To appear in Proceedings of the First Westminster Conference on Professional Awareness in Software Engineering, Royal Society, London, 1-2 February 1996.

COMMUNICATION PROBLEMS IN REQUIREMENTS ENGINEERING: A FIELD STUDY

By Amer Al-Rawas[1] and Steve Easterbrook[2]

[1]School of Cognitive and Computing Sciences
University of Sussex,
Falmer, Brighton, BN1 9QH, UK.
E-mail: ameral@cogs.susx.ac.uk

[2]NASA/WVU Software Research Lab
NASA IV&V Facility,
100 University Dr, Fairmont, WV 26554, USA.
E-mail: steve@atlantis.ivv.nasa.gov

Abstract

The requirements engineering phase of software development projects is characterised by the intensity and importance of communication activities. During this phase, the various stakeholders must be able to communicate their requirements to the analysts, and the analysts need to be able to communicate the specifications they generate back to the stakeholders for validation. This paper describes a field investigation into the problems of communication between disparate communities involved in the requirements specification activities. The results of this study are discussed in terms of their relation to three major communication barriers : 1) ineffectiveness of the current communication channels; 2) restrictions on expressiveness imposed by notations; and 3) social and organisational barriers. The results confirm that organisational and social issues have great influence on the effectiveness of communication. They also show that in general, endusers find the notations used by software practitioners to model their requirements difficult to understand and validate.

1. Introduction

Requirements specifications are based on domain knowledge be it technical, functional, administrative or social. Ideally, the requirements team members are selectively recruited so that both the levels and distribution of knowledge within the team cover all aspects of the domain. However, this is seldom the case because of knowledge shortfalls such as the thin spread of application domain knowledge in most organisations [Curtis, Krasner & Iscoe 1988]. In general, individual members do not have all the knowledge required for the project and must acquire additional information before accomplishing productive work [Walz et.al. 1993]. Knowledge acquisition and sharing can only be achieved through effective communication between the various stakeholders.

It is widely recognised that communication problems are a major factor in the delay and failure of software projects [Curtis et.al. 1988]. This is especially true of "socio-technical" software systems, which must exist in a complex organisational setting. The organisational domains into which such software is introduced are often too intricate and fluid to be fully understood. Hence requirements descriptions are necessarily uncertain. Furthermore, the specification documentation may be so large that no member of the software team has read it all. In this situation, misunderstandings and conflicting views are rife.

There have been a number of field studies into software engineering in general and requirements engineering in particular [Curtis 1990]. Our study differs from previous field investigations in that it focuses on the communication characteristics of the requirements engineering process. Moreover, our investigation not only utilised the experience of software engineering practitioner, it also reflects the views and experiences of endusers based on their recent software procurement projects. The domain of our study was the requirements engineering phase of fully customised software systems development projects. The field study was conducted in two stages using two data gathering methods (interviews and questionnaires).

The research method is described in section 2. Section 3 describes the communication difficulties and their causes. These include difficulties caused by the nature of software engineering notations and methodologies as well as communication barriers caused by social and organisational factors. Each sub-section is supported by results from this field study with reference to the relevant literature. Section 4 concludes the paper with a discussion of these findings and their implications on professional software engineering, outlining some future research questions.

2. Research Method

A combination of learning, data gathering and analysis techniques were applied to investigate the communication problems, their causes and consequences. The two principle sources of information were the literature and the empirical study. The ever growing literature on software engineering in general and requirements engineering in particular was surveyed to gather information about the software development problems, especially those that occur in the early phases, and the sort of tools and techniques that were or are being developed to overcome these problems. A cross section of social science and computer supported co-operative work (CSCW) literature was also surveyed to help in the analysis of the empirical results and reasoning about the possible causes and consequences of communication difficulties.

2.1 Empirical Work

The aim of the empirical part of this research is to provide material for hypotheses, to aid the identification and reasoning about the communication difficulties and their causes and consequences. Although there are inherent complexities in combining qualitative and quantitative methods, it was decided that such an empirical base was essential to avoid unsupported assertions.

The empirical work was carried out in two stages. The first, consisted of informal interviews and observations to establish some knowledge about practices and methodologies of both developers and their customers. These interviews concentrated mainly on the communication channels between agents participating in any software development project, as well as on the problems that can be attributed to the ineffectiveness of those communication channels. Other management and technical issues were also discussed. Most of these interviews were taped for further analysis and reference. The second stage of our empirical work was based on two questionnaires; one for clients (or endusers of the software) and one for the software developers. These questionnaires were designed to get a quantitative evaluation for the various aspects of the communication activities during RE.

