final ten weeks of release three.

3.1.2. Beta.uk—the MMT project
Beta.uk was also a UK-based software organization developing commercial software systems in conjunction with partner companies in the UK and North America. These software systems were mainly embedded systems for use in the deployment and management of large telecommunications networks. One aspect of Beta.uk’s work was to progressively take responsibility for the ongoing development of an existing product for the monitoring and management of a telecommunications network (hereafter referred to as the MMT project) - a product which had previously been the responsibility of a partner company in North America (Beta.na1). Key features of the MMT project were as follows:

- The MMT system included Java and CORBA technologies, with product components running in both embedded-mode systems and on Unix-based servers incorporating relational database technology.
- The project did not have clearly defined beginning and end points. It was an ongoing programme of work to develop and support the MMT product through a series of releases, with a new version release typically every six months.
- The UK development effort was organized as a group consisting of six teams (see Fig. 5). The teams concentrated on different aspects of the MMT technology.
- Overall management responsibility for the MMT project resided with one of the North American partner sites (Beta.na1). Other North American sites (Beta.na2 and Beta.na3) were also involved in the MMT programme, having input at a planning and technical level.
- For this study we examined one of the UK-based teams, comprising a principal software engineer (John), four senior software engineers (SSEs), and two software engineers (SEs). SSEs typically had eight to ten years experience in software development, with SEs having three to five years. In addition we interviewed another principal software engineer (Al) who was not directly attached to any of the six teams. He had a distinct role within the group as an ‘Advisor to all’. All team members worked full-time on the MMT project.
- At the time of our study, the project was in its infancy with the UK team only in existence for approximately four months, and the overall MMT UK group growing rapidly through recruitment, to a planned size of approximately forty engineers.

3.2. Research method
Through a qualitative and interpretative approach, our aim in the fieldwork was to collect rich data, from multiple perspectives, regarding the operation of the two projects.

3.2.1. Data collection
Our primary data collection method was interview, supplemented by review of relevant project documentation, on-site demonstrations of software, and informal conversation. In both cases interviews were conducted by two people.
on company premises using either a meeting room next to the project work area or at an engineer’s desk.

The concepts of co-ordination theory, communication genres and collective mind theory informed the structure of the interviews. Questions were asked under five headings: personal profile questions, team structure and communication mechanisms, workflows and task dependencies, communication patterns with respect to above dependencies, and examples of co-ordination in context. (See Appendix A for the interview questions). These headings, with sample questions related to their particular software project, were circulated in advance to the interviewees. In both cases, from our preliminary meetings with the project manager, we had produced a project structure diagram for use as a reference point in our interviews.

3.2.2. Data analysis
A qualitative content analysis technique [8,26] was used to extract key co-ordination themes from the interview data. We regarded as key co-ordination themes topics which were mentioned by two or more interviewees, or which recurred during the course of an interview, or which we considered significant from a co-ordination perspective. We then re-examined our data to identify co-ordination mechanisms used in the project, and the co-ordination dependencies they were used to manage. The third pass through the data was concerned with identifying the communication genres used in the project. Of course, this process of data analysis was highly iterative.

In doing our data analysis, our aim was to identify, describe and make sense of the co-ordination issues and mechanisms at work in each project. To improve the quality of our interpretation, we repeated back our initial findings to each organization, particularly seeking their comments on accuracy. In both cases, this served to rectify some omissions and misunderstandings in our analysis.

3.2.3. Limitations of approach
Arising from the research method used and inherent features of the two cases studied, it is important to note the limitations of our approach. Our method of investigation was qualitative and interpretive. While providing rich data, such a method is subject to bias. However, within our research method, a number of the measures outlined above were introduced to reduce this bias, such as: the construction of an analytical framework based on established theories to guide the interview questions and subsequent data analysis; the conduct of interviews in pairs with comparison and cross-referencing of interviewer notes; and providing post-interview feedback to the host organizations for error checking. The inherent features of the cases themselves are potentially limiting. They are not controlled, experimental settings, therefore it is not possible to isolate and objectively describe particular co-ordination practices. Furthermore, the two cases represent early-project and late-project co-ordination practices and potentially miss important mid-project issues.

Nevertheless, it is our view that the merits of a rich, qualitative analysis outweigh the disadvantages arising from such an interpretive approach. Our method gives insight into important project processes and our analytical framework provides a basis for generalizing findings from these cases for the benefit of other software engineering projects.

4. Empirical observations
In this section we present the main observations from our fieldwork. For each case we first give an overview of the day-to-day activities on the project. We then report on the key co-ordination practices to emerge from the field data in terms of the communication mechanisms at work, communication genres in operation and, where appropriate, collective mind issues. Our purpose here is to illustrate the type of issues uncovered through our analytical approach. Implications for software engineering practice are discussed in Section 5, addressing the extent to which our description of co-ordination practices can fill the gap between software process models and situated practice.

