The Dry and the Wet[1]

Joseph A. Goguen[2]

Centre for Requirements and Foundations
Programming Research Group
Oxford University Computing Laboratory
11 Keble Rd., Oxford OX1 3QD, United Kingdom

Abstract:
This paper discusses the relationship between formal, context insensitive information, and informal, situated information, in the context of Requirements Engineering; these opposite but complementary aspects of information are called "the dry" and "the wet." Formal information occurs in the syntactic representations used in computer-based systems. Informal situated information arises in social interaction, for example, between users and managers, as well as in their interactions with systems analysts. Thus, Requirements Engineering has a strong practical need to reconcile the dry and the wet.

Following some background on the culture of Computing Science, the paper describes some projects in the Centre for Requirements and Foundations at Oxford. One of these is a taxonomy for Requirements Engineering methods. Another is applying techniques from sociology and sociolinguistics to requirements elicitation, and in particular, to determining the value system of an organisation. These projects draw on ideas from ethnomethodology and Conversation Analysis. The paper also demonstrates that structures as dry as abstract data types occur in the ordinary discourse of social groups.

1 Introduction

This paper is concerned with the relationship between formal, context insensitive information and informal, situated information; these two opposite but complementary aspects of information will be called "the dry" and "the wet." Following some cultural background from Computing Science, the paper describes the Centre for Requirements and Foundations at Oxford, using some of its projects to raise and discuss issues about the relationship between the dry and the wet.

One conclusion is that novel approaches may be needed, as well as a willingness to be eclectic rather than dogmatic. In particular, it may be necessary to abandon, or at least dilute, the notion of "objectivity" in order to properly handle situated information. In this respect, I draw on ideas from ethnomethodology and Conversation Analysis, as well as from the work of Jean-Francois Lyotard on "postmodernism," Bruno Latour on

[1]The research reported in this paper has been supported in part by a contract from BT, and grants from the Science and Engineering Research Council, the System Development Foundation, and the Fujitsu Laboratories.

[2]Also with SRI International, Menlo Park CA, USA.

"immutable mobiles," and Leigh Star on "boundary objects." Also, we discover that structures as dry as abstract data types can be found in the ordinary discourse of social groups, such as sports fans.

This paper uses Requirements Engineering as a site for research on the relationship between formal and informal information. Informal situated information arises in social interaction, for example, in the worlds of users and managers, as well as in their interactions with systems analysts. Formal structures occur in the internal representations of computer-based systems, which are subject to the formal syntactic and semantic rules of computers and computer languages. Thus, Requirements Engineering has an especially strong practical need to reconcile the dry and the wet. Indeed, one might almost say that this kind of reconciliation is the essence of Requirements Engineering.

Acknowledgements

I wish to thank the members of the Oxford Centre for Requirments and Foundations for many stimulating conversations on the topics discussed here, particularly Marina Jirotka and Matthew Bickerton; in fact, many of the ideas here arose in response to a question asked by Marina. I also wish to thank my wife Kathleen for carefully reading several drafts of the paper and offering many helpful comments. I thank the participants of COSCIS'91 (Conference on Collaborative Work, Social Communications and Information Systems) in Helsinki for their many interesting comments following the presentation of this paper as part of a panel discussion there; particular thanks for Prof. Ronald Stamper for organising the panel and inviting me to participate in it. Thanks to Frances Page for preparing the figures in this paper. Finally, special thanks to Dr. Charlotte Linde for our long collaboration, during which I learned much of what I know about Discourse Analysis.

2 Four Cultures in Computing

Difficulties with large and complex systems are accelerating the fragmentation of Computing Science. This can be seen in the increasing frictions among the four subcultures within Computing Science that are described in this section. Evidence for such a division can be found in systematic differences in the vocabulary, conferences and journals used by computing scientists, as well as in the semantic domains of their conversations, lectures and publications. However, this is not the place to pursue such evidence in detail.

2.1 Hacker Culture

Members of the old culture, which we shall call the "*hacker culture*,"[3] just sit down, "bang code," and patch it until it works. While this can be effective and amusing for small projects, it does not work well for large projects, and industrial practice typically

[3]In the U.S., the media seem to be trying to change the meaning of the word "hacker" so that it connotes law breaking, rather than a fascination with computing and communications technology as such. Here, we use the original meaning.

imposes a complex system of management controls, including documentation standards, required walk-throughs, change management, and so on. Hackers hate this, and argue that it does at least as much harm as good. Hacker culture is somewhat undisciplined, but it is also lively and creative, and hackers can often build effective systems where other approaches fail.

2.2 Dry Culture

Another group, let us call them the "*dry culture*," take the precision of formal mathematical logic as their ideal. They say that computer programs can be seen as mathematical objects, and the criteria for their correctness can be formalised as mathematical formulae, so that the whole of programming can be seen as a branch of mathematics. Moreover, the same can be done for hardware, by regarding system designs as mathematical objects. System development can therefore be formalised as a process of moving, step by step, from an abstract statement of what is wanted, to an object that delivers it, in such a way that each step can actually be *proved* correct (e.g., see [25]).