To ensure representative coverage our subjects included users and developers of various levels of experiences, qualifications and backgrounds. The users included some who had just had a software system installed and some with an ongoing project. Their experience with computers ranged from absolute computer illiteracy to qualified experts. The developers were all involved in either developing a new software system or maintaining an existing one. Some were also involved in the provision of hardware systems. Their working area covered all aspects of software development from requirements gathering through to maintenance, as well as project management.

In order to utilise the experience of software engineering practitioners as well as the software system procurement experience of their clients, it was necessary to produce two separate

questionnaires; developers questionnaire and clients questionnaire. The former concentrate on the collective experience of practitioners involved in the requirements specification and interpretation. The latter was targeted at endusers who use a new customised software system regardless of their involvement in its development. Another reason for separating the two questionnaires was the technical gap between the two main communities of developers and endusers. Table 1 shows the interviews sample and the questionnaires responses. Targeting the questionnaires was very difficult. We relied on personal contacts plus some commercial directories. We asked the recipients of the questionnaires to pass them to the appropriate people. Questionnaires were sent to over 50 companies in the UK and Oman. Responses represent a cross-section of the companies that were targeted.

	Developers (software practitioners)	**Clients** (endusers of the software)
Interviews Sample Size	6	5
Questionnaires Responses	37	32 (18 Participating & 14 Non-participating)

Table 1: Interviews sample size and questionnaires responses.

The interviews showed that practitioners, no matter how experienced, found it easier to be precise about facts and procedures than about opinions and judgements. Therefore we could not simply ask for direct judgements for each of our hypothesis. Instead, each hypothesis was tested in terms of its outward effects and indicators. Most of these turned out to be multi-dimentional and thus had to be measured through more than one indicator. For example, the effect of organisational power was multi-dimentional in that it governs both the choice participants and their working procedures. It had to be tested in two separate questions each of which had a number of variable answers. In order to get a value for the strength of feeling for each indicator, we employed a Likert Scale method, also known as Semantic differential [Frankfort-Nachmiais & Nachmiais, 1992] . For each variable we used one to five values of strength in relation to the other variables within the same question.

3. Communication Difficulties

Large software projects suffer serious breakdowns in co-ordination and communication throughout the their development life cycle. In this section we present some of the causes for the breakdown of communication during the requirements engineering phase of software development projects.

3.1 One Way Communication Channels

In many ways, software engineering methodologies are communication methodologies. Much emphasis is placed on the notations used to convey information both within the development team and with the various stakeholders. Ideally, the channels of communication between these various communities would be perfect, so that all knowledge is shared. In practice, it is expensive and time-consuming to support extensive communication between the communities, and the channels are restricted to one way communication in the form of specification documents. Curtis et. al. [1988] observed that documentation is ineffective for communication, as it does not help resolve misunderstandings.

Nevertheless, an implicit “over-the-wall” model exists in most software development projects: at each stage in the project, a specification is thrown over a wall to the next team who are waiting to proceed with the next phase. The metaphorical wall is sometimes encouraged by management

practices, but more often is merely a result of the practicalities of co-ordinating a large team. The results of this study showed that specification documents are still the most common format in which analysts communicate requirements back to their clients for validation (*see* figure 1).

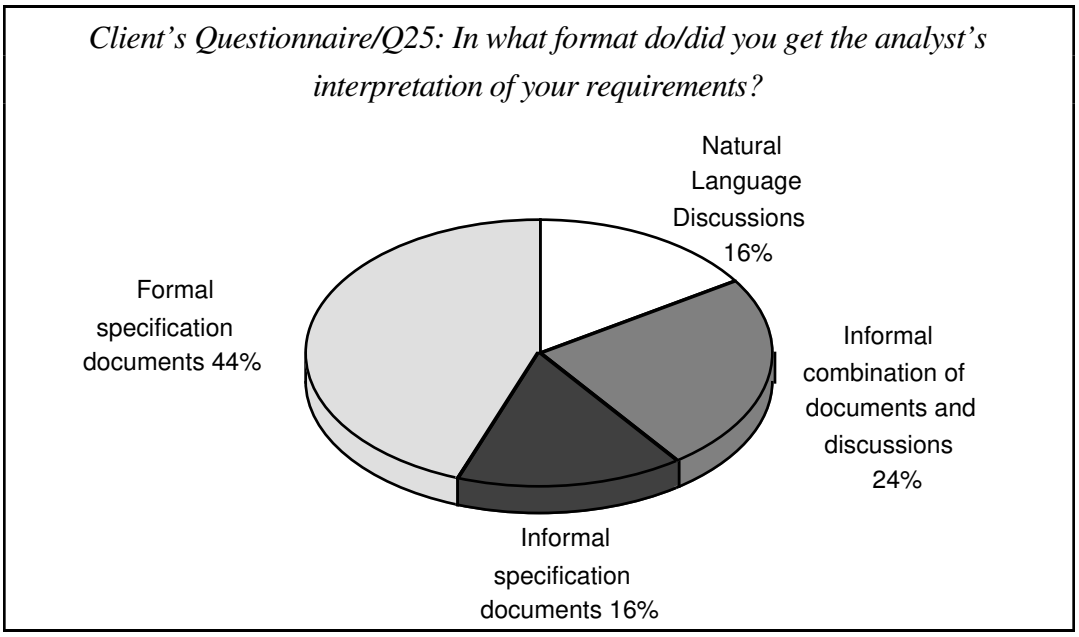

Figure 1: The formats in which requirements are communicated.