4.1. The CDB project at Alpha.uk

4.1.1. An overview of CDB project activity
The company’s own software process was used to structure the CDB project. Senior users developed functional requirements for the system and supplied these to the UK team. Functional requirements centred around the screen designs and associated business rules for the major application features. Each screen, its underlying functionality and database procedures were developed, tested and integrated into the system by the UK team. It was normal practice for functional specifications to require clarification and modification. This was achieved through controlled communication between the two project teams, with Paula’s role designated as having the main responsibility for this communication. Indeed, it was Paula’s role and the related separation of responsibilities across roles in the UK team which emerged as a significant co-ordination strategy in the project.

From a co-ordination theory perspective, and at a high level of abstraction, the project can be characterized by two types of dependency at work: a conventional producer/customer (flow) dependency between the work processes of the North American team and the UK team, and a task/sub-task hierarchy of dependencies in the UK team, as represented by the project plan and the team’s overall goal of delivering the project on time.

The project plan also served as a principal co-ordination mechanism at the project level. It was a shared document
between the UK and North American teams, and internally between members of the UK team. A genre system was clearly evident in the updating of this document, involving the informal progress reporting from the UK team members to Paula, a weekly face-to-face progress meeting between Paula and Tom, and communication between Tom and the project business manager via email, telephone, or videoconference.

4.1.2. Separation of responsibilities across project roles

A significant co-ordination feature we observed was the separation of responsibilities across various roles in the UK project team. Tom had overall project management responsibility for the project. While managing this project, he was also managing another project. Paula’s role could be described as that of ‘feature management’. Amongst other things, she was the main point of contact in the UK team for the senior users. The remaining software engineers fulfilled roles which were almost exclusively technical, involving coding, testing, and integration of application functionality.

Paula’s role was central to the project. She clearly understood her role to be one of ‘controlling communication’ between the user and development groups. The other members of the project also clearly understood this to be the case. When a UK team member had a query for the senior users, it would initially be directed via Paula. Only if she were not available would they make direct contact with the users. Similarly, North American team members would invariably direct their communications to Paula, and rarely to other members of the UK team.

Paula’s role was a primary co-ordination mechanism for managing the critical dependencies between the two teams. In terms of ensuring that information flow was the right information to the right people at the right time (usability, accessibility, prerequisite dependencies), a number of subordinate mechanisms were employed, such as prioritising queries, filtering queries and synchronization of communications.

Paula’s role was seen as liberating by other members of the team. As noted by one engineer, Paula took all of the administrative duties or ‘distractions’ away from the developers, allowing them to concentrate on development. She also ‘handles all of the heat’, referring to her role in dealing with crises on the project, insulating the engineers from such matters.

Coupled with this, Paula managed day-to-day task allocations on the project. While the ‘big picture’ was set out in the project plan, Paula would control this at the day-to-day level. For example, during testing and bug fixing, Paula would distribute bug descriptions to appropriate team members. Also, developers would try not to work on parallel versions of the same code as the configuration management tool did not handle this particularly well. Where this had to happen, they would liaise with Paula as she knew the problems associated with each application screen and the related code base.

Though formally a senior software engineer, Paula’s role was summed up by Tom as ‘more about communicating than technical details. She is a repository of business knowledge’. Paula’s role was the locus for many of the communication genres identified in the project - her role represented a central co-ordination mechanism in which many project communication genres came together.

4.1.3. ‘Keeping people in the loop’

Within the project, the practice of copying or ‘cc-ing’ email communications to team members was referred to as ‘keeping people in the loop’. To a greater or lesser extent, this feature was evident from all the interviews. It can simultaneously be considered as a co-ordination mechanism and an institutionalised communication genre. For example, when Paula would email the North American database administrator, requesting a database design change, the email would be copied to the UK team. On those occasions when UK team members would be in email contact with senior users, Paula would always be copied any communications. For some issues, the whole team would be copied (as with the database change), sometimes Paula and Tom, sometimes a programming partner (see below) or other interested individual within the team.

4.1.4. Programmer pairing

Programmer pairing was a phenomenon associated with the technical members of the UK team. The concept of programmer pairing had been introduced by the project manager to strengthen project teamwork, though specific pairings developed informally according to the requirements under development or an individual’s technical specialisms and previous experience. For example, some pairing was to compensate for different levels of experience, often to assist newcomers in the team; less experienced members of the team would work with experienced members for screen design and coding duties through all layers of the three-tier architecture. On other occasions, pairing was on the basis of complementary skills; for example, SE2 was enthusiastic about the work dynamic between him and two other developers on the team, as he had experience in database procedures, another had experience in C++, and the other had expertise in Microsoft foundation classes. One of these developers had brought his experience from a separate project in the company. SE2 described how they worked well together, with lots of cross-fertilization of technical knowledge. They were also ‘good mates, and continually exchanging conversation, ideas and solutions’.

