Requirements Engineering as the Reconciliation of Technical and Social Issues*

Joseph A. Goguen†

Centre for Requirements and Foundations, Programming Research Group
Oxford University Computing Lab, Oxford OX1 3QD, United Kingdom

1 Introduction

Much of the information that requirements engineers[1] need is embedded in the social worlds of users and managers, and is extracted through interactions with these people, e.g., through interviews and questionnaires. At its source, this information tends to be informal and highly dependent on its social context for interpretation. On the other hand, many representations that appear in constructing and using computer-based systems are formal, in that they are defined by the formal syntactic and semantic rules of computers and computer languages, so that their interpretation is relatively independent of social context. We suggest that both the formal, context insensitive, and the informal, socially situated aspects of information are crucial to the success of Requirements Engineering projects. Elsewhere, we have called these two aspects "the dry" and "the wet," respectively [14]. Here, we suggest that Requirements Engineering has a strong practical need to reconcile them, and that this kind of reconciliation may be the very essence of Requirements Engineering. To this end, we suggest a new notion, the "situated abstract data type," which joins the formal and informal aspects of information.

The need for progress in Requirements Engineering is acute: large projects have a embarrassingly high failure rate[2]. Moreover, the requirements phase of a large system development project is the most error-prone, and these errors are the most expensive to correct [5, 9]. Consequently, this phase has the greatest economic leverage. Unfortunately, it is also the least explored, and has the least satisfactory intellectual foundations. Therefore it seems a promising area in which to invest research effort. The fact that the social issues at the root of many difficulties cannot be modelled by the usual technical methods suggests that novel approaches will be needed. Moreover, the immaturity of the field suggests that a willingness to be eclectic rather than dogmatic could be valuable. Consequently, we are exploring techniques from a variety of fields, especially Ethnomethodology and Discourse Analysis, for analysing written

*The research reported in this chapter has been supported in part by a contract from British Telecommunications plc, grants from the Science and Engineering Research Council and Fujitsu Laboratories, and a contract under the management of the Information Technology Promotion Agency (IPA), Japan, as part of the Industrial Science and Technology Frontier Program "New Models for Software Architectures," sponsored by NEDO (New Energy and Industrial Technology Development Organization).

†Written in part while on sabbatical at the Technical University of Nova Scotia, Halifax, Nova Scotia, Canada, with partial support from the Nova Scotia provincial government.

[1] A more traditional name for this rôle is "systems analyst," but we intend the name "requirements engineer" to suggest an increasing importance, responsibility, and training for this profession.

[2] A 1979 study by the U.S. Government Accounting Office [36] reported that 95% of funds spent in a sample of nine projects totaling nearly 7 million dollars did not result in a system that was actually used. Because the large organisations that commission large systems are understandably reluctant to release embarrassing information, it is difficult to obtain current data on this problem. However, there are indications that things may be somewhat better now, though still very far from ideal.

texts, spoken language, and natural interactions in their social context. Section 2 reviews some of these techniques. Also, we use ideas from Computing Science and the Sociology of Science to help understand the formal aspect of Requirements Engineering. One outcome is some initial steps towards a social theory of information, as described in Section 2; another outcome is the situated abstract data type notion.

Sections 3, 4, 5 and 6 give an examples to illustrate our approach. These are: the classification of system construction activities into so-called lifecycle “phases”; a taxonomy for Requirements Engineering methods, using some ideas of Jean-François Lyotard on postmodernism; a value system for an organisation using some research on the structure of stories due to William Labov; and a conventional structure for a regatta, building on some work of Stephen Toulmin. Section 6 explains what abstract data types are, how they are specified, and what we mean by their being socially situated. A formal specification of the regatta abstract data type is given in the appendix. Some similarities and differences between these four examples are discussed in Section 7, explicating how some of their formal and informal aspects are related. We then draw some implications for Requirements Engineering.

Acknowledgements

I thank the members of the Centre for Requirements and Foundations for many stimulating conversations on the topics discussed here, particularly Marina Jirotka and Matthew Bickerton. I also wish to thank Dr. Susan Leigh Star for several interesting and helpful conversations. Thanks to Frances Page for preparing the figures, and to André Stern for some interesting remarks about time and requirements. Finally, special thanks to Dr. Charlotte Linde for our long collaboration, during which I learned much of what I know about discourse analysis.

2 Techniques for Requirements Elicitation

It can be difficult to find good data on which to base requirements. Experience shows that simply asking managers what they want often works poorly. They do not (usually) know what is technologically feasible, and they cannot accurately describe what their workers really do, what their clients really do, or even what they really do. This is not because managers are incompetent; on the contrary, they are (usually) genuine experts at their own job. Rather, it is due to what philosophers [39] call the problem of *tacit knowledge*, i.e., the phenomenon that people may know how to do something, without being able to articulate how they do it. Examples include riding bicycles, speaking languages, negotiating contracts, reconciling personal differences, and evaluating employees. One important reason for this difficulty is the social situatedness of the information involved.

However, in order to build a system that effectively meets a real business need, it may be necessary to find out what workers, clients and managers really do. But just asking workers what they do is subject to the same problems as asking their managers. Instead, if we really need this information, it would be best to go where the work is done, and carefully observe what actually happens. Various techniques from sociology and sociolinguistics seem promising for this purpose, and some of these are discussed later in this section; a more detailed exposition of these techniques with further comparison is given in [19].

2.1 Ethnomethodology

Traditional sociology has been much influenced by what it considers to be orthodox science, where a scientist first formulates a theory, on the basis of which he makes predictions, which he then tests empirically. The aim is to achieve *objectivity*, in the sense that the desires and biases of the scientist cannot affect the conclusions. Hence, there is a rigid separation between subject and object, between observer and observed. Physics has already moved rather far from this kind of objectivity[3], and so it should not be surprising if sociology, and the social aspects of computing, had to go even further. In particular, if objective information is replaced by situated information, then the orthodox approach of formulating and then testing hypotheses objectively, for example through statistical sampling, will not be valid, because the random events observed can no longer be assumed to be statistically independent. However, statistical methods are the foundation for much of traditional sociology, for example, the design and evaluation of questionnaires. I do not suggest that statistics and questionnaires are never useful, only that they are *not always valid*, and in particular, that they should not be used in situations where context plays a significant rôle. (See [19] for a more detailed discussion of questionnaires and interviews.)

Ethnomethodology [11] can be seen as a reaction against the "scientific" methodology of traditional sociology. (More detailed discussion of ethnomethodology and science may be found in [45] and [46]). Ethnomethodology tries to reconcile a radical empiricism with the situatedness of social data, by looking closely at how competent members of a group actually organise their behaviour, and in particular, at the *categories* and *methods* that they use to render their actions intelligible to one another; this contrasts with presupposing that the categories and methods of the analyst are necessarily superior to those of members. The methods and categories of members are identifiable through the procedures by which members are held socially accountable by other members of their group. I like to describe this by saying that the analyst is used as a measuring instrument. Through experience, such an analyst gradually learns to pay attention to doubts and hints of associations, and to follow them up with further questions. Through immersion in data from some particular social group (such as stock brokers), the particular competencies are gradually acquired that enable the analyst to be a sensitive and effective instrument in that domain. In this way, subjectivity is harnessed rather than rejected.

One basic principle underlying Ethnomethodology is that members are held *accountable* for certain actions by their social groups; moreover, exactly those actions are considered socially significant by those groups. A member performing such an action is always liable to be asked for an account, that is, a justification[4]. Let us call this the *principle of accountability.* From this follows the *principle of orderliness*, that social interaction is *orderly*, in the sense that it can be understood. This follows from the fact that the participants themselves understand it, because of accountability; therefore analysts should also be able to understand it, if they can discover the methods and categories that members themselves use to make sense of their interactions.

It is time to be more precise about what we mean by saying that social interaction is *situated*: we mean that the events that occur in some interaction can only be fully understood in relation to the particular, concrete situation in which they actually occur. The following list of *qualities of situatedness* (inspired in part by work of Lucy Suchman [48] on plans) may help to further clarify this point:

1. *Emergent.* This refers to the claim that social events cannot be understood at the level of the individual, that is, in terms of individual psychology, because they are jointly constructed as social events by the members of some group through their on-going interaction.

2. *Local.* This refers to the claim that actions and their interpretations are constructed in

[3] Penrose [38] gives an elegant and readable exposition that illustrates just how strange the theories of contemporary physics can be.

[4] Of course, this does not imply that such accounts are always, or even usually, requested by members of the group, or that they are necessarily given when requested.

some particular context, including a particular time and place.

3. *Contingent.* This refers to the claim that the construction and interpretation of events depends upon the current situation (potentially including the current interpretation of prior events). In particular, interpretations are subject to negotiation, and relevant rules are interpreted locally, and can even be modified locally.

4. *Embodied.* This refers to the claim that actions are linked to bodies that have particular physical contexts, and that the particular way that bodies are embedded in a context may be essential to the social interpretation of some events.

5. *Open.* This refers to the claim that theories of social events (both those constructed by participants and by analysts) cannot in general be given a final and complete form, but must remain open to revision in the light of further analyses and further events.

6. *Vague.* This refers to the claim that practical information is only elaborated to the degree that it is useful to do so; the rest is left grounded in tacit knowledge. (This can be seen as a corollary of openness.)

I make no claims for the completeness of this list, let alone its superiority. On the contrary, it derives its plausibility from its general similarity to many other such lists. For example, qualities like these are familiar to anthropologists (e.g., see various comments in Lévi-Strauss [30]); however, anthropologists have not (to my knowledge) been very precise in distinguishing among different qualities. Also, Suchman [48] gives a rather similar list of qualities for plans.