There are indeed some successful examples of this, but there are also a number of difficulties. One difficulty is that it can be a lot of work. Perhaps computer-based tools can help reduce this effort; for example, syntax checkers, theorem provers, and sophisticated configuration managers should be useful. But with the present state of the art, it appears inevitable that a fully rigorous system development approach will take more effort than an informal approach. Advocates of the dry culture argue that the extra effort is worthwhile for certain systems, including safety critical systems. Interestingly, hardware projects may be easier to automate, because certain useful formalisms are also simple, although somewhat incomplete.

Examining the sources of difficulty for large systems projects reveals that incorrect code is a relatively insignificant factor, dwarfed by incorrect specifications and (even more significantly) incorrect requirements [2, 3]. Formal methods can indeed increase the likelihood that code satisfies some given specification; but this is not much help if the specification is wrong. However, by providing a precise *language* for specifications, formal methods can also aid in the social process that produces specifications, by reducing misunderstandings. Trying to formally verify certain properties of a specification may also reveal some bugs. But experience seems to show that formal methods are most effective when used in a relatively informal way as a medium of communication [26]. This can make advocates of the dry culture uncomfortable, because it is rather far from the ideal of mathematical perfection.

Please note that I am not denying that formal mathematical models can be useful in many important situations. For example, compiler writers use formal language theory, and information system designers use data modelling. But the higher levels of formality seem to be unsuitable for communication with (most) managers and users, and can even slow down communication among designers and implementers. Moreover, formal methods do not (and cannot) address the kind of issues considered in the next subsection.

2.3 Wet Culture

A still closer examination of the major source of difficulty in large software projects — which is their requirements — shows that factors from the social world often lie at the root. Advocates of what I shall call the "*wet culture*" say that this must be taken seriously, because social, political and cultural factors ultimately determine the success of systems. For example, if a large information system does not meet the day-to-day needs of its users, then it may not be used at all. This culture therefore calls for using methods from the social sciences, particularly during the requirements phase of large system development projects. A significant difficulty with this, however, is that few computing scientists know very much about the social sciences; indeed, many computing scientists have little sympathy with the social sciences, and prefer the relative certainties of hardware, software and dryware, to the ambiguities, conflicts and vagueness of wetware[4].

2.4 Theory Culture

There is a fourth culture that should also be mentioned, that of the *theoreticians*. They have close links with both mathematicians and the dry culture, and (in general) like to prove interesting theorems about some area of Computing Science that has become sufficiently formalised. Sometimes they may produce "airware," i.e., results with little or no immediate practical application. It is perhaps sad, but true, that basic definitions often have the greatest practical value, and that hard theorems rarely have direct practical value. Perhaps this helps to explain why the importance of theoretical work is often underestimated. But results that seem useless today may have significant practical importance in the future. For example, the whole field of Computing Science arose out of highly theoretical work in metamathematics due to Turing; and revolutionary improvements may well arise out of some of today's seemingly most esoteric research.

3 Requirements as a Site for Research

The *lifecycle* of a system is often considered to consist of a number of "*phases*". There is no universal agreement on what these are, and in fact, their number, names and boundaries are somewhat arbitary, but roughly speaking, the following may be distinguished:

1. *needs*, in which the desirability of a certain kind of system is identified at an executive level;

2. *requirements*, in which properties that the system must satisfy in order to succeed are determined;

[4]I have been using the term "wetware" for patterns of social interaction since the early 1970's, but unfortunately, it has recently taken on a new meaning in the so-called cyberpunk literature. In both cases, the name is derived from the fact that human (and more generally, most animal) bodies are mostly water. "Dryware" refers to the formal documents (except software) associated with systems, including requirement, design, and specification documents. Of course, such documents are not in general fully dry, i.e., fully formal, but also have an informal situated aspect.

3. *design*, in which a rough architecture of the system is determined, e.g., as a block diagram of its major components;

4. *specification*, in which the behaviour of the components is described;

5. *construction*, in which the components are actually built, and then assembled to form the system;

6. *validation*, in which the resulting system is tested against its specifications;

7. *deployment*, in which the system is installed in its target environment; and

8. *maintenance*, in which the system is continually modified, upgraded and debugged.

It is interesting that most of the effort for typical large systems goes into the maintenance phase. Some advocates of dry culture have argued that this is because not enough effort has been put into being precise in earlier phases, particularly specification, but I believe the real reason is that much more is going on in the so-called maintenance phase than meets the eye. The first observation is that in real projects, there is no orderly progression from one stage to the next (contrary to the so-called *waterfall model*) but rather, there is a continual process of projection forward and backward; for example, the client may perceive a new need (or reassess an old one), or the implementer may perceive a new opportunity (or impossibility); also, rapid prototypes may be built to assess feasibility, and can lead to changes in requirements, specifications, etc. In fact, both feedback and feedforward go on all the time, at least in successful large projects. In particular, reassessment of requirements, specification, and code, as well as redoing of documentation and validation, is very much a part of the so-called maintenance phase; thus, the maintenance phase may contain smaller versions of the complete lifecycle[5].