The pairing mechanism did not constrain wider team collaboration. GSE3 commented that if his partner was not available, he could turn to others for help. The team members seemed instinctively aware of each other’s strengths and weaknesses and who to turn to for specific help. There was a strong sense of teamwork on the project.
As resource requirements within the UK organization fluctuated, engineers could be transferred from one team to another. This was considered by the company to have a positive impact on the CDB project, enabling skills and experiences from other projects to be applied.

4.1.5. Task allocation
The way in which work was allocated to team members had co-ordination implications for the CDB project. For its first release, tasks were typically allocated on a per level basis in the three-tier architecture (see Fig. 6). For releases two and three, this was done on a screen-oriented basis, cutting through the three layers of the three-tier model. Engineers reported this to be a more efficient means of task allocation. It can be seen as an effective co-ordination mechanism for managing the task/sub-task dependencies in the project, allocating them in such a way as to minimize coupling between each engineer’s work.

Primary task allocations were defined in the project plan. These were refined and managed by Paula in response to issues arising between the UK and North American teams. With Paula responsible for receiving progress reports from the team members, she adjusted task allocations to maintain progress on the project plan. For bug fixing, Paula also controlled the task allocation process. She would allocate bug descriptions to individuals through the bug tracking system, receive from them estimates on time to complete, and update the bug list as bugs were corrected and fixes implemented. Overall, task allocation was co-ordinated by Paula who, by virtue of her role, was able to respond to the local needs of the project team and the requests of the senior users.

4.1.6. Other issues
A number of other co-ordination practices emerged from our study at Alpha.uk and are outlined here for completeness.

(i) The project’s software configuration management (SCM) tool played a central role in achieving effective technical co-ordination. With multiple developers working on the same code base, the tool helped maintain the integrity of the project code through well established configuration management principles.

(ii) Formal team meetings played a useful co-ordination role, though they were relatively infrequent on the CDB project. Team meetings were typically held prior to a major build of the software, and were used to highlight and discuss technical issues requiring resolution before the build could proceed.

(iii) It was a common occurrence for team members to make technical queries to co-present colleagues in the UK team or to project members in North America. In the former case, this was often through face-to-face communications; on some occasions email was used. When querying remote project members, email or telephone was used. Querying was more than an ad hoc enquiry mechanism; it had the characteristics of a communication genre, being enacted in response to a recurrent situation (i.e. the need to resolve a project uncertainty), with a socially recognizable communicative purpose and form (i.e. to obtain a quick and informal response to a project issue through free-form email or a telephone conversation). The role of queries will be discussed more fully in the Beta.uk case, where they played a critical role in achieving effective co-ordination.

4.2. The MMT project at Beta.uk

4.2.1. An overview of MMT project activity
The MMT project required Beta.uk to set up a new organizational group with a planned total size of approximately forty software engineers. This group was divided into six teams, reflecting the various aspects of the MMT product. The team to which we had access (referred to here as John’s team) had been allocated various responsibilities, primarily ensuring backward compatibility of the MMT product with older network technologies. At the time of our study, the group was in its early stages of development, with many of the engineers new to the Beta.uk organization. The UK teams were ‘finding their way around’ both the MMT product and the organization of the North American teams. The task allocation mechanisms within the project reflected this ‘start-up mode’. At a high-level, Sam (the UK group leader), liaised with his North American counterparts to agree major work areas. These would be allocated to the appropriate UK team, typically in the form of so called PTRs (problem tracking reports). Often these work areas would be specified in general terms, with team leaders such as John refining the requirements with their North American counterparts. On occasions, Al would be involved at this stage of requirements clarification. Ultimately the agreed work areas would be assigned to members of John’s team.

At this early stage of the project, much of the work allocated to the UK group was designed to ease them into
the features and functionality of the MMT product. Consequently, some of the work was investigative in nature, or was more concerned with bug fixing rather than feeding directly into the MMT release plan. This had the effect of adding a layer of complexity to the co-ordination dependencies between the UK and North American teams. For example, whereas work might have been allocated as described above via Sam, Al and John, the communication channels which opened during the execution of the work often by-passed these people. A typical pattern was that SSE1 established links with a couple of software engineers in one of the North American teams, agreed detailed requirements and interim milestones between themselves, and exchanged interim and final software products, test plans and technical know-how. This pattern was replicated between other members of the MMT project, both within the UK site and between the UK and North American sites. These producer/consumer co-ordination dependencies where further complicated by daily queries and information flows - with the project being in start-up mode, numerous queries were made by UK team members to engineers in North America and to other engineers in the UK group.

During our period of investigation, the net effect of these numerous co-ordination dependencies was a highly complex and dynamic set of project work processes. Though the engineers working for John were constituted as a team, holding responsibilities in particular areas of the MMT product, the most cohesive units of engineers actually working together often consisted of UK software engineers collaborating over a period of weeks or months with engineers from one or more of the North American teams. Key co-ordination dependencies were inter-team, across geographically distributed sites, rather than intra-team within the UK group.