There are two standard approaches to this problem. The first emphasises the development of better (richer) notations, and effective use of electronic repositories. The second emphasises the importance of contact between the development team and other stakeholders, and has given rise to practices such as end-user participation and ethnographic techniques. Each of these approaches has its own set of problems, and neither directly addresses the question of facilitating *appropriate* and *effective* communication over restricted channels. Our study showed that practitioners find it easier to adapt a compromise of the two approaches by enriching notations with natural language descriptions and by utilising the personal contact of face-to-face discussions (see results in section 3.2).

Moreover, standard software development cycles rely heavily on documentation as an exit condition in moving from stage to stage. Documents are also used as the media for communicating ideas. Consequently, a colossal amount of paper work is generated. The nature, the domain complexity, the variety of methods and notations and the interdependency of partial requirement make it very difficult to establish total consistency. For example, we found that the recipient of the services provided by one of the enduser organisations was referred to as client, customer, applicant, candidate and land owner by different participants who contributed to the production of the specification documents. All these names were used to describe the same entity. The names used reflected the concern of each stakeholder.

3.2 The Notations War

While programmers, software engineers and analysts are happy talking about the system in terms of it procedures and data structures, endusers prefer to talk about the system in terms of its general behaviour, functionality and applications. The different communities involved in the specification process prefer different types of notation, and various people will be unfamiliar with various notations. For example, a user will not want to learn to read formal specification languages, but the programmer may require these to obtain an appropriate level of detail.

It is often the analyst's responsibility to choose the notations that will best describe the system for each interest group. Thereafter, the chosen notations are used to explain the system differently to each group. In doing so, the analyst combines the notations with other explanation techniques, to make notations easier to read and understand. The choice of the explanatory tools utilised by the analysts and the extent to which they are needed depend on the notation used and the audience's familiarity with the notation.

Typically, two types of knowledge are used as a high level framework to anchor detailed knowledge: the control flow information, which might be represented by specialised notations such as pseudo code and flowcharts, and data structure information, which might be represented using diagrams or a textual description [Shneiderman, 1982]. Some software programs such as the traditional numerical analysis systems have complex control structure with relatively simple data structures. On the other hand, traditional commercial applications have complex data structures with relatively simple control flow. Sheppard, Kruesi and Curtis [1981], conducted an experiment to compare comprehension with nine forms of program description including natural language, a program design language, flowcharts and hierarchical diagrams. They found different results for different types of questions, but no particular style appeared to dominate. However, in their study of program coding from the nine notations, Sheppard and Kruesi [1981], found that the program design language and the flowcharts diagrams were more helpful than natural language descriptions.

Regardless of the chosen notations, most users express their requirements in natural language. Then it is the job of the analyst to translate requirements statements into some kind of representational objects in a domain model. Once the requirements are modelled, they are presented to endusers for validation. At this stage the analysts are faced with another communication problem when endusers are not familiar with the notations used to model their requirements. On the other hand, when analysts, under pressure to keep up with the project schedule, pass raw natural language requirements to programmers, then time is wasted in trying to interpret them. A programmer we interviewed during our empirical study complained that he often has to read large amounts of text in order to understand a single requirement, which could have been represented very concisely using a diagram or a formal notation. In one case he had to read over a page of text to understand the requirements for a screen layout for a particular database form. This, he said, could have been represented more accurately by drawing a diagram which indicate the required dimensions of each section of the screen.

When asked whether their clients find the notations they use readable, only 4 developers (14% of 35 developers who answered this question) said that their notations are readable and understandable to endusers, and 31 developers (86%) said that their customers would normally need additional explanation in order to understand the notations in which requirements are specified. In order to examine the ways in which this additional information is provided, we asked those who provided additional information about the methods they use. Figure 2 shows the results.