Unfortunately, Ethnomethodology can be difficult to understand; however, relatively comprehensible expositions of certain points have been given by [25], [31], [48] and [19]. 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 (see [41, 42]). Interaction Analysis extends Conversation Analysis to include video data, particularly in institutional settings. See [24] for a recent overview of Conversation Analysis, and [27] for a collection of essays on and applications of Interaction Analysis. All these fields are strongly empirically based. Typical projects can involve hundreds of hours of work in recording, transcribing and analysing data. In these approaches, it is important to use "naturally occurring" data, collected in a situation where members are engaged in activities that they regularly and ordinarily do; otherwise, the basic principle of accountability will not apply, and we cannot be sure that events in the data have any natural social significance. For example, data collected from interviews cannot be used (unless one wants to study what happens in interviews!).

Although it is not usual in the literature of ethnomethodology, a distinction between the methods and categories used by members and those used by analysts can help to clarify the social status of requirements objects. Analysts often use categories and methods that members of the group they are studying do not use. For example, analysts of a telecommunications system may want to accumulate statistics on calls, even though these statistics would be incomprehensible to most of those making the calls. In fact, analysts form groups that have their own distinct cultures, with their own categories and methods, and it is necessary to evaluate their actions from this perspective. Nevertheless, analysts can often benefit from knowing the methods and categories of members, particularly when they want to understand something that members regularly and ordinarily do themselves, as is often the case with Requirements Engineering.

I believe that Ethnomethodology provides useful general guidelines for how to collect high quality data about social interaction, and that Conversation and Interaction Analyses embody these guidelines in ways that are directly applicable to many problems in Requirements Engineering. However, it is far from the only way to elicit requirements, and it is not the best method for all circumstances. (Further discussion of these issues, with examples, may be found in [19].)

2.2 Discourse Analysis

Although natural language is often criticised, e.g., by advocates of formal methods, 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 the conflicts and ambiguities that may arise from the initial situation; it is important not to prejudge the many tradeoffs that will have to be explored later on, such as cost versus almost everything else, including speed and functionality. Also, natural language, possibly supplemented by graphics, is often the medium preferred by the individuals who represent the client. And finally, natural language can permit the "diplomatic" resolution of conflicts through the careful construction of deliberate ambiguities; for example, this is rather common in large government financed projects.

There is a growing body of evidence that natural language is far more structured than most people realise, and that discourse structure carries much important information about what is being described. For example, work by Abbott [1], and by Enomoto, Horai and others [43] shows that the nouns and verbs used in stating requirements can 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 indicate relationships of inheritance and clustering. However, one cannot expect to find mechanical algorithms that will always do such analyses reliably, in part because of the importance of context.

According to the principle of accountability, a member of some group who tells an informal story must establish to the audience the relevance of the actions reported. The classic case, studied by William Labov [28], is the narrative of personal experience, where the narrator is an agent in the story. The most typical way to accomplish accountability is to include specific *evaluative material* within the body of the story that relates the *narrative material* to shared social values. It may be surprising that values are an integral part of the internal structure of stories, rather than being confined to an optional summary "moral" at the end. But in fact, naturally occurring stories embody evaluative material in many complex and subtle ways, and its syntactic placement can be a significant clue to its importance.

To further explain and illustrate these ideas, let us analyse a simple story. The nursery rhyme that we use is not, strictly speaking, a naturally occurring spontaneous story, let alone a narrative of personal experience, as studied by Labov [28]. However, it is often read, or repeated from memory, to children in natural social settings, and thus an analysis of the values in it should tell us something about what our society teaches its children. Our analysis will be somewhat sketchy, omitting many details of argument; otherwise, it could be rather tedious to read. Here is the text:

> Jack and Jill went up the hill
>   to fetch a pail of water;
> Jack fell down and broke his crown,
>   and Jill came tumbling after.
>
> Up Jack got, and home did trot,
>   as fast as he could caper,

Jill put him to bed and plastered his head
with vinegar and brown paper.

(The second verse is one among several variations; see [37] for this and other background information.) The first line is a straightforward narrative clause, recounting an action in the narrative past tense, while the second line is an evaluative clause, giving a reason for the action of the first clause. The third and fourth lines give further narrative clauses (there are two in the third line).

A very basic feature of narrative is that the order of narrative clauses is taken as the order of the events that they report, unless some trouble is taken to indicate otherwise. Thus, in the first verse, they first went up, then Jack fell down, then he broke his crown, and then Jill came tumbling after. This basic principle is called the *narrative presupposition*. Note that it is a convention, and *not* a necessary feature of narratives; for example, Becker [3] shows that in Balinese narratives, if no special care is taken then the events reported in a sequence of narrative clauses are taken as occurring *simultaneously* rather than sequentially[5].

Now that we know ordering is significant in English narratives, it is interesting to notice that Jack always comes before Jill. As far as the semantics is concerned, this ordering would not matter in the first line, but because it is part of a general pattern, we can consider it to be an evaluative feature of the narrative. (Note the delicacy, and not quite water-tight quality of this argument; rigorous proofs are impossible in this area.)

What can we now conclude? I think we may conclude that water is important to this (somewhat mythical) culture, and that males are more important than females in it.

This need not be the end of the analysis (although further elaboration might push the limits of tedium): one could get some further results by using the so called *causal presupposition*, which says that, other things being equal, given clauses in the order A, B, we may assume that A causes B. (For example, "You touch that, you gonna die.") The reader may wish to follow up this remark as an exercise. As before, it is not so much a matter of *proving* something, or of *extracting the truth*, as it is of uncovering a resonance of the text with a possible interpretation. Any such analysis is contingent, local and open; moreover, it is best done in a group, so that the analyst is accountable to other analysts, in which case the analysis itself becomes emergent and embodied at that level. Nonetheless, any such interpretation can be considered to be some part of the meaning of the text; of course, each interpretation will seem more cogent to some analysts and groups than others, and some may seem dubious to most.

Work by Goguen and Linde [16, 17] shows that information extracted from naturally occurring instructional discourse can often be translated without much difficulty into dataflow diagrams of the kind used in structured design methodologies, such as those of De Marco [35] and Yourdon [51]; nevertheless, there are questions about how such translations can be justified. Other research by Goguen and Linde [32, 22, 18, 20] shows that plans, explanations, directions, and other everyday types of discourse have a high level structure that relates directly to their social and semantic domains. What is called "command and control" discourse in [18] seems especially relevant to Requirements Engineering, because its analysis can be used to reveal the structure of tasks unfolding in real time. Unfortunately, the data on which these structural theories of large grain discourse were based was not collected with the rigour of Ethnomethodology (see [25]); nevertheless, we hope that a more careful ethnomethodological study might support at least some aspects of such theories.

[5] A Computing Scientist might say that the default connective for a narrative sequence in English is sequential composition (";"), whereas in Balinese it is parallel composition ("||").

2.3 Towards a Social Theory of Information

If we knew more about what information is, and how it is used, then we might be in a better position to improve the practice of Requirements Engineering. This subsection presents what may be some initial steps towards a social theory of information. Objective and quantitative theories of information, such as the statistical theory of Shannon [44], as well as objective but qualitative theories, such as the situation semantics of Barwise and Perry [2], are not adequate for our purpose. Although each may illuminate certain aspects of information, neither considers the social processes that create and sustain information through interpretation and negotiation. Shannon's theory is unable to take account of the meaning of information, even in the narrowest technical sense, and neither theory can explain how information can vary in its degree of context dependence, while remaining thoroughly situated. Discourse Analysis provides one approach to such questions; this section discusses some other approaches that go even further in reconceptualising context.

2.3.1 Sociology of Science

The study of the Sociology of Science is an important new development. One exciting voice in this field is Bruno Latour, who has identified certain properties that distinguish the work of scientists from other kinds of work [29]. Latour introduced the phrase *immutable mobile* for a representation that can be interpreted in (what counts as) the same way in a variety of contexts; thus, immutable mobiles are information structures that have been (at least partially) dried out. These structures are frequently what Latour calls "*re-representations*," which are representations that have undergone some form of concentration; for example, a large set of observations of planetary motion might be summarised by a single equation. Latour claims that the qualities of immutability, mobility, and concentration are characteristic of the information that occurs in the discourse of science.

It would be interesting to take a similar approach to the study of information more generally, viewing Latour's work as exploring the special case of scientific information; this is because Computing Science must deal with information from a wide variety of fields other than science, such as commerce, government, law and literature, e.g. in designing information systems. It also seems promising to apply techniques from Ethnomethodology to the information in computer-based systems. Work that may help point the way has been done by Eric Livingston [33], showing the lived work involved in mathematical proofs. There are also ethnomethodological studies of several other aspects of science, and such an approach might help provide a richer empirical foundation for research in the Sociology of Science.

2.3.2 Dry and Wet Information

I suggest that *information* is distinguished from *mere signs* by the existence of a social group that can be held accountable for its appropriateness[6]. Signs, in the sense of semiotics, are configurations that do not necessarily have significance, so that "mere signs" have no significance. But this is only a theoretical possibility, because the very notion of sign as configuration presupposes that a classification system for configurations has already been pre-selected (e.g., a certain character set), and because this classification system must be known to someone, it will impart to its signs at least the significance of its being known to be part of this system.