4.2.2. The knowledge network

When the UK project group was established, it was staffed by three existing Beta.uk engineers. Their role was to investigate the issues involved in bringing MMT work to the UK site and to begin some of the technical development. This involved travel to the main North American site for approximately six weeks. As the UK development effort began to grow, Sam sought to maximize the usefulness of these project pioneers by their strategic placement within the UK group as nodes in an evolving knowledge network. In particular, rather than be allocated to a team, Al was given the role of ‘advisor to all’. Al was a central figure in the setting up of the UK group - he had liaised significantly with MMT engineers in North America, and consequently knew the roles and responsibilities of key members in the project (‘contact’ know-how) and had acquired a sound knowledge of the MMT product architecture and North American development processes (‘technical’ know-how). During our period of investigation, approximately 70–80% of his time was spent in reactive mode, responding to queries from all sections of the UK group. One of the other original software engineers (SE1) was allocated to John’s team. Though given specific technical work to complete, he was also available to resolve queries. Initially this accounted for 70% of his time, though during our period of study this had reduced to about 10%.

From a co-ordination theory perspective, the knowledge network was supporting three main dependencies. An important responsibility of these original engineers was to ensure that the work done by the UK teams was consistent with existing product features and future requirements. In other words, part of their role was ensuring the usability of the UK deliverables (i.e. that they were producing the ‘right thing’). A second aspect was helping UK team members resolve queries by guiding them to the right people at the lead North American site, facilitating the transfer (right-place dependency) of information between geographically dispersed members of the MMT project. A third aspect of the knowledge network was to facilitate the sharing of critical knowledge within the UK group. This was achieved through their strategic distribution throughout the organization structure, their flexible role description and also the suitability of their inter-personal skills to this communication intensive role.

4.2.3. Queries

The need to make queries to the North American sites was a routine, almost daily, activity. In large part, this was due to the start-up nature of the group and the fact that the engineers were new to the Beta.uk organization in general and the MMT project in particular.

Though a query may be viewed as an essentially unstructured and ad hoc problem solving mechanism, our analysis suggested that it could be characterized as a communication genre. Where it was necessary to resolve a technical issue of any description, the query mechanism was employed, i.e. it had a socially recognized purpose. A query mainly took one of two forms. Free form email was one method, often consisting of a preamble to lead the recipient into the context of the query. The detail of the preamble was judged depending on the recipient’s role in the project and familiarity with the problem. Such emails were copied by the sender to those he thought should be informed of the issue. There was no policy as to who should be emailed - this was left to the judgement of the sender. Alternatively, the query could be made by telephone, with a verbal preamble to set the scene. In many cases a telephone query was not a stand-alone mechanism but rather a follow-on to an email query. A recurring problem with the query phenomenon was knowing to whom it should be directed. Identifying the recipient constituted an essential element of the query; if the engineer knew the best person to ask, this went a long way to ensuring the successful resolution of the query. In the start-up stages of the MMT UK group, much effort could be expended in establishing contact with the right person. The knowledge network played a key role in facilitating this ‘searching’ aspect of a query.
Queries supported two main co-ordination dependencies. Primarily, they were associated with usability (right thing) dependencies, i.e., ensuring that the work to be done was based on correct facts and shared assumptions. The query mechanism also helped manage pre-requisite (right time) dependencies by ensuring that other North American sites had completed pre-requisite activities before the UK team could progress some of their work (such as upgrades to related software modules). When an engineer considered the solution of a query to be of benefit to others in the UK group, he was encouraged to submit it to an Intranet FAQ (Frequently Asked Questions) facility which was being developed for the MMT project. The FAQ resource thus had the potential to become a co-ordination mechanism to complement the query mechanism.

4.2.4. Problem tracking reports

Work was allocated to the UK group mainly in the form of problem tracking reports (PTRs). Their purpose was broader than their name implies - PTRs were used to formally record and manage any product requirement (such as new product features, enhancements, changes and bug fixes). In effect they were a management device for task allocation and control. As such, they served as a common language and framework for work. For example, once a PTR was issued it would progress through a series of predefined states (such as assigned, coded, answered, verified) until its completion. PTRs were issued through an automated tracking system which, on entering or exiting certain states, would send notification emails to relevant project staff.

From an organizational and technological perspective, PTRs were a unifying co-ordination mechanism in that they addressed various key dependencies between activities in the MMT project. In terms of relating tasks to sub-tasks, they were a standardized mechanism for the allocation and management of work. With respect to the elementary producer/consumer cycle operating between the UK teams and the North American sites, PTRs addressed key dependencies. In relation to right-thing dependencies, the usability of delivered products was addressed through the quality checks associated with key PTR states. In relation to right-time dependencies within the development process, PTRs standardized the work cycle for project activities. In terms of right-place dependencies, PTRs provided for automatic notification of when key work states were completed. The pre-defined state model for PTRs ensured right-time dependencies were addressed by defining the pre-requisite constraints which work should follow. The PTR process also provided co-ordination at a higher level through a PTR Board. Based at the main North American site, this board considered and prioritized future work for the MMT project, serving as a co-ordination mechanism for allocating tasks to the project engineers.