We can see from figure 2 that almost half of the respondents (15 out of 31) said that they 'always' annotate the notations with natural language. Around 40% (12) also said they often use this method, around 10% (3) said they seldom use it and only one responded saying they never use this method. This makes 'annotating the notations with natural language' the most popular method for making the requirements representations readable to endusers. The second most popular method is providing face-to-face explanation of the notations. However, around a third (10) of the respondents said that they 'never' go out of their way to choose a notation that is readable to their clients as a practice that aims to aid the communication process. We can also see from figure 2 that for every method, at least half the respondents use them 'always' or 'often'. This suggests that most practitioners usually use one or more of theses methods. The choice of the explanatory method utilised by the practitioner depends on the notation used and the clients' familiarity with the notation.

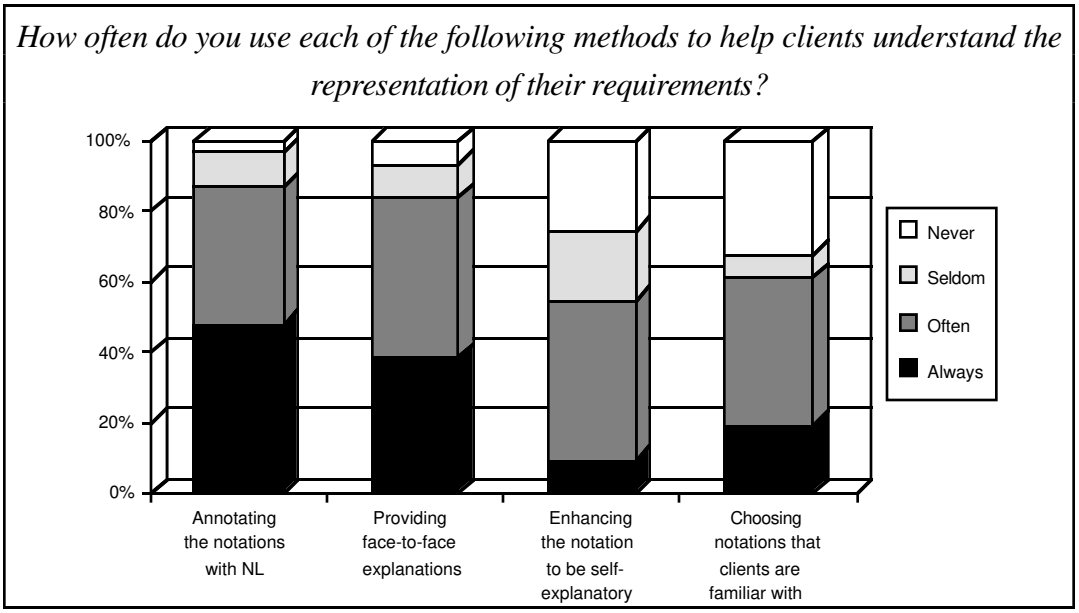

Figure 2: Methods of providing additional explanation

3.3 Organisational Barriers

Organisational barriers that inhibit communication among development teams are often ignored. For example, conventional software engineering practices divide the processes and tasks performed by different groups into separate phases. This can inhibit effective and timely communication between groups performing different tasks at different phases. For example, RE activities and software design activities are often separated. That is, they are performed and discussed by different sets of people. While RE meetings include both developers (usually analysts) and enduser representatives, design review meetings are normally restricted to members of the development team.

Within existing software development practices, design review meetings are used to mitigate communication problems. Formally, these are used to pre-empt problems, by using senior members of the project team to review evolving designs at various stages. This will involve checking for common mistakes, and questioning the assumptions made by the designer. Meetings are also used, usually informally, to respond to problems. Note that design review meetings do not, as a rule, involve anyone outside the development team: meetings with clients are usually stage-managed. The strength of design review meetings lie in their ability to exploit different perspectives to spot things that the designer or programmer might have overlooked. During the process, many of the assumptions held by members of the group will be explicitly tested. The fresh perspectives are able to identify many of the blindness that a designer inevitably develops when immersed in the design process. It also serves to clarify ambiguities in the specification documents. However, Curtis [1990] observed that design meetings are often dominated by a small number of individuals characterised by their deep understanding of the application domain coupled with the ability to translate application behaviour into computational structures. Consequently, the less experienced individuals do not get equal opportunities to voice their ideas and concerns.

We found that the effects of organisational power start with the selection of the client representatives who attend requirements meetings and go on to control what goes on in these meetings. We presented the developers with a set of options for how they choose client representatives, and asked them to give a value for how often they use each one. We present their responses in figure 3.