We can further distinguish information that can be understood in a wide variety of contexts from information that is so thoroughly situated that it cannot be understood except in relation

[6]This is intended as a distinction that may be useful in practice, rather than as a metaphysical assertion about the nature of information.

to very particular contexts. We call these types of information *dry* and *wet*, respectively, following [14]. Note that there is really a continuum of intermediate cases, e.g., "damp" information, such as cookery recipes. In general, information cannot be fully context sensitive (for then it could only be understood when and where it is produced) nor fully context insensitive (for then it could be understood by anyone at any time). A fairly extreme case is the "raw data" collected in a scientific experiment; although it may be just a collection of numbers, it is very highly situated, because those numbers only make sense to a very small group who share a very particular context. On the other hand, an equation that summarises the numbers is relatively more dry, and a physical law is even drier. It is important to notice that information, however dry, must still be interpreted in some local context. Therefore, the qualities of situatedness in Section 2.1 apply to information; that is, information is always emergent, contingent, local, embodied, open and vague. This has important implications for Requirements Engineering, in suggesting what we can reasonably expect from requirements documents, as opposed to the unattainable ideals of positivist philosophers.

These considerations suggest that formalisation may be a key technique for making information drier. Let us call this the *formalisation hypothesis*; several illustrations of it are given in the following sections. We can also relate this point to the immutable mobiles described by Bruno Latour [29], which are distinguished by the possibility of their being interpreted in what counts as the same way for practical purposes in a variety of contexts. Latour discusses the example of cartographic maps: given the proper instruments, and good weather, such maps can be used anywhere in the world; but each use is still a local interpretation. As Latour notes, immutable mobiles are often constructed by concentrating previously available information; this is a form of abstraction. But I wish to point out that there are other forms of abstraction; for example, a database can provide easy access to a large body of information by providing keys, without actually compressing it in any way.

Latour also notes that the construction of immutable mobiles can be a way to mobilise support, by compressing large amounts of information into simple graphical representations; then anyone who wishes to disagree must mobilise the resources to represent and compress comparable amounts of information. The use of dataflow diagrams in requirements meetings seems to provide good examples of this phenonmenon. In general, the use of dataflow diagrams is beyond the capabilities of users and managers, in part due to the huge volume of information involved in large projects. Moreover, Requirements Engineering has recently developed special tools to help collect and organise large volumes of requirements information. One class of tools organises information about designs on selected requirements issues. For example, the gIBIS tool [8] is organised around issues, positions and arguments. These tools are based on normatively given presuppositions about the way that projects should be organised. Moreover, it will be difficult for ordinary users and managers to use such tools themselves.

Information systems seem a particularly interesting site for research. Such systems are repositories for immutable mobiles, and increasingly they provide the means for producing new immutable mobiles, for transporting them into new contexts, and for further concentrating and summarising information. This means that they can be sources of 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 should be recognised and protected; Boland [7] describes an interesting case study that illustrates the importance of power struggles in understanding organisations. The failure to take account of such factors explains why many large information systems have not worked well in practice.

Leigh Star [47] has introduced the term "boundary object" to describe objects that are used in different ways by different social groups. For an information system to be successful, it must often serve different social groups in different ways, and so I suggest that research on boundary objects might be usefully applied to information systems. For example, the notion of a *view* in that field already reflects the idea that it may be useful to present information in different ways to different users. Similarly, a requirements document must serve a number of different stakeholders, and thus may need to incorporate multiple views [10]. The work of Latour is also very relevant to these issues.

2.3.3 Interpretation versus Representation

It is perhaps worth digressing briefly to contrast the view of information and meaning that has been suggested above with the *representational theory of meaning* that is considered more or less standard in Computing Science and more generally in the Anglo-American analytic tradition of philosophy.

According to our social theory of information, meaning is the on-going achievement of some social group; it takes *work* to interpret some signs so that they can be seen as information, and interpretation necessarily occurs in some particular time and place. By contrast, a representational theory of meaning claims that a meaningful configuration of signs *represents* something in the real world. In sophisticed representational theories, such as situation semantics [2], what is represented by (say) a given phrase of English can vary with context, and need not be a simple object, but can be a complex relationship, such as a situation. This may be quite adequate for some purposes. However, even the most sophisticated representational theory leaves out the work of interpretation and the social accountability that goes with it.

2.4 Case Studies

The Centre for Requirements and Foundations at Oxford is now involved in some case studies to test the practical application of techniques of Ethnomethodology and Interaction Analysis to Requirements Engineering. One project, led by Marina Jirotka, involves the analysis of video tapes of stock brokers at work, supplemented by ethnographic interviews, including feedback from the brokers on our interpretations of selected video clips. Another project, led by Matthew Bickerton, may involve videotaping interviews conducted by requirements engineers, as well as their internal working sessions, with a view towards explicating their work practices, and suggesting ways to improve requirements elicitation and analysis; the target domain here is software for multimedia communication.

Sections 3, 4, 5 and 6 give examples that illustrate some of the issues that arise in trying to reconcile the formal context insensitive with the situated social aspects of information. Although each example organises some data into what computing scientists call a tree structure, each structure is situated in its social context in a different way, as discussed in section 7.

3 The Situated Lifecycle

The *lifecycle* of a system is often considered to have a number of "*phases*." There is no universal agreement on what these are, and in fact, their number, names and boundaries are somewhat vague and arbitrary. However, it seems that activities of the following kinds may be roughly distinguished with some consistency:

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

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

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.

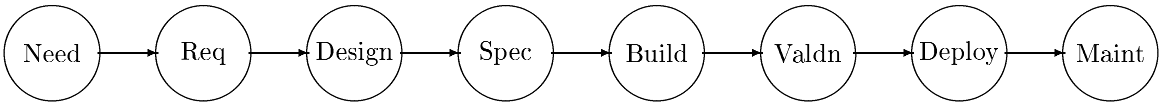

Figure 1: Information Flow in the Waterfall Model

In the so-called *waterfall model*, these (or similar) phases must be enacted in exactly the indicated order, as shown in Figure 1. However, experience with real projects shows that there is no such orderly progression from one phase to the next; instead, there is a continual projection forward and backward. For example, the client may perceive a new need (or reassess an old one), or an implementer may perceive a new opportunity (or impossibility); also, prototypes can suggest changes in requirements, specifications, etc. In fact, at least in successful large projects, the phases are constantly overlapping, and many actions and events are difficult to assign to phases. However, this is precisely what Suchman's work on plans [48] should lead us to expect, because lifecycle models are actually plans. As Suchman points out, plans serve as resources for action, rather than as rigid programs for future action. Hence, actual lifecycles are contingent, emergent, local and open, and so their relationship to models of the software process will necessarily be problematic. But we might also note that because they abstract relevant prior experience, life cycle models are also (partially) immutable mobiles.

The waterfall model has been widely criticised, and many so-called *process models* have been proposed as alternatives. Even so, the nature and limitations of such models do not seem to have been widely appreciated. For example, they still assume a division into phases. However, as the paper by Button and Sharrock in this volume shows, a real project in a commercial environment may well have to deliver code before completing requirements. Also, McDermid's paper argues that some high level design is often needed in defining requirements. It follows that rigidly following a fixed process model can severely limit adaptation. One very pernicious factor is procurement procedures that try to rigidly separate phases with contractual barriers in the name of competition; this approach fails to provide system developers with the on-going access to users that is often necessary for success.

The belief that lifecycle phases should be enacted sequentially follows from the myths that there is a more or less unique best system to be built, and that its construction can be controlled in a top-down hierarchical manner ([13] discusses these and some related myths of system development).

Similarly, in the requirements phase, it seems to be widely believed that there is some unique best model of the organisation and its needs. However, requirements are *emergent*, in the sense that they do not already exist in the minds of clients or requirements engineers (or anywhere else); instead, they gradually emerge from interactions between requirements engineers and the client organisation. Moreover, requirements are *open*: they are always subject to change, because organisations and their contexts continually change. Requirements are also *local*, in that requirements documents must be interpreted in the context of a particular organisation at a particular time. (For example, I conjecture that if you have not yet read the background material in Section 6 and the appendix, then you will have difficulty in understanding Figure 6 on page 18.) Moreover, they are *contingent*, because they are an evolving outcome of an on-going process that builds on prior interactions and documents.

It is interesting to note that most of the effort for typical large systems goes into the maintenance phase. Some advocates of formal methods have argued that this is because not enough effort has been put into being precise in earlier phases, particularly specification. However, I believe a deeper reason is that much more is going on during so-called maintenance than is generally realised. In particular, reassessment and re-doing of requirements, specification, and code, as well as documentation and validation, are very much a part of maintenance; thus, the maintenance phase may contain smaller versions of the complete lifecycle. It therefore seems likely that problems with requirements will be a significant part of the maintenance cost, so that methods that help with requirements will also help with maintenance. Similar considerations apply to other phases, but perhaps less dramatically.

3.1 The Principles of Volume and Inertia

Large systems seem to raise special problems because of the huge volume of information that they can produce, sometimes literally roomfuls. Consequently, special attention must be paid to the management of this information. Let us call this the *principle of volume.* Actually, it can be argued that no genuinely new problems arise for large systems, but rather it is merely necessary to devote considerable attention to issues that are trivial for small systems. These issues include version and configuration management for requirements, specifications, code, documentation, process requirements, standards, test data, financial data, as well as appropriate links among these.

A closely related principle may be called the *principle of inertia.* It says that the larger the volume of information involved in a system development effort, the more difficult it will be to make changes. For example, this principle can explain why large systems are often already out of date when they are delivered, why large organisations usually evolve slowly, and also why languages evolve slowly.

Perhaps it is misleading to dignify these observations with the title "principle" because they are really only heuristics that may, in some circumstances, help in formulating an initial understanding of certain situations. Actually, this applies, to a greater or lesser degree, to all of the principles that we have formulated: they are intended as guidelines for practical action, rather than as metaphysical pronouncements about the nature of reality, or the nature of research. Notice that the proponents of particular methods may take a very different attitude to the principles that they claim underlie their methods.