There was also an important communication perspective to the PTR process. The PTR mechanism was a clear communication genre within the MMT UK group. It had a socially recognized purpose (to formally record and manage a product requirement) and form (in terms of the predefined properties and states of a PTR, supported by the PTR database, a formalized system of state transitions, PTR analysis reports, and notification emails). In the relatively short time period since the start-up of the UK MMT group, the PTR mechanism was an established genre within the group. At this stage in the group’s life, no standard process model had been introduced. To the extent that a process was in place, it was the operation of the PTR mechanism as a communication genre - a widely understood, efficient and effective co-ordination mechanism suited to the purposes of the group at that point in time.

4.2.5. Software tools

Software tools played an important role in the operations of the MMT project in its early stages, and the intention of John and other team leaders was that the role of such technologies would significantly increase as the project became established.

The MMT project was a distributed software engineering project. Robust SCM tools were critical to the management and control of the code base. The same SCM tools were used across all MMT project sites, with Beta.na1 having overall responsibility for build management and the controlled release of development code bases. Use of the SCM tool was implicit in every engineer’s interaction with the code. A feature of the SCM software allowed engineers to record a comment or description with each change they made to the code base. Team members were informally encouraged to put a short and meaningful comment with the upload, and used their own judgement to determine the level of detail to include.

SCM tools provided a standard process for all members of the MMT project when working with the code base. Features of the tool for change management were the driving force behind the process of PTRs mentioned above. Much of the PTR process standardization was derived from the state model for changes provided by the SCM software. Software development such as the MMT project would not be possible without sophisticated SCM tools. They are a core co-ordination technology, addressing every major type of co-ordination dependency related to code sharing by multiple developers.

A common programming technology (Java) and development environment (Unix-based) was in use across the entire MMT project. This in itself was an important co-ordination mechanism as it provided tool standardization for every engineer. However, there were no project wide coding standards in place. Each engineer was encouraged to use JavaDoc [35] as a standard documentation style and template for all programs. Al was given responsibility for investigating the role of coding standards. He commented that ‘a fine balance has to be maintained between enforcing a standard and letting programmers do their own thing’. SSE1 described JavaDoc as being invaluable.
for understanding other people’s code, though the extent to which code had been annotated to maximize the potential of JavaDoc varied. Engineers working on specific PTRs found JavaDoc to be useful in understanding other people’s code, though at the project level there was no overall architectural design for the MMT product, and this was a frustrating factor for the UK group. Standardization of technology at a detailed level facilitated multi-party working on shared code resources, but lack of an architectural design, as a shared knowledge resource, was a limiting factor at the project level.

A key task in which John was engaged was the establishment of a ‘team database’. This was an Intranet facility for recording and disseminating a range of key facts about the MMT project group, mainly for the benefit of the UK personnel. Features included a browsable and searchable people contact database, with results usable as a dynamic email distribution list, a detailed calendar of tasks for each engineer (tasks, holidays, training), an action point list for each engineer, a risks and issues list for each engineer, plus an equipment requisitioning system. John regarded this system as a critical application for the UK group, especially in its early stages; for example, it was used by new engineers to quickly get acquainted with the profiles of other team members and their work areas. The risks and issues list was an important shared space for recording technical issues to be addressed.

John also initiated an MMT project FAQ on the Beta.uk Intranet. Initially this was organized into three areas - an FAQ for new engineers, one for setting up the desktop computing environment, and a third for MMT project specific information. The FAQ was used by everybody on the team for finding solutions and also for formally recording solutions they had uncovered. Individuals or a group of individuals from the team would informally decide if something should go into the FAQ. As the MMT-specific FAQ grew, it had become necessary to organize it into specific sections and provide a search facility. There was also some consideration being given about how to more formally manage the FAQ content. The FAQ can be seen as a co-ordination mechanism for managing shared knowledge dependencies in the project. There was a general understanding that once a technical issue had been resolved, its solution, or a generic version thereof, should be made available via the FAQ. This was complementary to the knowledge network and over time it was hoped it would reduce the volume of queries by being a substantial repository of project technical knowledge.

4.2.6. Other issues

Other co-ordination practices were observed at Beta.uk and, as with the previous case, are outlined here for completeness.

(i) Work in progress was monitored through the completion of weekly reports by each engineer. This took the form of a free-form email, submitted on a Friday afternoon, summarizing the week’s work. The content and level of detail in the email was left to the judgement of each engineer. John then collated and summarized these reports and forwarded them to Sam, who used them in submitting a weekly report to Beta.na1.