The waterfall model has been widely criticised, and many alternatives have been proposed; even so, feedback and feedforward are not as appreciated as they should be, and many projects have process models that severely limit adaptation. One very pernicious factor is procurement processes that attempt to rigidly separate phases with contractual barriers in the name of competition.

The belief that the steps of the lifecycle should be executed sequentially is a crude form of the myth that there is a more or less unique best system to be built. Similarly, in the requirements phase, one might believe that there is a unique best model of the organisation. However, requirements are *emergent*, in the sense that they do not already exist, but rather emerge from interactions between analysts and the client organisation. Moverover, requirements continually change, just as the organisation does, and sometimes the requirements process can have a significant impact on an organisation, for example, causing it to adopt some recommendations before the system is actually delivered, and perhaps even rendering the system unnecessary.

As already noted, the requirements phase of a large software and/or hardware development project is the most error-prone; moreover, these errors are the most expensive

[5]It follows that methods that help with requirements can also help with some aspects of maintenance.

to correct [2, 3]. Consequently, this phase has the greatest economic leverage. Unfortunately, it is also the least explored, and has the least satisfactory intellectual foundations. It therefore seems a good place to invest effort.

3.1 The Oxford Centre for Requirements and Foundations

At the Oxford Centre for Requirements and Foundations, we are trying to reconcile the cultures described in the previous section. We want to do practical work on real systems, and we also want to explore the foundations of this difficult area. The need for progress is acute. For example, large projects have an astonishingly high likelihood of failure[6]. Our research is particularly focused on the earliest phases of the software lifecycle, the requirements and specifications.

But the problems in this area seem very difficult, so that it would not be wise to expect too much progress too quickly. Also, we believe that some really new ideas are needed, because the social issues that lie at the root of the difficulties are not amenable to modelling by traditional technological techniques. Consequently, we are exploring techniques from Discourse Analysis[7].

At present, we have three major projects in the requirements area. The first of these aims to classify and evaluate existing requirements methods in a scientific, unbiased way; this is important because of the many exaggerated claims that can be found in the marketplace. Some preliminary results from this project are described in Section 4 below. The second project is a case study, exploring the viability of using social science methods in Requirements Engineering. Some preliminary results from this project are presented in Section 5 below.

We expect to start a third project on "hyper-requirements" in the near future. This will consider requirements for systems to improve the *traceability*, *accessibility*, *modularity*, and *reusability* of the numerous objects that arise and are manipulated during the requirements phase. We will explore a flexible, user-configurable object oriented database to support links among related objects (in a wide variety of senses of "related," as in hypertext systems), in order to ground requirements decisions in the prior objects that justify them. We hope that this will support the *situatedness* of requirements decisions, as well as their *traceability* through an idealised chain of causal stages. This approach associates related objects into what are called *module clusters* in [5]. Generic modules may lead to improved reuse, and a generalised notion of *view* is used to organise the links that give a rational reconstruction of the reasons that lie behind decisions. These techniques should also be useful in the design phase, during which specifications are produced from requirements, as well as in the coding and maintanence phases. We hope to eventually produce a prototype for such a system, perhaps with a hypermedia user interface.

There are three other projects that we expect to start soon in the Centre: a study of

[6] A 1979 study by the U.S. Government Accounting Office [27] reports that 95% of funds were wasted in a sample of nine projects totalling nearly 7 million dollars; there are some indications that things may be better now.

[7] In this paper, I use "Discourse Analysis" in a broad sense that includes techniques for analysing written texts in their social context, as well as techniques for analysing spoken language and other natural interaction.

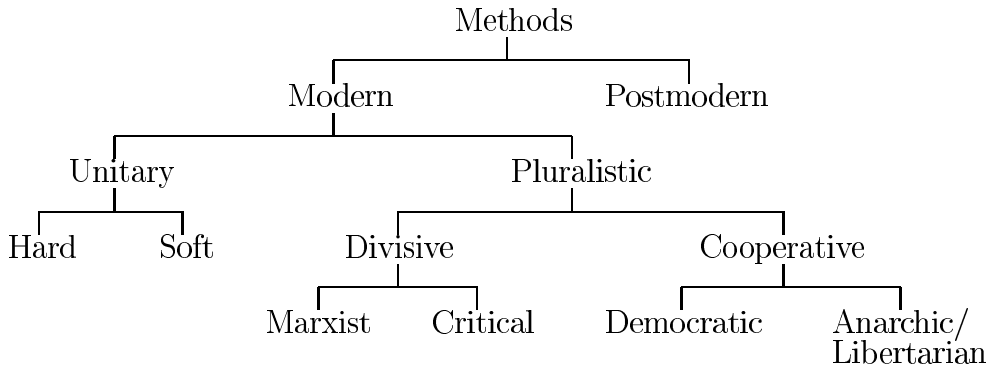

Figure 1: A Taxonomy for Requirements Methods

multi-media medical records, with a view towards developing common European standards; a study of the semantic foundations of object oriented information systems; and a long term study of situated, adaptive software. The first two are sponsored by the European Commission under its Esprit programme, and the third by the Japanese government agency MITI. In addition, there are related projects of a more formal character, concerned with object oriented specification and theorem proving, sponsored by SERC, DTI and Esprit.