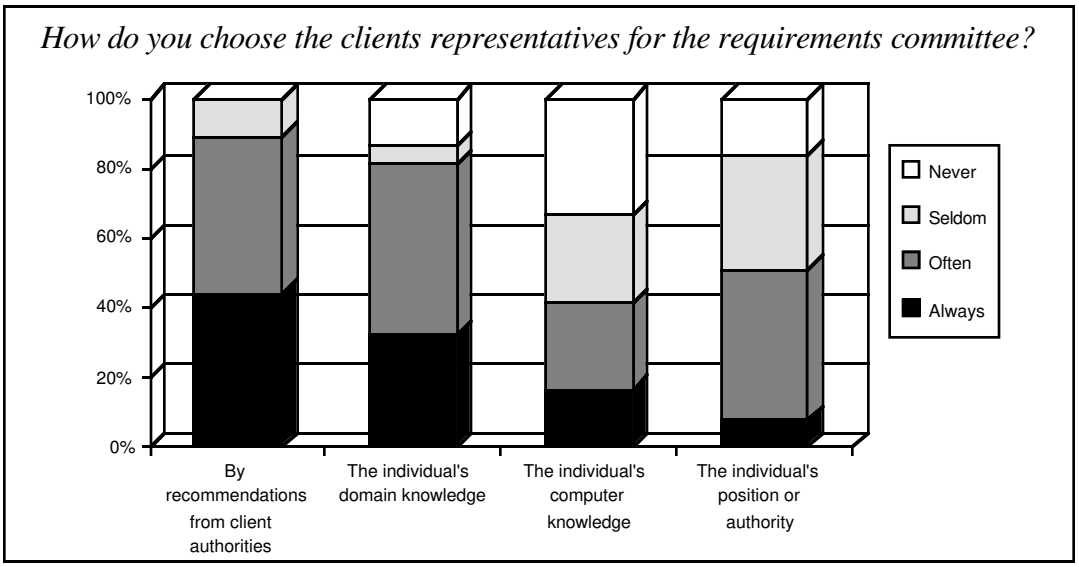

Figure 3: The basis for choosing client representatives for requirements committees.

We can see from the above results that software practitioners depend heavily on the client authorities in selecting the requirements committee members to represent the client organisation. Ideally, 'domain knowledge' would be the most important quality on which software practitioners should base the choice of members. However when control is given to the client's managers, the choices can be based on a number of factors, many of which have more to do with other commercial interests of the clients' business than with the software project. To confirm this we asked enduser who were selected to participate in the requirements specification process to indicate why, in their opinion, they were chosen. We also asked those who did not participate to indicate why they were not chosen. Figure 4 shows a shocking similarity between the responses to these two questions.

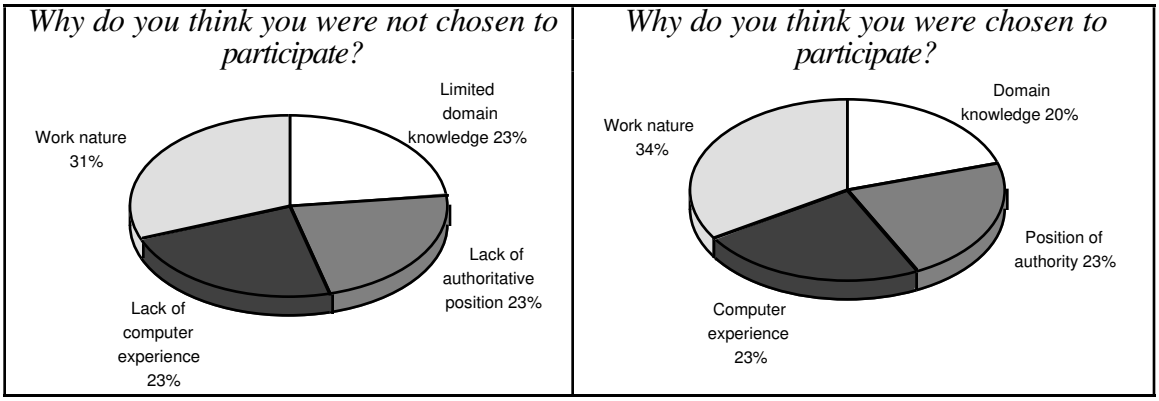

Figure 4: Endusers' opinions on the factors that affect their selection to participate.

We can also see from the above results that 'work nature' is the most decisive factor in the selection of endusers for participating in the requirements specification process. This may sound like the right thing to do, after all domain knowledge depends on the work nature. However the interviews showed that managers, when selecting members of their staff, try to strike a balance between allowing the software practitioners to talk to the right people and maintaining the smooth running of the rest of the business. The results in figure 5 show the amount of interference to business caused by the software procurement project. The amount of interference reported by the participating end-users was less than expected. An explanation for this is that the choice of participants is made in such a way as to minimise interference. This in turn suggests that the

smooth running of the clients organisation is given greater priority than the successful outcome of the software development project.