(ii) SE1 was enthusiastic about the use of whiteboards as a shared work space. As one of the original project engineers, he was often asked by colleagues to explain features of the MMT product. He used the whiteboard to illustrate and record many of these explanations. By leaving the information on the whiteboard for a period of time, others in the team could refer to what was being discussed.

(iii) During our period of investigation, John had just introduced an ‘instant message’ system using the group’s Intranet. This was to support the exchange of brief, on-line messages, announcing events such as ad hoc team meetings, network downtimes, or social events.

5. Discussion

In this section we reflect on the role of our framework in identifying communication and co-ordination practices critical to effective software engineering, yet not part of mainstream software process models nor the body of best practice recognized in the software engineering community [19]. Our purpose is to illustrate the types of activity that have come to light through our analysis and to suggest implications for software project planning and practice.

5.1. Co-ordination mechanisms—essential project practices

Sommerville and Rodden [34] state that informal activities, such as expert consultation, are critical to the success of a software project. We have used our analytical framework to uncover a comprehensive set of unspecified and tacit work activities which are critical to the effective co-ordination and operation of a software project. We describe these co-ordination mechanisms as unspecified in that they are typically not defined in standard software engineering process models and tacit in that they are part of the situated, day-to-day problem solving strategies that software engineers employ. Though the mechanisms reported relate to the specific projects studied, we suggest they are illustrative of essential co-ordination practices in successful software projects.

Keeping people in the loop through the copying of emails is an example of one such co-ordination mechanism. In both cases studied, there were no formal rules for who should be copied any given communication. We observed that engineers just knew who to include. And yet keeping people in the loop was a critical and pervasive co-ordination...
mechanism in the project. As one of the software engineers at Alpha.uk commented, copying emails was regarded as ‘more than just being polite, it was something you could get your knuckles rapped for if not implemented’. It served as a mechanism which bound the team together, giving it cohesion, yet enabling the required degree of loose coupling between the work of individual project members.

Another example of an unspecified and tacit co-ordination mechanism was the ‘query mechanism’ observed in both projects. In both cases, queries were a primary information gathering and problem solving technique. Engineers had devised various strategies for making queries as effective as possible, such as evaluating the contextual knowledge of the recipient of the query, judging what level of detail to provide in the query preamble, and identifying alternative recipients and communication mediums for the query. In Beta.uk in particular, the query was a critical co-ordination mechanism for managing the information flow in the start-up mode of the MMT project, and for helping new team members orient themselves within the project. Such activities do not fall within the scope of software process models, and software process models would not claim to address such activities. While the important role of communication skills in software engineering is widely recognized and documented, our analysis points to the way in which these communication activities actually maintain the coherence of project activity - they are the glue which holds the project together. As we will discuss later, understanding this activity in terms of communication genres enables the analysis and improvement of these communication processes.

Not all the co-ordination mechanisms we observed were unspecified or tacit. Some were an explicit feature of the project structure. For example, SCM tools were essential for the co-ordination of multiple engineers working on a common code base. The associated co-ordination mechanisms were explicitly encoded in the configuration management procedures supported by the tool. The tool was used in the context of standardized procedures, enforced by the technology, supporting co-ordination through standardization of low-level work processes. Other examples of explicit co-ordination mechanisms were the weekly reporting mechanisms in both organizations, the risks and issues mechanism introduced by John in Beta.uk, and the knowledge network introduced by Sam in Beta.uk. The knowledge network set up in the MMT project is consistent with recommendations made elsewhere in the context of global software development [9,17].

The distinction between explicit and tacit co-ordination mechanisms is significant in relation to the effectiveness of work practice. Effective work is not so much determined by process engineering and rigorous project planning but by the capacity of people to solve unanticipated and complex problems in complicated situations [31]. Accommodating human-factors in project work requires a balance between explicit (organizational) elements in work and tacit (activity-oriented) elements. There is a necessary but limited amount to be achieved by putting in place explicit co-ordination mechanisms. Our analysis would support the view that it is often tacit co-ordination practices which make a project succeed and they should not be regarded as unprofessional or unstructured. Software project managers should recognize this, seeking to maintain a balance between explicit project planning and facilitating tacit co-ordination strategies.

5.2. Communication genres—process in action

Communication genres enable us to identify and study co-ordination mechanisms from the perspective of the communicative practices which engineers use in their management of co-ordination dependencies. In both organizations, certain genres emerged as central to the operation of the project. We have already discussed the role of the query and of copying emails. The query was a socially recognized way of solving a technical problem. Some of the activity we identified is best described as a genre system. Genres or genre systems which were enacted in the projects could be regarded as the situated processes which made the project work - process in action. Whereas a software process model defines a top down template of what processes should be enacted, genres represent a bottom-up emergence of day-to-day project processes. For example, the PTR genre was a process which was understood across the MMT project for how to formally record and manage a product requirement. It was not a feature of an organization, group, or project wide process definition, but rather emerged at an early stage as a way of working, based on the configuration management technology used by the MMT project engineers. In Alpha.uk a similar observation was made in relation to bug fixing, as managed by Paula.

