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

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)

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).

Client's Questionnaire/Q25: In what format do/did you get the analyst's

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.

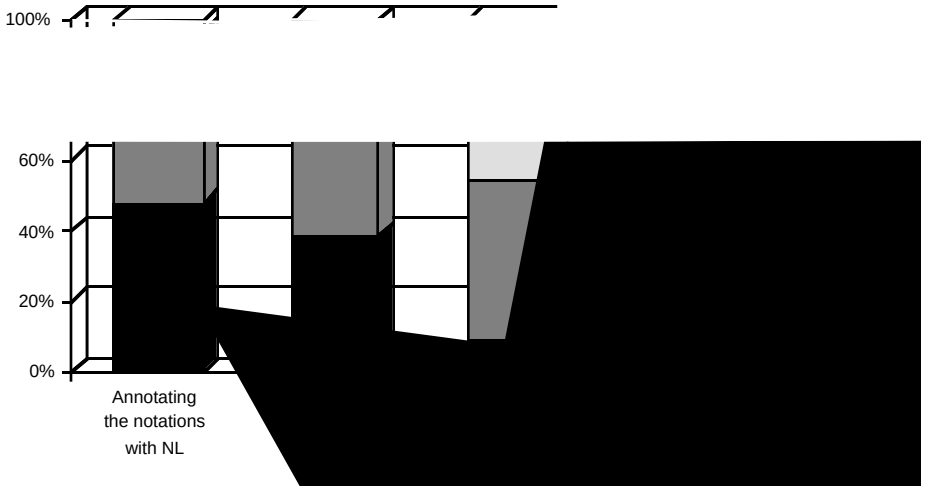
Typically, two types of knowledge are used as a high level framework to anchor detailed knowledge: the control flow information, which might be represented by specialised notations such as pseudo code and flowcharts, and data structure information, which might be represented using diagrams or a textual 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.

How often do you use each of the following methods to help clients understand the representation of their requirements?



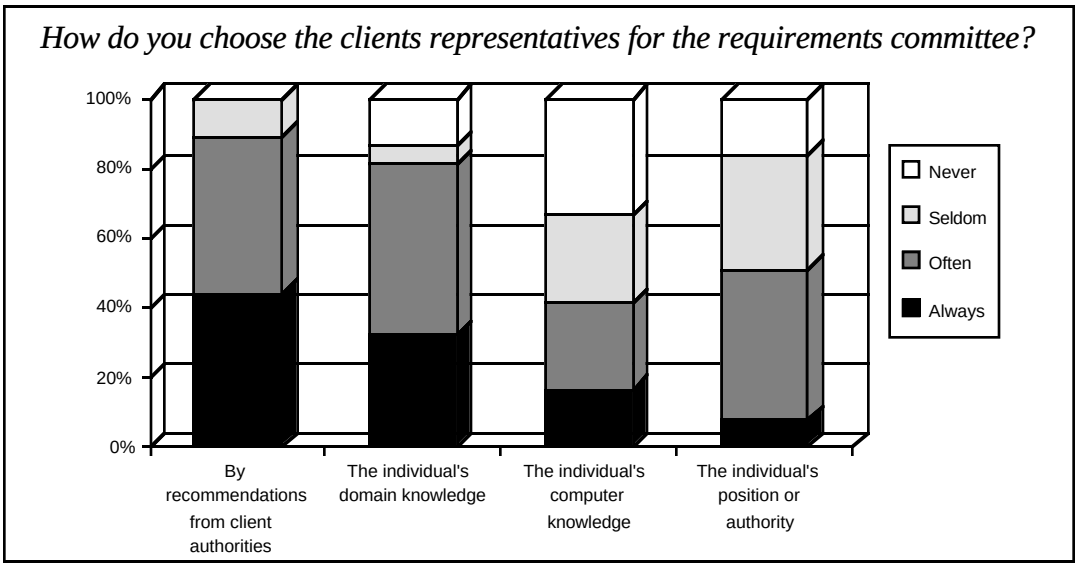
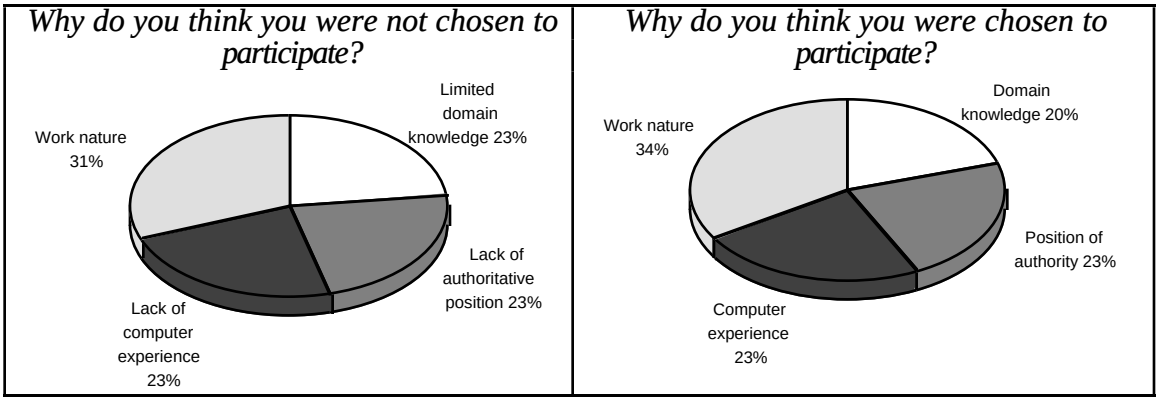


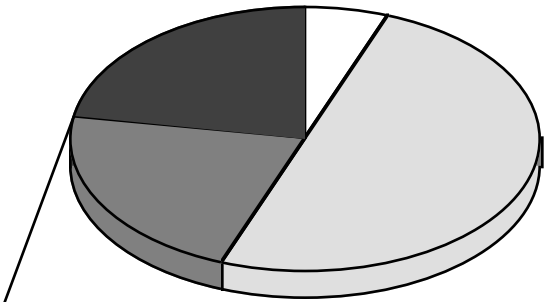
Figure 3: The basis for choosing client representatives for requirements committees.

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.

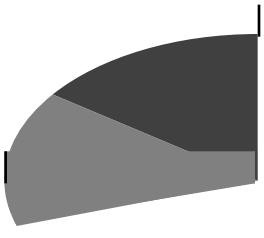


smooth running of the clients organisation is given greater priority than the successful outcome of the software development project.

Did/does working on the software project interfere with your other duties?



How do you establish traceability and responsibility for requirements?



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 theirstrict me
abto t1troermtod(7 pry Tw (passintsven movditi0rnd pcommunntieaanisati0ns.)' 0.071 Tc 220012 Td In pha
strequipaerc t accommodat fd sufollowinatio7e

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),

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.