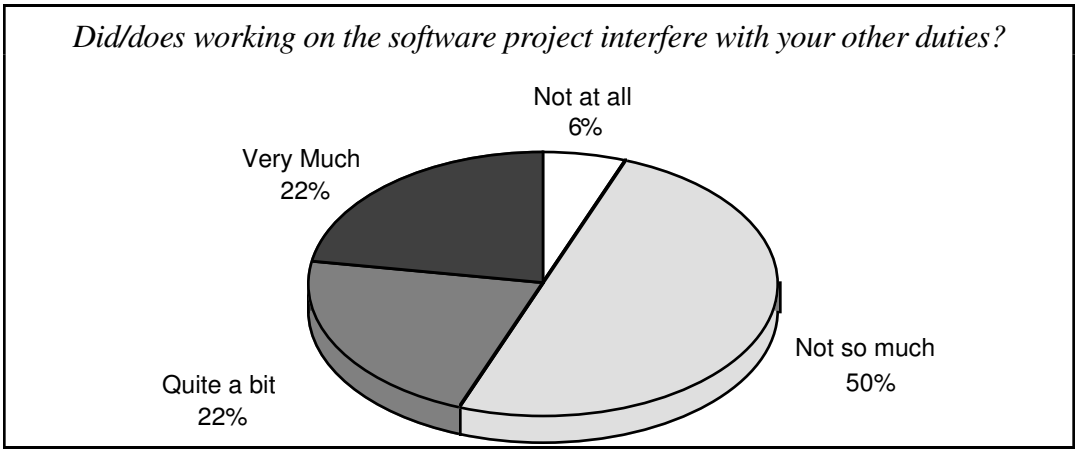

Figure 5: Interference to business caused by the software project.

The above results, show why managers might not select their best staff to be committed to work with the software practitioners throughout the software project life cycle. Such a conflict of interests makes the analysts job much more difficult. We found that the requirements committee members are often people of authority, with limited contact with the low-level tasks of the business. These low-level tasks are often the tasks that need to be computerised.

Informal Communication

Organisations are, traditionally, described in terms of an *Organisational chart.* This is often the first thing handed out to anyone inquiring about the structure of the organisation. However, many important power and communication relationships are not represented in the organisational chart. Mintzberg [1979] makes an analogy between the organisational chart and a road map, where the map is invaluable for finding towns and their connecting roads, but it tells us nothing about the economic or social relationships between the regions. Although, very useful in terms of providing information about formal authority and the division of organisational units, the organisational chart does not tell us anything about the informal relationships that exist in every organisation.

During the interviews that we conducted with practitioners, they outlined many difficulties that are caused by unexpected interactions between elements of the system, be it software modules or humans. In spite of the time and effort spent on studying organisational structures and the flow of power and information through them, our subjects admit that they can never account for all possible interactions and often have to backtrack as a result of discovering a new relation or line of communication that has to be incorporated into the system. These interactions are often too complex to be traced or regulated.

Informal communication is based on friendships, common interests and special co-operative relations creating channels through which information flows easily and perhaps taking shortcuts. On the other hand, such informal communication channels can be destroyed by rivalries and animosities, which can discourage co-operation and affect the normal flow of information. However, there is no specific support for informal communication within the organisational settings for software projects especially at the early phase of RE where analysts and their clients, in many cases, meet for the first time.

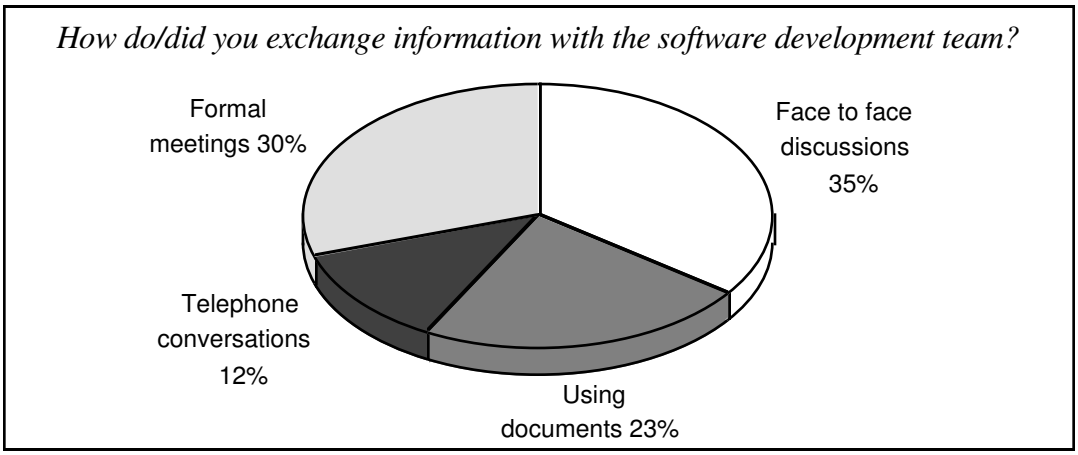

Figure 6: Methods in which endusers exchange information with analysts.