This has implications for software project planning and practice. If we accept genres as reflecting and being consistent with the situated practice of software engineering, then these genres and genre systems are a basis for identifying lightweight, bottom-up processes which reflect the reality of project work. The genre mechanism offers a means of describing the features of these processes and planning for their future evolution and improvement, complementing and augmenting existing standard processes. It recognizes that the most effective and legitimate processes may not be those prescribed by an organizational process model or project plan. The underlying feature of communication genres as something which evolve over space and time in an organization is consistent with the software engineering notion of process improvement. We believe there are lessons here for organizations introducing new software processes - that they should be aware of, and where possible accommodate, the existing and effective co-ordination practices at the project level. Communication genres, such as queries, are not just background noise in a project, but are the tacit co-ordination mechanisms which
make the project work, and their heedful enactment plays an important role in achieving project success.

5.3. Heedful interaction—maintaining team awareness

We have already discussed the contribution of tacit co-ordination mechanisms and communication genres to project success. However, the application of these practices does not necessarily imply they will lead to project success - what matters is whether these practices are conducted heedfully. As collective mind theory suggests, where team members maintain an awareness of each other's roles through heedful interaction, the greater the reliability of team performance.

An observation we frequently made was that many processes were not written down. Amongst project engineers at both organizations, there was a heavy reliance on 'just knowing' how certain tasks should be done. At Beta.uk for example, this was evident in the form and detail of weekly reports, determining who to contact when raising a query, determining when to copy and when not to copy emails, and knowing what level of comment to put in to a code submission to the configuration management tool. A collective mind analysis suggests that reliable enactment of unspecified processes derives from the heedful interaction of team members as each relates his or her work processes in a careful and critical fashion to the work of others. What we observed was confirmation of Weick and Roberts' statement that 'heedful interrelating foreshadows heedful contribution'. An engineer's understanding of where and how he fitted in to the project and wider organization preceded doing a good job. Engineers in both projects oriented themselves within the project, evolving their understanding of their role in the project and the project's role in the wider organization.

The notion of heedful interaction provides an interesting analysis on the MMT project's move towards greater automation of some key co-ordination mechanisms. Weick and Roberts state that ongoing team reliability is associated with ongoing heedful performance of tasks, not habitual performance. Interestingly, activities which contribute to collective mind are apparent in the early start-up stages of a team, arising from the need to orient oneself within the team, clarify facts, and test assumptions. Consequently, collective mind may be strongest in the early stages of a team's existence. As the team matures, there is a tendency for habitual performance to become more dominant, unless conscious steps are taken to maintain strong collective mind. We observed that the MMT project aspired to fully exploit Intranet technology for group work applications such as project plans, project risks and issues, FAQs, and design review automation. To the extent that automation is associated with habitual performance of tasks, collective mind theory perspective would caution against over-reliance on technology. For example, if engineers tended to find the answers to their queries on the Intranet rather than through personal interaction, then aspects of heedful interrelating might not be enacted, leading to a deterioration in collective mind. This is supported by other work in human factors which demonstrates the need to maintain human co-operation in work design in order to achieve effective and reliable system performance [30]. While a project manager might aim for quality and efficiency gains through increased automation of the development process (for example using CASE tools), such a strategy may have the effect of inhibiting the software engineers' shared understanding of the project environment - an understanding which ultimately leads to improved project performance.

Other questions arise when looking at the cases from a collective mind perspective; for example, are particular genres associated with strong collective mind or weak collective mind? Does heedful interrelating determine the types of genre used and how they are used? These are matters for further study.

6. Conclusions and future work

Our purpose in this paper has been to identify and articulate some of the co-ordination practices that contribute to software project success. The analytical framework devised for this purpose provides a focus for data collection when studying real-world software projects, enabling the early identification of salient co-ordination features of a project. The framework also serves as a useful model for understanding the complexities of the field data. What at first seems an unstructured and complex web of personal interactions can be unravelled and interpreted using the elements of our framework. When we presented our interim findings to both Alpha.uk and Beta.uk, we were encouraged by the positive feedback in terms of the accuracy with which we had identified issues which the companies themselves considered to be significant or problematical. Notwithstanding the limitations mentioned earlier in the paper, we consider our framework useful both as a tool for troubleshooting software projects with co-ordination difficulties and also for longer-term research in software engineering co-ordination practices.