3.2 The Retrospective Character of Requirements

I wish to claim that the classification of events and objects into lifecycle phases makes the most sense retrospectively: from the viewpoint of some relatively stable state of a given system, one can determine the needs that it serves, the requirements that it meets, the design that it realises, etc. In general, such a retrospective reconstruction will have to delete or revise some of the original requirements, and add some new ones, in order to achieve the *traceability* of decisions back to some coherent basis in requirements objects. That is, a lifecycle only makes sense retrospectively, because only when knows one how the system works in its context can one tell whether particular events are important or trivial, successful or disastrous. In many projects, the work required to achieve a retrospective reconstruction is never done, and by the time the system is deployed (if it is), all that remains of the requirements process may be some colourful stories. (This is explained by the principle of volume.)

Another basic principle of a Social Theory of Information may be an extension of Suchman's work on plans [48] to the broader claim that only our *post hoc* explanations for situated events appear to attain relative stability and independence from context; let us call this the *retrospective hypothesis*. Support for this principle can be found in the empical work on plans and explanations reported in [32] and [22], as well as in the work of Suchman.

In particular, the interpretation of requirements, indicating whether or not they have been successfully met, is the outcome of a complex social process that typically involves negotiation and retrospective reconstruction; it may even involve legal action. Therefore, in many situations it can be quite misleading to think of requirements as pre-given; but of course, this does not hold for all situations. More radically, it may sometimes be useful to think that causal sequences, and in this sense, even time itself, are determined retrospectively, by the interpretations that we put upon events.

Software Engineering has developed process models that are more flexible than the waterfall model, such as the "spiral" models of Boehm [6]. However, we can predict that following any process model too rigidly will lead to difficulties, because one cannot really be certain where one is in the lifecycle except retrospectively. In particular, it will be impossible to say with certainty what the requirements are, let alone whether or not they are met, until the system is actually in place and running.

4 A Classification of Requirements Methods

One project at the Centre for Requirements and Foundations at Oxford had as its goal to classify and evaluate existing requirements methods[7] and tools in a unbiased way; this is important because of the often exaggerated claims by vendors of the many competing methods and tools that can be found in the marketplace. This project is exploring the hypothesis that each requirements method has its own, usually unarticulated, theory of organisations, which is therefore an implicit sociological theory. From this it follows that a good classification of sociological theories would provide a good basis for classifying requirements methods.

We found a useful starting point in the work of French philosopher Jean-François Lyotard [34]. His scheme first distinguishes between (what he calls) *modern* and *postmodern* theories of society; a modern theory relies upon a so-called *meta-narrative*, or "grand unifying story," to legitimate its claim to universality. Modern theories are further divided into (what we call) *unitary* and *dual* theories. Unitary, or *systems* theoretic, approaches assume that there is

[7]What we call "methods" are often called "methodologies" by practitioners. But in an academic context, the word "methodology" should properly be used for the study and comparison of methods, and that is how it is used in this chapter.

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 a split along class lines, for example, between workers and managers.

Lyotard’s definition of postmodernism [34] says that societies are composed of many “local language games” that cannot necessarily be unified, or neatly divided into parts. Many other notions of postmodernism appear in the literature, some of which seem very superficial. Lyotard’s definition appeared relatively early, and has some substantial content, being inspired by ideas from Wittgenstein’s late period. In general, the word “postmodern” seems to have been overworked.

We[8] 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 *dual* and *critical*, and *democratic* and *network*, respectively. (“Network” here refers to theories which claim that society is organised into a number of loosely coupled nodes; Bruno Latour is a leading advocate of this viewpoint.) Figure 2 gives a diagramatic representation of this taxonomy.

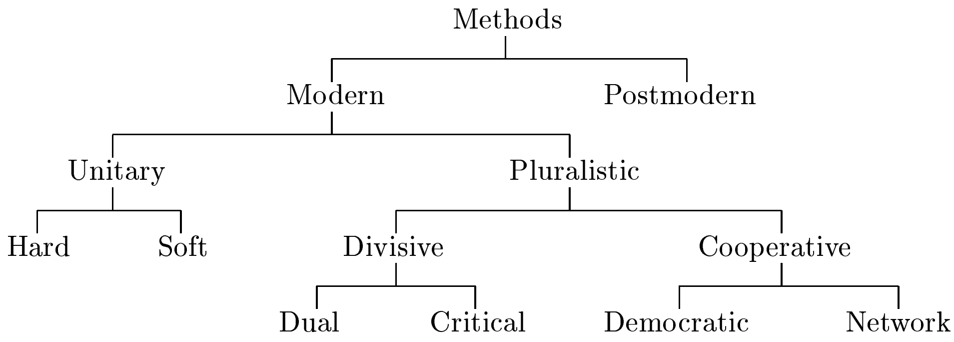

Figure 2: A Taxonomy for Requirements Methods

Most existing work in Requirements Engineering falls within the unitary systems classification; in particular, the most familiar structured design methods are all hard unitary. Work in the so-called Scandinavian School falls under the cooperative democratic classification. There is very little work in Requirements Engineering that could be called postmodern; perhaps some recent directions in Computer Supported Cooperative Work come close, and there are also the studies described in the next two sections.

A major goal of this project was to compile a *Methods and Tools Handbook* that would be useful to managers in deciding how to organise actual projects. This document includes a taxonomy of relevant methods and related disciplines, with an annotated selection of relevant books, papers, individuals, groups, and tools. It tries to identify the best of these, and to indicate the kind of application for which they may be particularly suitable. The first criterion used for classification is the social theory that underlies the method, based on our hypothesis that the most important determinant of success is that this social theory should match the actual social organisation where the system will be deployed. See [4] for more details on this project.

It is interesting to consider the methodological aspect of this classification scheme. It was not possible to base the scheme on the current practice of requirements engineers, because

[8] This version of the taxonomy is the result of much discussion within the Centre, including some very significant help from Dr. Susan Leigh Star; Matthew Bickerton was the project leader.

(we believe that) current practice does not take sufficient account of the social dimension of requirements. In particular, nearly all current industrial projects use hard unitary modern methods. Hence, this classification scheme was constructed by the researchers, using their judgement based on literature, their own experience, and discussions with others; it is not directly situated in the work of practising requirements engineers. Of course, the requirements methods that we classified are in the world of working requirements engineers, even though no single engineer can be expected to have had experience with all of them.

5 Value Systems

Goguen and Linde [16, 17] have developed a method that yields a value system for an organisation (or part thereof) from a collection of naturally occurring stories (and possibly jokes[9]) told by members of an organisation among themselves on informal occasions, such as their coffee and lunch breaks. Note that, unlike the nursery example given in Section 2.2, here we are primarily concerned with what Labov called narratives of personal experience, in which an individual relates events that were personally experienced [28]. The method classifies the evaluative material (in the sense discussed in Section 2.2) of the stories collected, using a formal structure called a *value system tree*, in which higher level nodes correspond to higher level values, and lower level nodes correspond to refinements, applications, or corrections of superordinate nodes. Because members of an organisation who tell a story are socially accountable for doing so, the evaluative material that they use to justify their telling that story reveals their shared values.

Figure 3 shows part of a value system tree[10] obtained by Goguen and Linde [16] in 1978. It represents the values of a small corporate recruitment (i.e., "head hunting") firm. The tip nodes in this tree are situated in the sense that they are taken directly from actual narratives by members require more background information in order to be understood. Many interior nodes, which express superordinate values, are also situated in this sense, but others were created by the analysts, by clustering nodes into larger and larger related groups, in the general style of the KJ method[11] [26].

The edges in Figure 3 express relationships of subordination; these are situated to the extent that there is evidence for them in the structure of the discourse; moreover, members could have been asked about them. The nodes at the three top-most 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 nodes at the fourth level of Figure 3 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. This seems to be one reason why it is difficult (or even impossible) to elicit values from members just by asking for them. Indeed, value systems, like many other aspects of social life, are tacit knowledge.

The tree in Figure 3 is an analysts' construction: the members of the company analysed do not talk about the properties of their value system tree, such as its number of nodes or edges; they would not have names for these categories, and they would not regard questions about them as meaningful. Instead, this tree serves as a formal summary of the data that was collected, and of the analysts' understanding of it; its structure is *open*, in the sense that it remains subject to revision in the light of further data and further analysis, and *contingent* in

[9] See the classical work on jokes by Harvey Sacks [40].

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

[11] This method was introduced by the Japanese anthropologist Jiro Kawakita for classifying artifacts, and it is now rather widely used by Japanese businessmen and computing scientists. It provides heuristic guidelines for combining clusters, separating clusters, etc.

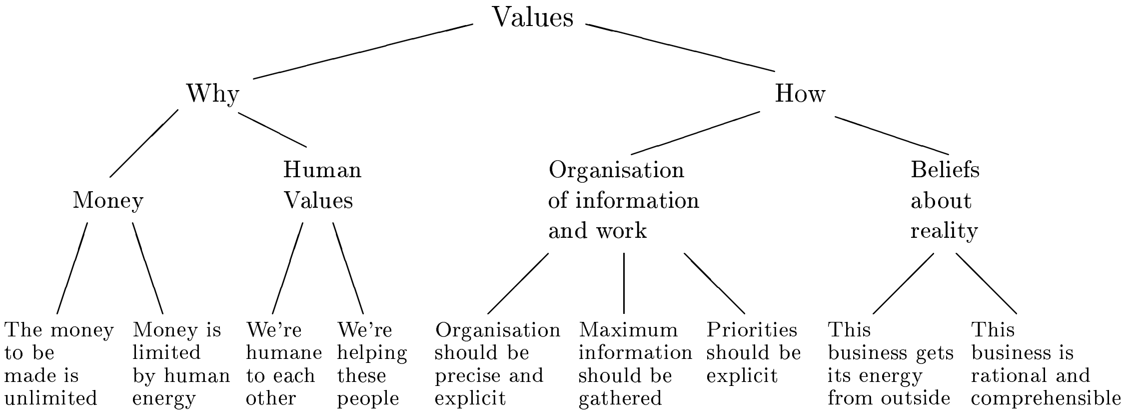

Figure 3: A Value System Tree

the sense that it is the result of many discussions, during which it gradually evolved to the form shown; quite possibly, further discussion would have led to further evolution. Nonetheless, we would expect that any changes produced by new data or analyses would be relatively small, based on the effort that had already gone into constructing this structure.