4 The Classification of Requirements Methods

We are working under the hypothesis that every requirements method has its own, usually unarticulated, theory of organisations, which is therefore an implicit sociological theory. Hence, a good classification of sociological theories would provide a good basis for classifying requirements methods. We have found a useful starting point in the work of French philosopher Lyotard [23]. His scheme first distinguishes between (what he calls) *modern* and *postmodern* theories; in the former, it further distinguishes between (what we call) *unitary* and *dual* approaches. Unitary, or *systems* theoretic, approaches assume that there is some unique pre-existing "real system" to be "captured." *Dualistic*, also called *critical* or neo-Marxist, approaches assume that the most important feature of an organisation is the split between its workers and its managers. Postmodern theories assert that organisations are composed of many "local language games"[8] that cannot necessarily be unified, or neatly divided into parts.

We[9] have modified Lyotard's taxonomy, first by subdividing the *unitary* class into *hard* and *soft* subclasses, and second by extending the *dual* class to be *pluralistic*, with major subclasses *divisive* and *cooperative*, which are further subdivided into *Marxist* and *critical*, and *democratic* and *anarchic/libertarian*, respectively. See Figure 1.

Most existing work in requirements falls within the systems theory classification; for example, the most familiar structured design methods are all hard unitary. Work in the

[8]This phrase was introduced by Lyotard [23], inspired by Wittgenstein's late work on language games.

[9]This version of the taxonomy is the result of much discussion within the Centre, and some significant help from Dr. Susan Leigh Star.

Scandinavian tradition falls into the cooperative democratic classification. There is very little work in requirements that falls within the postmodern classification, although there is some related work in Computer Supported Cooperative Work. For this reason, we are undertaking a case study ourselves, as described in the next section.

A major goal of our survey project is to compile a *Methods and Tools Handbook*, intended be useful to managers in deciding how to organise actual projects. This document will include a taxonomy of relevant methods and related disciplines, with an annotated selection of relevant books, papers, individuals, groups, and especially of methods and tools. We will try to identify the best of these, and to indicate the applications for which they seem particularly suitable. We will be especially concerned with actual experience, and we will also be alert to the possibilities of hybrid methods.

5 Discourse Analysis and Requirements Engineering

Although natural language is often criticised for its informality, ambiguity, and lack of explicit structure, these features can actually be advantages for requirements. For example, these features of natural language can facilitate the gradual evolution of requirements, without forcing too early a resolution of conflicts and ambiguities that may arise from the initial situation. Also, natural language, possibly supplemented by graphics, is often the medium preferred by the individuals who represent the client.

There is a growing body of evidence that natural language is actually far more structured than most people realise, and that this natural discourse structure carries much important information about the structure of what is being described. For example, work by Abbott [1], and by Enomoto, Horai and others [30] shows that the nouns and verbs used in stating requirements provide important clues to an object oriented design for the system. In particular, the nouns give clues about classes and their attributes, and the verbs give clues about methods. Syntactic structure can also show relationships of inheritance and clustering.

Work by Goguen and Linde [7] shows that task oriented descriptions can be readily translated into data flow diagrams. Therefore, the application of linguistic analysis to naturally occurring instructional discourse is highly compatible with standard structured design methodologies, such as those of De Marco [24], Jackson [14], and Yourdon [34]. Other research by Goguen and Linde [22, 12, 8, 10] uses techniques from sociolinguistics to show that explanations, directions, and other everyday types of discourse have a well defined high level structure that relates directly to their semantic domains. What is called "command and control" discourse in [7] is relevant to Requirements Engineering, because it can reveal the structure of tasks unfolding in real time.

We consider that all such discourse structures are situated, emergent, open, locally organised, and contingent. As mentioned before, "situatedness" refers to the social context that is needed to fully understand such structures, and "emergence" refers to the claim that these structures are jointly constructed by members through their on-going interactions. "Openness" refers to the view that theories of such structures cannot in general be given a final and complete form, but should remain open to revision in the