Gotel and Finkelstein [1994] found that practitioners attach extreme importance to personal contact and informal communication. We found that informal communications such as 'telephone conversations' are the least used method in exchanging information between the two communities (*see* results in figure 6). On the other hand, face-to-face discussion are widely used in spite of their higher cost. The lack of alternative interactive support for such informal communication forces analysts to resort to expensive and time consuming discussions and meetings.

3.4 The missing link

The inability to trace the human sources of actual requirements and their related information is identified as the crux of the requirements traceability problem [Gotel and Finkelstein 1994]. Requirements Traceability is vital for all phases of the software development cycle to aid reasoning about requirements and justify changes. Our study showed that the traceability problem is particularly serious in the later stages of requirements engineering (e.g. requirements review) and in cases when new requirements are introduced late in the project life cycle.

In order to check that newly introduced requirements do not conflict with existing requirements, it is often necessary to re-establish communication with the human sources of

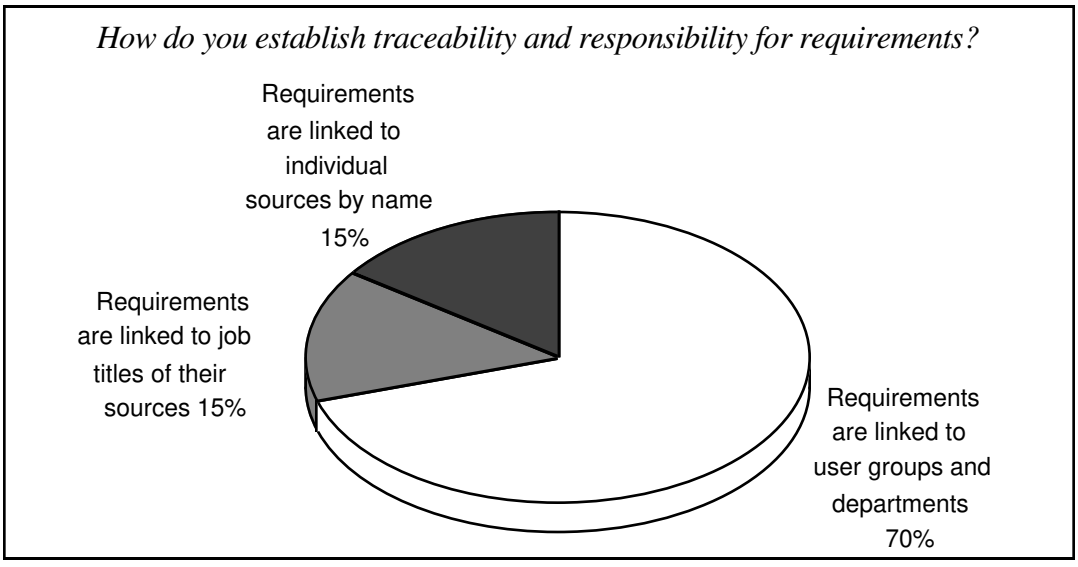

Figure 7: Links that software practitioners use to trace requirements sources.

existing requirements. This is particularly problematic, because by this time the analysts may have reduced, or even halted, contact with the end-users of this software and may even have started working on a new project, while their programmers get on with later phases of this project. Figure 7 presents the links that practitioners use in order to establish requirements traceability to the requirements sources.

We can see from the above results that most analysts link the requirements only very generally to user groups and departments, which means that traceability is not direct. Only 15% linked the requirements to their individual sources by name, which means that those source can be traced directly if necessary. The link to job titles might sound like a good idea, but it does not work in dynamic organisations where people move between jobs and even move between organisations.

There are growing numbers of specialised tools that support requirements traceability, but their use is not widespread. Requirements traceability problems are still cited by practitioners who do not use such tools [Gotel and Finkelstein 1994]. In fact, none of the practitioners we interviewed used requirements traceability specialised tools, and those who used the more general CASE tools were not able to see any major improvements in requirements traceability. This is due to the constraints imposed by many of the CASE tools, the time and effort put into following their strict methodologies, and their limited support for the early stages of requirements specification.

Rationales of design decisions are rooted in the requirements specification. Many design decisions involve trade offs between competing requirements. The decision taken might not be the best solution, instead it may be an acceptable one for parties. Information about theses decisions and the rationales behind them is crucial for the later phases of the software development particularly the maintenance phase. However, because of the limitations of the conventional software engineering methodologies and notations many design rationales go unrecorded. Our observations during interviews showed that some developers do actually keep some kind of traceability for the rationales behind some of the features in the solution product rather than rationales behind the original user requirements.