Such a structure can help a requirements engineer to make appropriate trade-offs between conflicting requirements of the client and/or end user. The hierarchical structure of the tree suggests which requirements should be given precedence over others. Also, the nature of any conflicts that appear may be clarified, because the higher level values are more significant. For an even more detailed analysis, weights could be assigned to values based on their frequency in the data that support them.

6 The Structure of Sporting Events

An interestingly different example comes from Stephen Toulmin's suggestive book *The Uses of Argument* [49]. Here the nodes and the tree itself are familiar categories to members; in particular, members have technical terms for these and many similar categories. This example is the annual regatta at Henley. Figures 4 and 5 show part of an initial "draw" for this regatta. Toulmin [49] presented his data in the form shown in Figure 4, saying that it came from "the sports page of a Sunday paper," presumably the (London) *Times* in the mid-1950s. Figure 5 presents the same information in the form of a traditional Computing Science tree.

The tip nodes in Figure 5 represent boats (i.e., crews), and the non-tip nodes represent individual "heats" with their (not yet determined) winners. Such a structure represents a *regatta*, and the special case where no winners are filled in is called a *draw*. Each edge points to a participant in a heat, which in general is the winner of a "subregatta," represented by a subtree. The number of non-tip nodes is the total number of heats, and the number of edges is the number of instances of boats racing. Note that each non-tip node is the root (i.e., top-most node) of a subregatta, and that the query marks represent winners that are not yet determined.

This formal structure is situated in the events of a particular actual regatta, and it is used in an emergent, local, contingent, open way. The query marks on the non-tip nodes of the draw in Figure 5 become instantiated with the actual winners as the concrete events of the regatta unfold in actual time, until the whole regatta is summarised by a single structure that tells what happened, including which boats raced in each heat, and who won. Another manifestation of openness and contingency is that a boat could be disqualified or withdrawn, resulting in fewer heats than scheduled in the original draw. Moreover, the structure is also open at the *meta* level, in that new methods could be added or old ones modified. For example, if we initially only have methods for constructing a draw and for setting the winners in that structure, then we would have to add a new method for restructuring regattas in order to allow Stewards to drop and rearrange boats or heats, as would be needed for handling disqualifications and withdrawals. Thus, the regatta structure only attains immutability in retrospect, after the event is over. However, it is always mobile and compressible. As an example of compression, a radio announcer might tell the number of heats that have been held so far. In fact, sports fans are fascinated by statistics, which compress the structure of some event (or series of events) into a single number.

After a discussion of what formalisation is, we will formalise (some aspects of) the structure of the Henley Regatta as an abstract data type, and we will also formalise the meta language used to describe it.

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 4: A Draw for the Henley Regatta

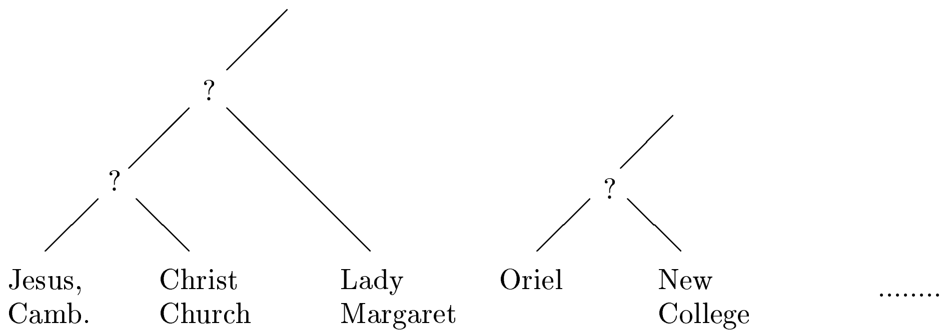

Figure 5: Tree for the Henley Regatta Draw

6.1 What is Formalisation?

Formalisation is not itself a formal activity, and in fact there are many interpretations for what formalisation may be. This subsection will give a rather precise explication of a fairly standard interpretation from Computing Science, but we will then relate it to social issues in a way that is far from standard.

The principles of accountability and orderliness suggest the possibility of formalising social interaction, but I hope our previous discussions have convinced the reader that there are limits to how successful any such formalisation could be. In particular, it will not be easy to formalise domains where there are many *ad hoc* special cases, or where much of the knowledge is tacit. Formalisation will be more successful on narrow and orderly domains, such as sporting events, that have long traditions, rule books, referees, regulating bodies, etc. For example, it would be more difficult to formalise a children's game than the Henley Regatta, and more difficult still to formalise queuing behaviour in Japan, where queues tend to be rather informal. In fact, there are degrees of formalisation, from dry to wet, and it can be important not to formalise beyond a certain appropriate degree. Cookery recipes are an interesting example, showing how an intermediate degree of formalisation is possible and helpful, whereas a very formal treatment would be unhelpful, if it were even possible.

Every formalisation requires a distinction between an *object* level, for describing that which is to be formalised, and a *meta* level, which provides a language for expressing the formalisation. The object level models the world of members, while the meta level is the language of the analysts who do the formalisation. The object level may have technical terms and rules not used by members themselves, but it is necessary to provide an interpretation for these terms into the members' world; in general, this cannot itself be formalised, and instead is a tacit part of the analysts' culture. Of course, any formalism is emergent, contingent, local, open, and vague. These qualities of situatedness can therefore be considered to describe some basic limitations of formalisation. Note that the distinction between the object and meta levels of description is parallel to the distinction between the member and analyst cultures discussed in Section 2.1.

In the driest formalisations, the meta language is also formalised, so that the object level model is a formal theory in the meta language. In less fully formalised models, the meta language may simply be a natural language, such as English, or a somewhat stylised dialect. Note that there may be rules at both the object and meta levels. Rules at the object level are part of the model, while rules at the meta level define the language that is used for formalisation.

It is important to notice that a formal description of some real world domain cannot in itself capture the meanings assigned by members concerned with that domain; the analyst's interpretations of the formalism are necessary for making this bridge.

The next subsection will formalise the structure of the Henley Regatta, using the formal meta language of abstract algebra, and will also explain its social situatedness.

6.2 Situated Abstract Data Types

Members are aware that the same structure can be represented in a variety of ways; for example, it can be represented by a table in a newspaper as in Figure 4, by a verbally presented list, by a list displayed on a scoreboard, or by the tree shown in Figure 5. This means that there is a precise structure that is independent of how it happens to be represented, i.e., we have what computing scientists call an *abstract data type*. This subsection explains this concept in more detail, while applying it to the Henley Regatta. Unfortunately, the formalism is rather technical; further mathematical details may be found, for example, in [21] and [15].

Several different kinds of entity are involved in a regatta. Some of these can be arranged in a hierarchical classification scheme according to what we will call their *sorts*. Sorts correspond to an important class of members' categories, but not every members' category is formalised by a sort; for example, heats are not formalised this way. We will say that boats are of sort `Boat`, completed regattas are of sort `Reg`, possibly not yet specified boats are of sort `Boat?` (a supersort of `Boat`), and possibly not yet completed regattas have the supersort `Reg?`; the latter includes all draws. It is convenient to assume that `Boat` is a subsort of `Reg` and that `Boat?` is a subsort of `Reg?`; these assumptions imply that there are trivial regattas consisting of just one boat, which could even be the unknown boat, denoted "`?`". Some sorts are *built in*, in the sense that they are already defined. Two examples are integers and identifiers. The latter have the sort `Id` from the built in module `QID` that provides identifiers, here used for naming boats; it is convenient to accomplish this by considering `Id` to be a subosrt of `Boat`. A diagram showing the subsort relations involved in this example is given in Figure 6 (the sorts `12` and `Index` are discussed in the appendix).

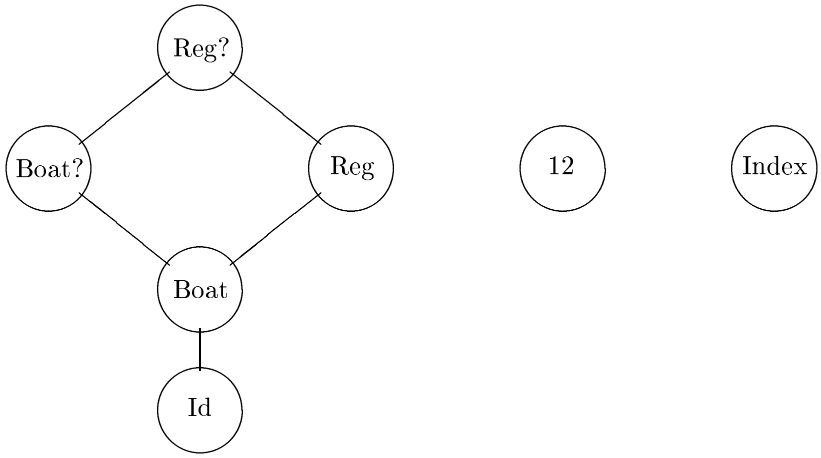

Figure 6: Classification Scheme for Some Members' Categories

How can we avoid being tied to one particular representation, as Toulmin was? The key is to focus attention on the *methods* that members use to describe (or construct) representations. It is useful to distinguish two kinds of methods: *constructors*, for building representations from more primitive parts, and *selectors*, for extracting particular information[12]. For regattas, the most important method is a constructor that adds a new heat; it must specify the two contestants, and also provide a slot for the winner. In Figure 5, each non-tip node represents a heat, where the two contestants are the winners of subregattas, or else are given boats; the query mark represents a not yet determined winner. In general, if `R` and `R'` are regattas and `B` is a boat, then `heat(R,R',B)` constructs a new regatta, by adding a heat in which the winners of the subreggatas `R` and `R'` race against each other, with `B` the winner of that heat.