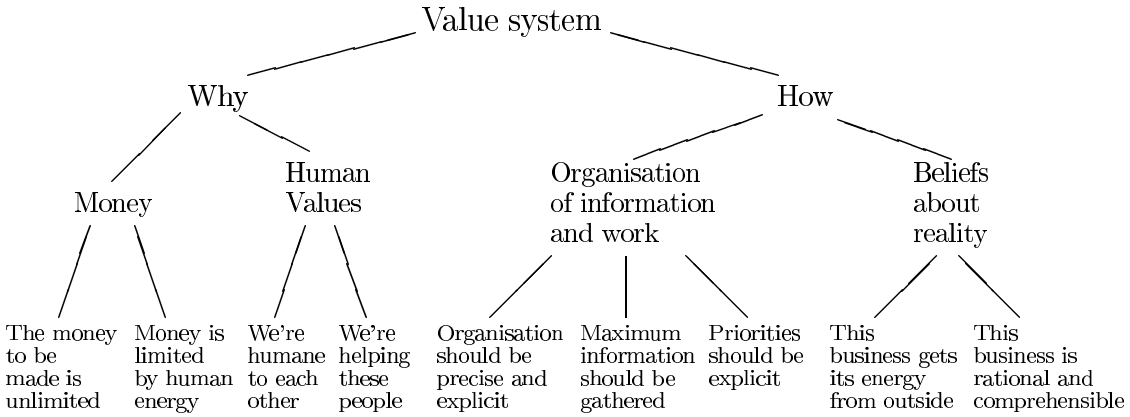

Figure 2: A Value System Tree

light of further analysis and further data. "Local organisation" refers to the idea that what participants do is conditioned by details of the situation in which they find themselves, including, for example, the timing of a previous utterance. "Contingence" refers to the fact that the interpretation of past actions, and the possibilities for future actions, depend upon the details of current interactions.

One specific result from [7] is that organisations (and parts of organisations) tend to have value systems, whose structure can be expressed as a *value system tree*, with higher level nodes corresponding to higher level values, and lower level nodes corresponding to refinements, applications, or corrections of superordinate nodes; this structure is similar to that of classes under inheritance in object oriented programming. The method used in [7] for discovering such structures is to analyse what are called *evaluations* in narratives, i.e., roughly the morals of stories. It may be surprising to some that such evaluations are an integral part of the internal structure of naturally occurring narratives, and are not confined to a summary statement at the very end. Indeed, the discourse and syntactic placement of evaluative material can be an important clue to its position in the value system tree. The classic work on the structure of narratives is due to Labov [18].

Figure 2 shows part of a value system tree obtained by Goguen and Linde [6] in 1978. It represents the values of a small corporate recruitment (i.e., "head hunting") firm. All nodes at the first three levels are analysts' constructions, with support from the data. The phrases in the fourth level are taken from the evaluative clauses of stories and jokes collected at this firm, mostly during lunch and coffee breaks. (Some of the nodes at the fourth level of Figure 2 have two more levels below them.) Note the contradiction between the first two nodes on the fourth level. This illustrates the fact that real value systems are not necessarily consistent. Perhaps this is one of the reasons why it is difficult (or perhaps impossible) to elicit values from members just by asking for them. Indeed, value systems, like many other aspects of social life, are hidden.

The value system tree of the client (and/or end user) can help the analyst make appropriate trade-offs between conflicting requirements (such as cost *versus* almost everything

else, including speed and functionality). The hierarchical structure of the value system tree can also tell analysts which requirements should be given precedence over others, and can clarify the nature of the conflicts involved [7], because higher level values are more significant. It is also possible to assign weights to values based on the frequency of clauses that support them. The analysis of task oriented discourse also seems promising for Requirements Engineering, because it can provide the fundamental units of traditional analysis, which are the main items of information involved significant tasks, and the main transformations performed on them. The analyses of task oriented discourse and of narratives were the two basic techniques used in [6]. See Section 5.1 for some further discussion of value system trees.

It can be difficult to gather good data upon which to base requirements. Experience suggests that simply asking managers what they want often does not work well. We believe that ethnomethodology [4] can provide useful general guidelines for how to collect high quality data about social interaction. This approach recommends looking closely at how competent members of a group actually organise their behaviour, and in particular, at the concepts and the methods that they use to render their actions intelligible to one another; this contrasts with presupposing that the concepts and methods of the analyst are necessarily superior to those of members. We are working with the hypothesis that members' concepts and methods can be useful inputs to the requirements process.

On the other hand, analysts often need to do things that members do not need to do, and then they may need methods and concepts that members do not use. For example, analysts may want to accumulate statistics on telephone calls that would be incomprehensible to those making the calls. Thus, the principles of ethnomethodology may not apply to the *analysis* of data. Even so, analysts can generally benefit from knowing the methods and concepts of members, particularly when they want to do something that members regularly and ordinarily do themselves. It should also be noted that relationships between ethnomethodology and the structure of large-grain discourse types, such as narratives and plans, have not been much explored, and there may be some incompatibilities, as discussed in [15]. We plan to pursue this issue further. See [21], [9] and [15] for relatively comprehensible expositions of ethnomethodology.

The emphasis of ethnomethodology on members' competence, members' concepts, and members' methods is a significant departure from the traditional dogmas of "scientific method", which call for a rigid separation between subject and object, i.e., between observer and observed. But physics has already moved rather far from classical objectivity[10], and so it should not be surprising if sociology, and social aspects of computing, had to go even further.