Co-ordination theory was useful for conceptualizing, abstracting and analyzing dependencies in project activity, and by comparing projects from a co-ordination theory perspective, we can potentially identify alternate co-ordination mechanisms for addressing similar problems in other projects. The notion of communication genre proved a useful concept for relating co-ordination mechanisms to the day-to-day communicative experiences of people in groups. Collective mind theory was useful for describing the use of tacit knowledge and the role of co-ordination mechanisms in support of this. Collective mind theory has the potential to explain why certain communication genres contribute to, or are associated with, project success - there is much scope here for further work.
In terms of software engineering practice, we can draw a number of conclusions. Shortly after our data collection at Alpha.uk, the CDB project finished, and was considered by the organization to be a successful project. Co-ordination mechanisms which contributed to this success were the careful separation of responsibilities across project levels, the institutionalized but tacit use of copying email for keeping people abreast of project developments, feature-oriented task allocation, situated programmer pairing, and cross-fertilization of skills across programming teams. These are examples of good practice for use on other projects.

The CDB project was managed in an informal manner. Though reference was made to the company’s system development process, this was applied mainly as a generic framework for project planning and control and for providing standard project documentation. The size and context of the CDB project permitted an informal style of management. Yet an informal approach to software project management did not equate to an unstructured or uncontrolled project. Our work would characterise the project as being managed (or self-managed) in a highly controlled and structured fashion through the operation of well understood and appropriate genres and genre systems. There are lessons here for projects of a similar scale and type, and also useful lines of inquiry for investigating large and more complex projects.

The main characteristic of the MMT project at the time of our study was its start up mode. Our analysis highlighted the important role of communication genres in providing the de facto process models in the early stages of the project. The communication genres we identified represented the bottom-up, day-to-day, way of working on the project. Rather than being discarded as unstructured work practices, communications genres can form the basis for process improvement. They are a starting point for project planning and for devising lightweight processes grounded in situated practice.

The substantial effort devoted to making queries by MMT engineers, though time consuming, had the important side effect of maintaining a heedfulness in the way engineers conducted their work. As the MMT group matured it was looking for ways of using its engineers’ time more effectively. Though this is desirable and essential from a software engineering point of view, collective mind theory demonstrates the value of sustaining heedful performance of project tasks compared to striving for habitual performance.

Adopting a co-ordination perspective for the study of software projects offers new insights into software engineering practice and subsequent process improvement. The essence of our approach can be summarized by a comment from one of the senior engineers on the MMT project: ‘maybe communication is more important than technical achievement?’

Acknowledgements

The authors express their thanks to the software engineers who took part in this study. This work was partly funded by The Centre for Software Process Technologies at the University of Ulster which is supported by the EU Programme For Peace And Reconciliation in Northern Ireland and The Border Region Of Ireland (PEACE II).

Appendix A. Interview questions

A.1. Project level issues

Background to the project: Aim? Objectives? Scope? Customer? Context (relationship to other projects, internal and external influences)?

Development issues: Technologies/methods used to develop system and to manage, co-ordinate, allocate work? Timescales/Constraints? Project (Infra)structure (Team composition/user involvement)? Progress to date? Stage of lifecycle, expected size of system, application domain, key project/system attributes? Problems encountered to date? Type of problem? Source? Reasons for occurrence?

A.2. Individual interviews


Team structure and communication mechanisms (to establish how the individual, team and project related to the customer and the wider organization). What are your specific team roles and responsibilities? What decisions do you usually make in your role? Who do you communicate with in the course of your job? (who do you send communications to? who do you receive communications from?) What are the (a) formal and (b) informal channels through which you receive/send information? (Team structure?) How do you know what work you have to do? How do you report progress? For the people you communicate with, what is their role in the project? How does your team communicate with its ‘customer’?

Workflows and task dependencies (to establish key task dependencies, co-ordination mechanisms and collective mind processes in which individual was involved). What are the ‘items/resources’ (documents, information, software, budget, people’s time?) you need to do your job? For any of these items/resources, are they shared whereby you need to negotiate/wait/bid for them? What are your expectations of other peoples’ work? To what extent do you rely on them doing their job as you expect? Do the people you receive ‘items/resources’ from know what you need/expect of them?
What are the 'items/resources' (documents, information, software, solutions, time?) you produce or make available so that other people can do their job? For any of these items/resources, do some of them have to conform to predefined standards or requirements if they are to be of use? What are your priorities in doing/delivering your work (e.g. what aspect of detail do you devote most attention to?)?

Communication patterns with respect to above dependencies? (to establish communication styles and technologies used for working/managing above dependencies) Who/What/When/Where/Why/How/any of prevalent communication processes? e.g. how are any/all of the following used? email? configuration management tools? project database? project plan? timesheets?

Examples of co-ordination in context? (to relate co-ordination mechanisms to the day-to-day practice of the individual, in particular where co-ordination worked well and/or where breakdowns occurred) Describe one or more occasions recently where you were involved in or witnessed good or poor co-ordination... e.g. project clarification situations?, working under timescale pressures?, conformance to quality procedures/standards? What happened? Where did it occur? Who (what roles) were involved? What was the outcome? Why was the interaction considered to be effective or ineffective? What were the implications of this incident for project team performance?

References