Selectors correspond to certain members' categories. For example, there is a selector that extracts the winner of a given regatta by taking the winner of its final heat. But we do not formalise "winner" with a sort.

Methods respect the sorts of representations, in that it is only meaningful to apply a given method to representations of certain sorts, and not to others; moreover, each method has a *target sort*, for the kind of representation (information) that it yields. For example, the `heat` method takes three inputs, two of which are regattas, and one of which is a boat; it is not meaningful to give a regatta, or an integer, for its third input. We call the list of input sorts the *source* of the method. In addition, there can be "constants" that do not have any inputs, but do have a target sort; for example, the unknown boat "`?`" has target sort `Boat?`. It is convenient to think of these as methods having no input, i.e., as having an empty source.

Given a particular representation, say by trees, we can collect all of the possible structures in that representation into a system called an *algebra*, in which each sort corresponds to the set of representations of that sort, and the methods (and selectors) correspond to functions that map representations to other representations (or else to built in values, such as numbers). If A denotes a given algebra, then A_s denotes its set of representations of sort s. If s' is a subsort of s, then we require that $A_{s'}$ is a subset of A_s.

For example, the method that adds a new heat to a regatta can be seen as a function

[12]In object oriented programming, the word "method" refers to operations that can modify, and the word "attribute" refers to operations that only extract information, that is, to what we have called selectors.

```
heat : Reg? Reg? Boat? -> Reg?
```

which takes two regatta representations and a boat, and constructs the new regatta where the winners of the subregattas race against each other. Similarly, the selector that gives the winner of a regatta is a function

```
winner : Reg? -> Boat?
```

from regattas to boats. Also, constant methods (such as "?") designate particular representations in an algebra.

It is often convenient to have two different versions of a method, one for not yet completed structures, and the other for completed structures; then we have an *overloaded* method. For example, in addition to `heat` as defined above, we may also have a method

```
heat : Reg Reg Boat -> Reg
```

for constructing completed regattas.

A collection of sorts (with their subsort relations) and a collection of operations (including methods and selectors) over it are together called a *signature*[13]. Given a signature, we can construct its "well formed expressions," called *terms*: first, all constants are terms; and second, if o is an operation with source sorts $s_1, ..., s_n$ and if $t_1, ..., t_n$ are terms such that t_i has target sort s_i, then $o(t_1, ..., t_n)$ is also a term, and it has the same sort as the target sort of o. For example,

```
heat(heat('JesusCam, 'ChristCh, ?), 'LadyM, ?)
```

is a well formed term, in which OBJ3-style identifiers are used as constants for the names of boats; these identifiers begin with a quote and contain no spaces [23] (we have also further abbreviated the boat names).

Given any term t and algebra A over the same signature, that term denotes a unique representation in A. This *value*, or *denotation*, of t is determined just by first finding the values in A of any constants in t, and then applying the functions in A that correspond to the operations in t to those values, and then applying further functions to further values, and so on, until a final value is reached. For example, the above term denotes the left subtree in the representation shown in Figure 5.

We are now in a position to formalise what it means for two representation systems to be "essentially the same," in the sense that they can represent exactly the same thing. The idea is that there must be a translation between the two representation systems that preserves the meaning of all operations. We have already formalised the notion of a representation system as an algebra. So it makes sense that the formalisation of translation should be a system of mappings between the sets of the algebras that preserve the operations. Let's say that the two algebras are A and A', and that the set of sorts in the signature is S. Then we want a collection of functions $f_s\colon A_s \to A'_s$ for each s in S, such that if o is an operation with target sort s, with source sorts $s_1, ..., s_n$, and if A_o and A'_o are the functions that interpret o in A and A' respectively, then preserving the meaning of o amounts to requiring that

$$f_s(A_o(r_1, ..., r_n)) = A'_o(f_{s_1}(r_1), ..., f_{s_n}(r_n))$$

[13] We may also have to include some functions on built in values.

where $r_1, ..., r_n$ are representations in A of sorts $s_1, ..., s_n$, respectively. Also, if c is a constant of sort s, then we require that

$$f_s(A_c) = A'_c$$

where A_c and A'_c are the representations of c in A and A', respectively.

However, this isn't quite enough: we need the representations in A and A' to be in exact correspondence with each other. For this, we need to require that each of the functions f_s is bijective (i.e., is one-to-one and onto). A collection of functions f_s that has all these properties (of preserving all the operations and being bijective) is called an *isomorphism*.

The development and use of algebras in the sense explained above was a major advance in mathematics. Alfred North Whitehead [50] prefigured this revolution in the late nineteenth century, but it was Emmy Nöther who is largely responsible for systematising and widely applying these concepts, thus creating what is now called "modern algebra" or "abstract algebra," and widely taught in undergraduate mathematics curricula.

But we are still dealing with representations. How can we abstract away from these, to obtain structures that are truly independent of how they are represented? The answer has two parts, of which the second is unfortunately a bit technical.

The first is just to describe the *relationships* that necessarily hold among the given operations; it turns out that it suffices to use equations for this purpose. For example, the fact that the winner of any regatta of the form `heat(R,R',B)` is `B` is expressed by the equation

```
winner(heat(R,R',B)) = B .
```

This is a relationship between the category "winner" and the constructor method "heat".

So far, our approach for specifying an abstract data type consists of giving a collection of sorts with subsort relations, a collection of operations, and a collection of equations; let us call these three collections together a *specification*. Then a *model* for such a specification is an algebra that provides sets for the sorts (with subsets for the subsorts) and functions for the operations, such that all of the equations are satisfied. The elements of such a model provide a system of representations for the categories and operations in the signature. We now have a way of specifying representations that is truly abstract, in that it says nothing whatsoever about the representations themselves. But unfortunately, there are too many models, and they are not isomorphic to each other. We need one more principle to get what we want; this principle is called *initiality*, and it amounts to assuming the following two principles:

1. *No junk*: every representation in the algebra can be constructed using methods in the given signature.
2. *No confusion*: two terms denote the same representation if and only if they can be proved equal using the given equations.

So now we take as models only those algebras that satisfy not only the equations, but also the two principles above. Such models are called *initial models*, and it can be shown that any two initial algebras are necessarily isomorphic. Thus, we finally have the representation independent way of specifying structures that we have been seeking.

In this example, it is important to distinguish between the object and meta levels of description; a different language is used at each level, for a different purpose. The object level language involves boats, heats, and so on, and events here may construct draws, announce winners, etc. The meta language is also formal, and involves sorts, methods, equations, etc.; events at this level may add new methods, revise equations, etc. For example, in the OBJ3 code in the appendix, a method for restructuring regattas is given as an enrichment of the previously given specification that only involved methods for heats and winners.

The above has sketched a precise mathematical theory of abstract data types, in which an abstract type is an isomorphism class of many-sorted algebras; see [21] for some further details. One particular result in this theory is that any computable many-sorted algebra, and thus any abstract data type, has a representation by trees. Thus, it is not surprising that each of the examples discussed here has a tree representation[14]. (There is also a somewhat more sophisticated approach, based on behavioural satisfaction, that does not require initiality; see [12].) A complete formal specification of our Henley Regatta abstract data type is given in the appendix, using the specification language OBJ3 [23]. This specification is executable, and has actually been executed. (Experience shows that it is necessary to test all but the most trivial specifications in order to eliminate bugs.)

It may seem surprising how complex the complete specification is, with its various sorts, subsorts, methods, and especially equations. But the Henley Regatta really does have boats, heats, regattas, winners, not yet determined boats, etc., and the relationships among them really are rather complex. Also, we know from experience that it can take quite some effort to learn how some new sport is structured. It is clear that the methods for constructing and restructuring a regatta really are rather complex. Moreover, this kind of complexity is not unique to this example, but is typical of sporting events, games, and many other social phenomena.

It may seem even more surprising how sophisticated is the meta level language that we used for formalising the structure of the Henley Regatta, with its categories of sort, subsort, signature, algebra, term, isomorphism, equation, etc. Perhaps this language could be simplified, but the approach presented here was developed in the context of Computing Science, where there is a strong preference for a rigorous mathematical development [21]. In fact, the abstract data type concept is rather sophisticated, and I have simplified many details, and omitted others. However, the emphasis on the social situatedness of abstract data types seems to be entirely new.

Of course, the above discussion does not imply that members of a social group some of whose behaviour can be formalised by an abstract data type are familiar with the concept of an abstract data type; even Toulmin [49] only uses concrete representations, without realising that they are algebras, that different representations give isomorphic algebras, or that there is no unique best representation. Similarly, there is no implication that members are familiar with the social aspects of formalisation. For example, much the same kind of process as described in this subsection occurs when a computing scientist formalises some aspect of computing practice by designing a new abstract data type, but this does not imply familiarity with (for example) the qualities of situatedness or the retrospective hypothesis, or even with the algebraic formalisation of abstract data types. On the other hand, perhaps such a familiarity would help to prevent some rather ill-advised formalisations.