Conversation Analysis grew out of ethnomethodology through the work of Sacks on how speakers organise such details as timing, overlap, response, interruption, and repair in ordinary conversation (e.g., see [29]). Interaction Analysis extends Conversation Analysis from audio to video data. See [13] for a recent overview of Conversation Analysis, and [17] for a collection of essays on and applications of Interaction Analysis.

An essential property of real social data is that it is *situated*, that is, it can only be

[10]Penrose [28] gives an elegant and readable exposition which shows just how strange contemporary physics can be.

fully understood in relation to the particular, concrete situation in which it occurred. As Suchman [32] points out, situated actions are embodied, *ad hoc*, contingent, and local; it is only our *post hoc* explanations for them that appear to be what Latour [19] calls *immutable mobiles*, which are structures that can be transported from one context to another without undergoing essential change, and which therefore can participate in the discourse of science. Thus, "immutable mobiles" are information structures that have been at least partially dried out.

It is also worth noting that analysts construct immutable mobiles in order to support their positions in agonistic encounters, such as meetings with managers, or meetings with peers. Data flow diagrams are a good example. As Latour [19] points out, this is an issue of power, a way of mobilising support by compressing large amounts of data into simple graphical representations.

Information systems are a particularly interesting site for research. Such systems are repositories for immutable mobiles, and increasingly they provide the means for producing new immutable mobiles, and for transporting them into new contexts; information systems are powerful engines for concentrating and applying power. Consequently, the design of an information system is a natural occasion for power struggles, and it is important that the human interests of all stakeholders be recognised and protected. Failure to do so explains why many large systems have not worked well in practice. In fact, for an information system to be successful, it will often have to serve as what Star [31] calls a "boundary object," which is used and interpreted in different ways by different social groups, in order to meet their different needs.

We are now designing a case study of a "live" project within BT to test the practical application of techniques of Discourse Analysis to Requirements Engineering. This will involve the analysis of video tapes of interviews conducted by Requirements Analysts, and also of their internal working sessions, with a view to explicating their work practices, and developing better methods for requirements elicitation and analysis.

5.1 Two Examples

This subsection uses two examples to illustrate some issues that arise in trying to reconcile the situated and the context insensitive information. Each example organises some data into what computing scientists call a tree structure.

The first example is the value system tree of Figure 2. Here, the tip nodes are situated, in that they arise directly from actual narratives by members. Many interior nodes, which express superordinate values, are also situated in this sense, but others may be created by analysts, by clustering nodes into larger and larger related groups, in the general style of the KJ method [16]. The edges, which express relationships of subordination, are to a certain extent situated in that there may be direct evidence for them in the structure of the discourse, and in any case, members can be asked about them.

However, the tree as a whole is an analysts' construction: members do not talk about properties of the tree as a whole, such as its number of nodes or edges; they do not have names for these concepts, and they do not regard questions about them as meaningful. Instead, this tree serves as a formal summary of the data that has been collected so far, and of the analysts' current understanding of it. In fact, this structure is not only

Visitors' Cup. Heat 1: Jesus, Cambridge *v.* Christ Church; Heat 2: Oriel *v.* New College; ... Heat 8: Lady Margaret *v.* winner of Heat 1; ... Heat 26: Winner of Heat 23 *v.* winner of Heat 24; Final: Winner of Heat 25 *v.* winner of Heat 26.

Figure 3: Draw for the Henley Regatta

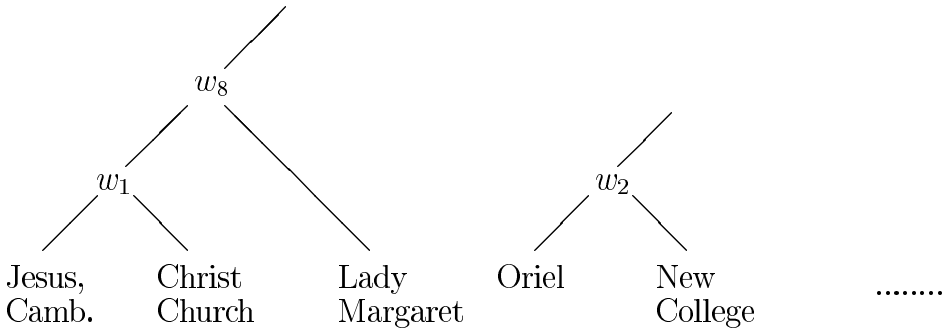

Figure 4: Tree for the Henley Regatta Draw

formal, but is also *open* and *contingent*, in that it remains subject to revision in the light of further data and further analysis. This is typical of summary objects for data from the social sciences, as pointed out, for example, by Lévi-Strauss [20].