5. Discussion and Conclusions

There are a number of pitfalls in trying to make effective use of restricted communication channels. One of the dangers is that each community interprets things in the light of their own background assumptions. This is especially problematic with non-interactive communication, such as specification documents, where there is no opportunity to check that the reader has interpreted them as was intended. McDermid [1993] points out a fundamental problem to do with the communication of abstract concepts, in that requirements specifications “document what it is that the analyst thought it was that the problem owner said he thought he might want”. The uncertainties that McDermid describes propagate and multiply at each exchange of information. Robinson and Bannon [1991] use the term “Ontological Drift” to describe the change in meaning of abstract terms as they are passed between different communities.

In this paper we noted that documents are a poor substitute for interpersonal communication. This we attributed to the inherent restrictions of the available notations. While we appreciate the role of meetings such as design reviews in clarifying ambiguities and resolving conflicts in the specification documents, we feel that more can be done to make these documents into a more effective means of communication. A pressing and practical problem is to find-out more about the communicational weaknesses of current notations and methods so we can accommodate for their weaknesses. For each concern, we need to determine what types of question that concern may wish to make of a description produced in a notation. This can only be achieved by observing the meetings and conversations in which descriptions are referred to. For such research to succeed, software professionals must be willing to co-operate and allow researchers to learn from their practical experiences.

There is, and there will always be, what we can call an informal organisation within the formal organisation that is represented by the organisational chart. Software professionals need to accommodate such an organisation in their project management and planing. They need to identify the different categories of communication; regulated, unregulated, formal, informal, interpersonal, internal to the project, external and so on. Informal communications need to be encouraged between all stakeholders. Software practitioners need to invest time in establishing direct and informal contact with endusers. However, much research is needed before support for such informal communication activities can be integrated into CASE tools.

The results described in this paper showed that organisational and social issues have great influence on the effectiveness of communication activities and therefore on the overall success or failure of the requirements engineering process. Such effects start with restricting the selection of client representatives and often propagate throughout the software project life cycle. The study has also showed that in general endusers have little or no influence on the choice of methods and notations in which their requirements are represented and consequently find the notations used by software practitioners difficult to understand and validate. Software practitioners have reported this communication problem and they normally deal with it through a combination of face-to-face explanations and natural language annotations of the notations rather than choosing notations that are readable to their clients.

In general, the findings presented here provide an insight for future research into the communicational and organisational aspects of software development. The practical implications of theses findings include: indicating where and how organisational power is used, outlining the extent to which software practitioners rely on documents as the main communication medium, revealing the dangers of the technical gap between the two main communities and presenting informal communications as the means for bridging that gap.

References

Curtis, B., Krasner, H., & Iscoe, N. (1988). *A Field Study of the Software Design Process for Large Systems*. Communications of the ACM, 31(11), pp.1268-1287.

Curtis, B. (1990). *Empirical Studies of the Software Design Process*., In Diaper et. al. (Eds), Human-Computer Interaction - INTERACT '90, Elsevier Science Publishers, North-Holland. pp.35-40.

Frankfort-Nachmiais, C. and Nachmiais, D. (1992), *Research Methods in the Social Science* (4th Ed.), Edward Arnold.

Gotel, O. and Finkelstein, A. (1994). *An Analysis of the Requirements Traceability Problem.*, Proceedings of the First IEEE International Conference on Requirements Engineering, Colorado springs, 18-22 April. pp.94-101.

McDermid, J. A. (1993). *Requirements Analysis: Orthodoxy, Fundamentalism and Heresy*. In M. Bickerton & M. Jirotka (Eds.), Requirements Engineering. London: Academic Press.

Mintzberg, H. (1979). *The Structuring of organisations*. Prentice-Hall.

Robinson, M. and Bannon, L. (1991). *Questioning Representations*. In L. Bannon, M. Robinson, & K. Schmidt (Eds.), Proceedings of the Second European Conference on Computer-Supported Co-operative Work (ECSCW-91), 25-27 September, Amsterdam, The Netherlands. pp. 219-233.

Sheppard, S. B. and Kruesi, E. (1981). *The effects of symbology and spatial arrangement of software specifications in a coding task.*, Proc. Trends and Applications: Advances in Software Technology. Held at NBS, Gaithersburg, MD, available from IEEE,1981, pp.7-13.

Sheppard, S. B., Kruesi, E. and Curtis, B. (1981). *The effect of symbology and spatial arrangement on the comprehension of software specifications*, Proc. 5th Int. Conference on Software Engineering, San Diego, CA, available from IEEE,1981, pp.207-214.

Shneiderman, B. (1982). *Control Flow and Data Flow Structures Documentation: Two Experiments*, In Ledgard, H. (Eds), Technical Notes: Human Aspects of Computing, Communications of the ACM 25 (1), pp.55-63.

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