7 Discussion and Summary

Although each of our four examples (in Sections 3, 4, 5 and 6) has a tree structure, these structures are used for different purposes, and are embedded in their social contexts in different ways, because of their different relationships of accountability. Let's compare them.

[14]The linear ordering in Figure 1 is a particularly simple kind of tree having no branches, and just one tip node; it looks more like a tree if it is stood on its side.

The regatta example of Section 6 shows that abstract data types occur "in nature," e.g., in ordinary social interaction, in the sense that the members of ordinary groups (such as sports fans) organise their talk in ways that correspond to instances of this concept. Different representations of the same abstract structure are displayed on scoreboards, printed in newspapers, shown on television in compact graphical forms, and are also presented in natural language. Of course, one expects to see abstract data types in the dry areas of Computing Science, such as programming languages and databases, and they are also familiar in the Structured Analysis notations of Requirements Engineering, but I was surprised to find abstract data types in live social interaction. However, we should not forget that these structures are still situated, and thus contingent, emergent, local and open; in particular, the categories and methods of our example need to be interpreted in the concrete events of an actual regatta. Moreover, new methods can be added, or old ones revised, to cope with new circumstances.

Note that even though we formalised both the object (member) and meta (analyst) levels of description in this example, contingence, emergence and openness occur at both levels in relation to any concrete instance: an analyst (the author) has constructed an abstraction which is now separated from the concrete instances that motivated its construction. This abstraction cannot be expected to precisely match any particular concrete instance. Of course, this is no different from the situation with maps, and such abstractions can still be a useful starting point for doing the practical work of understanding how some particular situation is produced by members.

In the case of the value system tree of Figure 3, the analysts are accountable for the structure, while the members of the organisation analysed are accountable for the bits of language attached to all but the highest level nodes. Thus, the tree structure itself is only meaningful to members of the analyst culture. All of the tip node labels were extracted from the data, and in fact so were most of the non-tip labels, although they get shorter and more abstract as they go further up; the analysts are accountable for the extraction and abstraction. Thus, this structure is open, contingent, emergent, and local with respect to two different communities, in two different ways.

The edges in the value system tree express relationships of subordination; these are situated in the members' culture to the extent that there is evidence for them in the structure of the data collected. Although the tree structure belongs to an abstract data type, the *meanings* of the labels on nodes are not captured by this formalism, nor is the restriction that the meaning of a higher level node is more general than that of its subordinates. In general, the lower a node is, the more background information about the organisation is needed to understand its meaning; this is because the tip nodes (and some of the interior nodes) are more situated, while the higher level nodes are more abstract immutable mobiles; these tend to be shorter, with fewer context bound terms.

The classification of requirements methods discussed in Section 4 is similar to the value system tree discussed in Section 5, in that the non-tip nodes were constructed by analysts. However, it differs in that the classification scheme was largely determined in advance, and the detailed assignment of methods to classes was done later. Nevertheless, there were some readjustments. For example, we originally had a category for methods based on psychology, but later realised that these were hard unitary modern methods; therefore we dropped that category. Assigning methods to classes could be considered a test of the hypothesis that implicit social theories are a significant property of requirements methods. It would appear that this hypothesis has been confirmed. Note that the Requirements Engineering community produced the tip nodes, the sociology community produced the internal nodes, and the analysts put the whole together. The *Handbook* is now in the hands of its user community, and we expect to receive further feedback, which will help us to assess the adequacy of our construction. Thus, the situatedness and accountability of this example involve three different communities, which are inter-related in a complex way.

In the waterfall lifecycle model of Section 3, each so-called phase is a members' concept, and so is each temporal ordering relationship; software engineers use these phase names frequently, to help them classify their activities. They also recognise the *logical* priority of "earlier" phases over "later" ones. However (based on informal ethnography), it seems to be mainly managers who insist upon the *temporal* priority of "earlier" phases; practising software engineers, especially those with some years of experience, generally recognise that "reality" often turns out differently (see also the paper of Button and Sharrock in this volume). The preferred number of phases, their names, and their definitions all vary somewhat with the organisation, textbook, method, or individual practitioner examined. Thus, the structure shown in Figure 1 does not represent a consensus, but rather an analyst's idealisation, necessarily involving some arbitrary choice. Nevertheless, any trained software engineer will recognise that Figure 1 does represent (a version of) the waterfall model.

In the regatta example, all the structure in the abstract data type can be referred back to the community of members; in particular, members have categories for both the nodes and the edges in Figure 5. However, the tree representation is only one among several that are used by members; others include tables (as in Figure 4) and verbal listings. But all of these are isomorphic when viewed as algebras. This is the precise sense in which we have an *abstract* data type. But we should not forget that any concrete instance, i.e., any particular Henley Regatta, is still very much socially situated.

The other three examples use a simple tree structure to support a hierarchical classification, in a way that resembles classes with inheritance in object oriented programming. This kind of tree can easily be described as an abstract data type. But it would be problematic to try to formalise the meanings of the labels, the relationships of subordination among them, or the meta language used for these. The classification scheme for methods is somewhat more of an analysts' construction than the value system tree. The analysts' rôle in the waterfall phases example was a fairly minimal tidying up. Figure 6 is also a hierarchical classification scheme.

It is important to recognise that in all four examples, the operations of abstraction have, to a varying extent, diminished the connection of resulting formal structure with its original social context. I wish to argue that this is not the result of an inadequate method, but rather that it is *always and necessarily* the case that operations of systematising, abstracting, theorising and concluding will have such an effect, even if carried out according to the most rigorous canons of ethnomethodology. The construction of immutable mobiles necessarily reduces the situatedness of data and makes it drier, and any assertions made by analysts necessarily fall into this area. Note also that dryness comes not merely in different degrees, but also in a wide variety of kinds, resulting from the complex relationships of accountability between different communities, including that of analysts.

Thus, the relationship between the formal and the social aspects of information is not one of antagonism, where one must be rejected and the other accepted; rather, these two aspects of information are *complementary*, and they are both essential for successful Requirements Engineering. Moreover, the relationship between the formal context insensitive and the social context sensitive aspects of information is very *complex*, in that there may be several different facets, which arise in different social contexts, have different levels of abstraction, and are interconnected in complex ways.

7.1 Summary

We have used some ideas from Ethnomethodology, such as the principles of accountability and orderliness, to better understand information. We have argued that the situatedness of information has consequences that are important for analysts to understand, including the emergent, local, contingent, and open qualities. In general, traditional Sociology has tried to avoid such consequences, in the misguided pursuit of objectivity. Ethnomethodology can be seen as a reaction against this, in its efforts to reconcile a radical empiricism with a deep appreciation of the situatedness of social data.

We have also used ideas from the Sociology of Science (especially work of Latour) and from Computing Science to explicate the nature of dry information. Dry information often loses the property of embodiedness, and is also less emergent, contingent, and local. However, even the most formal information is still situated, and in particular, is open, emergent, contingent and even embodied at the meta level, where analysts are accountable for its formalisation. Similarly, even the wettest information about social interaction is necessarily partially abstracted from its social context, in order to be presented to an audience of analysts. In particular, the analyst must always speak in a meta language.

We have suggested the *formalisation hypothesis*, which says that formalisation is a key way to make information drier, i.e., more immutable and mobile. We have also suggested the *retrospective hypothesis*, which says that plans and explanations (and much other socially situated information) can only become dry after the events to which they refer have reached some stage of completion.

We have argued that abstract data types can be found in the discourse of ordinary social groups, such as sports fans. It seems likely that many other instances of abstract data types can be found in social interaction. The "language games" proposed by Wittgenstein seem a suggestive hint, because we have seen that abstract data types provide a natural way of formalising games. However, it does not seem likely that all socially situated information can be represented by abstract data types. In particular, the use of the terms "category" and "method" in Ethnomethodology seems to cover a much larger ground than that formalised by abstract data types.

We have also suggested the importance of not adhering dogmatically to any one research method, whether it is ethnomethodology, discourse analysis, questionnaires, object oriented analysis, or formal domain modelling. Instead, we have argued for the flexibility to use whatever method seems best to fit the given circumstances.

7.2 Implications for Requirements Engineering

The retrospective hypothesis explains why it can be so difficult to manage the requirements of a large system: it only becomes clear what the requirements really are when the system is successfully operating in its social and organisational context. Thus, a reasonably complete and consistent set of requirements for a large, complex system can only emerge from a retrospective reconstruction. The retrospective hypothesis also explains why it can be so difficult to enforce process models on actual system development projects: it is difficult even to know what phase a given action fits into until some coherence has emerged retrospectively. Note that it takes work by members to achieve a retrospective reconstruction, and that this work is often not done for many projects, because of the principle of volume.

We can now understand why it is impossible to completely formalise requirements: it is because they cannot be fully separated from their social context. More specifically, the qualities of situatedness explain why the lifecycle phases cannot be fully formalised. Indeed, the activities that are necessary for a successful system development project cannot always be expected to fit in a natural way into any system of pre-given categories, and practising software engineers often report (informally) that they have to spend much of their time circumventing "the system." Nevertheless, it is the very essence of Requirements Engineering to use and construct such dry structures, because they are necessary for constructing actual computer-based systems. What is unnecessary is a slavish adherence to narrowly prescriptive plans and categories. Note that in general, abstractions (immutable mobiles) have only a practical utility, and must be interpreted concretely in order for that utility to be made manifest.

These considerations have consequences for tools to support Requirements Engineering. Perhaps the most important consequence is that requirements tools must provide very strong support for retrospective revision; in particular, they must be very flexible, to accommodate the frequent changes in requirements and their links with other objects (such as specs and code). Another requirement is that varying degrees of formalisation should be accommodated, ranging from raw data to mathematical formulae. Moreover, information that is heavily situated should come with pointers into its context (e.g., background ethnographic information, such as audio and video clips of work and interviews, questionnaires and their results, and sample documents from the work environment), in order to make it understandable by workers who have not had direct contact with the client organisation. This suggests that multimedia documents may have great potential for the support of Requirements Engineering.