Our second example comes from Toulmin's suggestive book *The Uses of Argument* [33]. Here, both the nodes and the tree itself are socially constructed, shared structures that are fully situated and available to members. In particular, members talk about global properties of this tree, including its internal nodes and its edges, and the number of each; they also have names for these concepts. This example is the annual regatta at Henley, in which nodes represent boats (i.e., "crews"), internal nodes represent winners of races (i.e., "heats"), and edges point to the participants in a race. Figures 3 and 4 show part of an initial "draw" for one such race[11]. The number of internal nodes is the total number of races, and the number of edges is the number of instances of crews racing. This formal structure is situated in the events of a particular actual regatta, and it is determined on a local, *ad hoc*, contingent basis. In particular, the internal variables $(w_1, w_2, ...)$ of the draw become instantiated by concrete situated events that unfold in actual time, until the whole race is summarised by a single structure that tells what happened, including which crews raced in each heat, and who won. Note that the structure may change during an instance of the regatta; for example, a crew can be disqualified, resulting in fewer heats than originally scheduled.

[11] Toulmin [33] presents his data in the form shown in Figure 3, saying that it comes from "the sports page of a Sunday paper," presumably in the mid-1950s.

Members are aware that the same structure (i.e., "race") can be presented in a variety of ways; for example, it can be represented by a table in a newspaper as in Figure 3, by a verbally presented list, or by the computer scientists' traditional representation[12], shown in Figure 4. Hence, there is a precise structure here that is independent of how it happens to be represented; this means that we have an *abstract data type*, in the precise mathematical sense of [11], which is an isomorphism class of computable many-sorted algebras[13]. Of course, this does not mean that members are familiar with this mathematical concept; even Toulmin only uses concrete representations.

5.2 Discussion

These examples, particularly the second one, show that "dry" structures occur "in nature," that is, in ordinary social interaction, in the sense that members of ordinary groups (such as sports fans) organise their talk in ways that correspond to such structures. These structures can be printed in newspapers or shown on television, in a compact graphical form. Nevertheless, the structures are still recognisably *situated*, that is, locally organised, contingent and *ad hoc*, and they attain a sort of immutability only in retrospect. In the case of a value system tree, the analysts are socially responsible for some parts of the structure, and the members of the organisation for other parts; the tree itself is only meaningful to members of the analyst culture.

These observations seem to offer some hope of making further progress in Requirements Engineering if we recognise the situated nature of the structures involved, and indeed, of the whole requirements process. This means that we must effect an ongoing, practical reconciliation of "the dry" and "the wet" in the *practice* of Requirements Engineering. Nor should we ignore the power of "hacking" code, or of deep theoretical reflection on the mathematical properties of certain structures; for example the hyper-requirements project described in Section 3 attempts to reconcile all of these aspects.

6 Summary

This section collects some points from the body of the paper that may be especially provocative:

1. Requirements Engineering is a very challenging and relatively little explored field, so that any conclusions are necessarily somewhat tentative.

2. It seems likely that approaches quite different from the traditional systems analyses methods will be needed, in order to take account of the social context in which computing systems are used.

3. The examples in Section 5.1 suggest that the social nature of information structures is a complex problem that may require sophisticated analysis. It seems that neither

[12]Note that this representation differs from trees on earth, which have their roots at the bottom.

[13]The different algebras in the class correspond to the different representations of the same abstract structure; see [11] for details.

a purely formal (i.e., dry) nor a purely social (i.e., wet) approach will suffice, and that an effective approach to systems development must reconcile the dry and the wet.

4. Structures as dry as abstract data types occur "in nature", that is, in the naturally occurring discourse of ordinary social groups, such as sports fans.

5. On the other hand, requirements are emergent, in that they arise from interactions between analysts and members of the client organisation. Moveover, requirements change as the organisation does, and sometimes the requirements process itself can have a significant effect on an organisation.

6. Much of what analysts do is not consistent with ethnomethodology; analysts' methods are often *not* members' methods. Nor should they be.

7. In particular, analysts construct what Latour [19] calls "immutable mobiles" in order to support positions in agonistic encounters, such as meetings with managers, or peers.

8. Computer systems are increasingly repositories for immutable mobiles, and increasingly provide the means for producing new immutable mobiles, and for transporting them into new contexts. Hence, computer systems can be powerful engines for concentrating and applying power.

9. Thus, requirements analysis for such systems is a natural occasion for power struggles, and it is important that the human interests of all stakeholders be recognised and protected. The failure to do so explains why many large systems fail to work well in practice.

10. In fact, to be successful, a computer system may need to serve as what Star [31] calls a "boundary object" which intermediates the needs of different social communities.

11. The relationship between the dry and the wet is *complex*, in that there are many different facets, which arise in different contexts and at different levels of abstraction.

12. The relationship between the dry and the wet is not one of antagonism, in which one must be wrong and the other right; rather, these two aspects of data and its interpretation are *complementary*, and are both needed for successful Requirements Engineering.

References

[1] Russell Abbott. Program design by informal English descriptions. *Communications of the Association for Computing Machinery*, 26(11):882–894, 1983.

[2] Barry Boehm. *Software Engineering Economics*. Prentice-Hall, 1981.

[3] Alan M. Davis. *Software Requirements: Analysis & Specification.* Prentice-Hall, 1990.

[4] Harold Garfinkel. *Studies in Ethnomethodology.* Prentice-Hall, 1967.

[5] Joseph Goguen. Hyperprogramming: A formal approach to software environments. In *Proceedings, Symposium on Formal Approaches to Software Environment Technology.* Joint System Development Corporation, Tokyo, Japan, January 1990.

[6] Joseph Goguen and Charlotte Linde. Cost-benefit analysis of a proposed computer system. Technical report, Structural Semantics, 1978.

[7] Joseph Goguen and Charlotte Linde. Structural semantic analysis of the information structure of organizations. Technical report, Structural Semantics, 1981.

[8] Joseph Goguen and Charlotte Linde. Linguistic methodology for the analysis of aviation accidents. Technical report, Structural Semantics, December 1983. NASA Contractor Report 3741, Ames Research Center.

[9] Joseph Goguen and Charlotte Linde. Techniques for requirements elicitation. Technical report, Centre for Requirements and Foundations, Oxford University Computing Lab, 1992. To appear, *Requirements Engineering '93.*

[10] Joseph Goguen, Charlotte Linde, and Miles Murphy. Crew communication as a factor in aviation accidents. In E. James Hartzell and Sandra Hart, editors, *Papers from the 20th Annual Conference on Manual Control.* NASA Ames Research Center, 1984.

[11] Joseph Goguen, James Thatcher, and Eric Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. Technical Report RC 6487, IBM T.J. Watson Research Center, October 1976. In *Current Trends in Programming Methodology, IV*, Raymond Yeh, editor, Prentice-Hall, 1978, pages 80–149.

[12] Joseph Goguen, James Weiner, and Charlotte Linde. Reasoning and natural explanation. *International Journal of Man-Machine Studies*, 19:521–559, 1983.

[13] Charles Goodwin and John Heritage. Conversation analysis. *Annual Review of Anthropology*, 19:283–307, 1990.

[14] Michael Jackson. *Principles of Program Design.* Academic, 1975.

[15] Marina Jirotka. Ethnomethodology and requirements engineering. Technical report, Centre for Requirements and Foundations, Oxford University Computing Lab, 1991.

[16] Jiro Kawakita. *KJ Method: a Scientific Approach to Problem Solving.* Kawakita Research Institute, 1975.

[17] Adam Kendon. *Conducting Interaction: Patterns of Behavior in Focused Encounters*. Cambridge University, 1990. Studies in Interactional Sociolinguistics Number 7.

[18] William Labov. The transformation of experience in narrative syntax. In *Language in the Inner City*, pages 354–396. University of Pennsylvania, 1972.

[19] Bruno Latour. Visualization and cognition: Thinking with eyes and hands. *Knowledge and Society: Studies in the Sociology of Culture Past and Present*, 6:1–40, 1986.

[20] Claude Lévi-Strauss. *The Raw and the Cooked*. Penguin, 1964. Translation by John and Doreen Weightman, 1986.

[21] Steven Levinson. *Pragmatics*. Cambridge University, 1983.

[22] Charlotte Linde and Joseph Goguen. Structure of planning discourse. *Journal of Social and Biological Structures*, 1:219–251, 1978.

[23] Jean-Francois Lyotard. *The Postmodern Condition: a Report on Knowledge*. Manchester University, 1984. Theory and History of Literature, Volume 10.

[24] Tom De Marco. *Structured Analysis and System Specification*. Yourdon, 1978.

[25] Carroll Morgan. *Programming from Specifications*. Prentice Hall, 1990.

[26] Christopher Nix and Peter Collins. The use of software engineering, including the Z notation, in the development of CICS. *Quality Assurance*, 14(3):103–110, September 1988.

[27] U.S. Government Accounting Office. Contracting for computer software development – serious problems require management attention to avoid wasting additional millions. Technical Report FFGMSD-80-4, U.S. Government Accounting Office, November 1979.

[28] Roger Penrose. *The Emperor's New Mind*. Oxford, 1989. Vintage paperback edition, 1990.

[29] Harvey Sacks. On doing 'being ordinary'. In J.M. Atkinson and John Heritage, editors, *Structures of Social Action: Studies in Conversation Analysis*. Cambridge University, 1984. Original date, 1971.

[30] Motoshi Saeki, Hisayuki Horai, Katsuyasu Toyama, Naoya Uematsu, and Hajime Enomoto. Specification framework based on natural language. In *Proceedings of the Fourth International Workshop on Software Specification and Design*, pages 87–94. IEEE, 1987.

[31] Susan Leigh Star. The structure of ill-structured solutions: heterogeneous problem-solving, boundary objects and distributed artificial intelligence. In Michael Huhns and Les Gasser, editors, *Distributed Artificial Intelligence*, volume 3, pages 37–54. Morgan Kauffmann, 1988.

[32] Lucy Suchman. *Plans and Situated Actions: The Problem of Human-machine Communication.* Cambridge University, 1987.

[33] Stephen Toulmin. *The Uses of Argument.* Cambridge University, 1958.

[34] Edward Yourdon. *Modern Structured Analysis.* Prentice-Hall, 1989.

Contents