The area that we have called the Social Theory of Information offers some hope of furthering Requirements Engineering, by clarifying the situated nature of the information involved, and indeed, of the requirements process and the whole lifecycle. This should help us to effect an ongoing, practical reconciliation between formal technical issues and socially situated issues in the actual *practise* of Requirements Engineering, as is needed for building systems that work successfully in their social context.

A Formal Specification for the Henley Regatta

This appendix gives a formal specification of an abstract data for the structure of the Henley Regatta, using the executable part of the programming and specification language OBJ3. It is not feasible to give a complete tutorial on OBJ3 here; the interested reader should look elsewhere for details, e.g., [23]. However, a few remarks may help the untutored reader to obtain a rough overview of the specification given below. There are four modules, each of which begins with the keyword "`obj`" and end with the keyword "`endo`." Immediately after "`obj`" comes the name of the module. Sorts, operations, subsorts, variables, and equations are declared after fairly obvious keywords; also, "`cq`" indicates a conditional equation, while "`pr`" and "`dfn`" indicate module importations, the latter with a renaming of the principle sort (in this case, from `List` to `Index`).

The first module, named "`12`", merely introduces two constants, "`1`" and "`2`"; these are used for indicating one or the other of the two boats in a heat. Note that this is an abstract data type, in the sense that we could have chosen different representations for the two boats, such as "`A`" and "`B`", or "`L`" and "`R`"; any such choice will yield an isomorphic (two element) initial algebra. The second module, named "`LIST`", is a *parameterised module* for forming lists of anything; the list constructor has the syntax "`_._`" for placing a new element at the head of a list; `nil` is the empty list. Inside the third module, we form and import the module `LIST[12]`, renaming its principal sort `List` to be `Index`; these lists are used for picking out particular instances of a boat racing in a regatta; a typical term of sort `Index` is "`1 . 1 . 2 . nil`". This module also introduces the constructor `heat` for regattas, the method `swin` for setting winners of heats, and the selector `winner`. The final module `SREG` adds a method `sreg` for restructing regattas. The line "`vars-of HENLEY`" just imports the variable declarations from the previous module `HENLEY`.

```
obj 12 is sort 12 .
```

```
  ops 1 2 : -> 12 .
endo


obj LIST[X :: TRIV] is sort List .
  op nil : -> List .
  op _ _ : Elt List -> List .
endo


obj HENLEY is sorts Boat Reg Boat? Reg? .
  pr QID .
  subsorts Id < Boat < Reg Boat? < Reg? .
  dfn Index is LIST[12].

  op ? : -> Boat? .
  op heat : Reg? Reg? Boat? -> Reg? .
  op heat : Reg  Reg  Boat  -> Reg .

  var B B' : Boat? .  vars R R' R'' : Reg? .  var I : Index .

  op winner : Reg? Index -> Boat? .
  eq winner(heat(R,R',B), nil) = B .
  eq winner(heat(R,R',B), 1 I) = winner(R,I) .
  eq winner(heat(R,R',B), 2 I) = winner(R',I) .
  eq winner(B,nil) = B .
  cq winner(B,I)   = ? if I =/= nil .

  op swin : Reg? Index -> Reg? .
  eq swin(heat(R,R',B), nil)   = heat(R,R',B).
  eq swin(heat(R,R',B), 1 nil) = heat(R,R',winner(R,nil)).
  eq swin(heat(R,R',B), 2 nil) = heat(R,R',winner(R',nil)).
  cq swin(heat(R,R',B), 1 I)   = heat(swin(R,I),R',B) if I =/= nil .
  cq swin(heat(R,R',B), 2 I)   = heat(R,swin(R',I),B) if I =/= nil .
  eq swin(B,I) = ? .
endo


obj SREG is
  pr HENLEY .
  op sreg : Reg? Reg? Index -> Reg? .

  vars-of HENLEY .
  eq sreg(heat(R,R',B), R'', 1 nil) = heat(R'',R',B).
  eq sreg(heat(R,R',B), R'', 2 nil) = heat(R, R'',B).
  eq sreg(heat(R,R',B), R'', 1 I)   = heat(sreg(R,R'',I),R',B).
  eq sreg(heat(R,R',B), R'', 2 I)   = heat(R,sreg(R',R'',I),B).
  eq sreg(heat(R,R',B), R'', nil)   = ? .
endo
```

Note that all of this code has actually been run in OBJ3.

References

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

[2] Jon Barwise and John Perry. *Situations and Attitudes.* MIT, 1983. (Bradford Books).

[3] Alton L. Becker. Text-building, epistemology, and aesthetics in Javanese shadow theatre. In Alton L. Becker and A. Yengoyan, editors, *The Imagination of Reality: Essays on Southeast Asian Symbolic Systems.* Ablex, 1979.

[4] Matthew Bickerton and Jawed Siddiqi. The classification of requirements engineering methods. In Stephen Fickas and Anthony Finkelstein, editors, *Requirements Engineering '93*, pages 182–186. IEEE, 1993. Also, Technical Report, Centre for Requirements and Foundations, Oxford University Computing Lab, 1992.

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

[6] Barry Boehm. A sprial model of program development and enhancement. *Software Engineering Notes*, 11(4):14–24, 1986.

[7] Richard Boland. In search of management accounting: Explorations of self and organization. Technical report, Case Western University, 1991.

[8] J. Conklin and M. Bergman. gIBIS: A hypertext tool for exploratory policy discussion. *ACM Transactions on Office Information Systems*, 6:303–331, 1988.

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

[10] Anthony Finkelstein, J. Kramer, and M. Goedicke. Viewpoint oriented software development. In *Proceedings, Third International Workshop on Software Engineering and its Applications*, pages 337–351. Springer Verlag, 1990.

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

[12] Joseph Goguen. Types as theories. In George Michael Reed, Andrew William Roscoe, and Ralph F. Wachter, editors, *Topology and Category Theory in Computer Science*, pages 357–390. Oxford, 1991. Proceedings of a Conference held at Oxford, June 1989.

[13] Joseph Goguen. The denial of error. In Christiane Floyd, Heinz Züllighoven, Reinhard Budde, and Reinhard Keil-Slawik, editors, *Software Development and Reality Construction*, pages 193–202. Springer Verlag, 1992. Also in *Four Pieces on Error, Truth and Reality*, Programmming Research Group Technical Monograph PRG–89, October 1990, Oxford.

[14] Joseph Goguen. The dry and the wet. In Eckhard Falkenberg, Colette Rolland, and El-Sayed Nasr-El-Dein El-Sayed, editors, *Information Systems Concepts*, pages 1–17. Elsevier North-Holland, 1992. Proceedings, IFIP Working Group 8.1 Conference (Alexandria, Egypt); also, Programming research Group, Technical Monograph PRG–100, March 1992, Oxford.

[15] Joseph Goguen. *Theorem Proving and Algebra.* MIT, to appear.

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

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

[18] 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.

[19] Joseph Goguen and Charlotte Linde. Techniques for requirements elicitation. In Stephen Fickas and Anthony Finkelstein, editors, *Requirements Engineering '93*, pages 152–164. IEEE, 1993. Also Technical Report, Centre for Requirements and Foundations, Oxford University Computing Lab, 1992.

[20] 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.

[21] 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.

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

[23] Joseph Goguen, Timothy Winkler, José Meseguer, Kokichi Futatsugi, and Jean-Pierre Jouannaud. Introducing OBJ. In Joseph Goguen, editor, *Applications of Algebraic Specification using OBJ.* Cambridge, to appear 1993. Also to appear as Technical Report from SRI International.

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

[25] Marina Jirotka. Ethnomethodology and requirements engineering. Technical Report PRG-TR-92-27, Centre for Requirements and Foundations, Oxford University Computing Lab, 1991.

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

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

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

[29] Bruno Latour. *Science in Action.* Open University, 1987.

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

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

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

[33] Eric Livingston. *The Ethnomethodology of Mathematics.* Routledge & Kegan Paul, 1987.

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

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

[36] 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.

[37] Iona Opie and Peter Opie. *The Oxford Dictionary of Nursery Rhymes.* Oxford, 1951.

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

[39] Michael Polanyi. *The Tacit Dimension.* Routledge and Kegan Paul, 1967.

[40] Harvey Sacks. An analysis of the course of a joke's telling in conversation. In Richard Baumann and Joel Scherzer, editors, *Explorations in the Ethnography of Speaking*, pages 337–353. Cambridge University, 1974.

[41] Harvey Sacks. *Lectures on Conversation.* Blackwell, 1992. Edited by Gail Jefferson.

[42] Harvey Sacks, Emanuel Schegloff, and Gail Jefferson. A simplest systematics of the organization of turn-taking in conversation. *Language*, 504:696–735, 1974.

[43] 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.

[44] Claude Shannon and Warren Weaver. *The Mathematical Theory of Communication.* University of Illinois, 1964.

[45] Wes Sharrock and Bob Anderson. Epistemology: Professional scepticism. In Graham Button, editor, *Ethnomethodology and the Human Sciences*, pages 51–76. Cambridge, 1991.

[46] Wes Sharrock and Graham Button. The social actor: Social action in real time. In Graham Button, editor, *Ethnomethodology and the Human Sciences*, pages 137–175. Cambridge, 1991.

[47] 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.

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

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

[50] Alfred North Whitehead. *A Treatise on Universal Algebra, with Applications, I.* Cambridge, 1898. Reprinted 1960.

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

Contents