TECHNICAL REPORT

Computer Science

The Untrained Eye: how languages for software specification support understanding in untrained users

Carol Britton and Sara Jones

Report No 321

Dec 1998

The Untrained Eye: how languages for software specification support understanding in untrained users

Carol Britton and Sara Jones
Department of Computer Science
University of Hertfordshire
College Lane, Hatfield, Herts. UK
AL10 9AB
Tel: 01707 284354 / 284370
Email: C.Britton, S.Jones@herts.ac.uk

'Running Head' Title: The Untrained Eye.

Carol Britton is a computer scientist with an interest in the specification of software systems; she is a principal lecturer in the Department of Computer Science at the University of Hertfordshire.

Sara Jones is a software engineer with an interest in cognitive aspects of software systems; she is a senior lecturer in the Department of Computer Science at the University of Hertfordshire.

ABSTRACT

It is generally recognised that choice of languages can have a significant effect on the system development process, particularly in the early stages. In the development of interactive systems, it is essential that all stakeholders can participate in a meaningful way. In order to do this, they must be able to understand representations of key concepts produced by the developers, especially those relating to problems and requirements for the system. Some stakeholders, such as clients and potential users of the system, may be unfamiliar with the languages used by system developers. They may therefore find it difficult to understand representations produced using such languages well enough to give useful feedback to the developer.

In this paper we identify ease of understanding of representations as a key issue for interactive system development and consider how the notion of ease of understanding may be defined in this context. We then discuss an approach to evaluating software specification languages in terms of properties which may affect the understandability of representations, and which may be amenable to objective measurement. Our intention is that results from this work will help to classify existing languages in terms of ease of understanding, provide a rational basis for predicting understandability in proposed new languages, and help developers to use current languages in more imaginative ways so that they can produce representations that are easier to understand.

"The untrained eye only sees what it expects to see, and it doesn't expect much." John Rothenstein (Director of the Tate Gallery 1938 - 1964), interviewed on Woman's Hour 1961 (reported in "Mr. Tate's Gallery", BBC2, July 1997).

1. INTRODUCTION

1.1 Problem Context

It is generally accepted that the use of different languages for representation has an impact on the effectiveness with which a variety of tasks can be performed (Stenning & Oberlander 1995). This is true especially in software system development, where the effect of the choice of language on successful system development has long been recognised (Green 1989 and 1991, Johnson, McCarthy & Wright 1995, McCluskey et al 1995, Modugno, Green & Myers 1994, Roast 1997). However, the relationship between languages, representations and the quality of the system development process is not fully understood. Little is currently known about what languages are likely to be most suitable for use in which contexts. The choice of languages for particular projects often reflects the experience or preferences of the development team more than an objective consideration of possible alternatives (McCluskey et al 1995).

In the case of interactive system development, representations of the problem and the intended system are constructed on the basis of information and validation from clients and users. Representations are used initially to prompt clients and users to contribute information about the problem and the intended system; they are also used subsequently to support the clients in checking that the developer has understood and specified the client's requirements. In this paper, we address the second of these contexts, where a specification has been constructed and is in the process of being validated by clients and users. For the purpose of validation, it is essential that all those involved, including clients and users who may be untrained in the use of languages used in software development, have access to a representation that they can readily understand. It is important that the representations used during development are easy to understand so that untrained clients and users are not forced to put effort into deciphering them, rather than concentrating on their content. This problem is described most eloquently in (Green 1989): "When a train of thought is broken again and again by the need to find something out the hard way, it is difficult to piece thoughts together into inspirations; it is difficult enough even to finish a simple train of thought without making a mistake, simply because of having to get the information in some tedious and error-prone way." Therefore ease of understanding of representations is a necessary (though not sufficient) condition for successful interactive system development.

In many interactive system development projects, the need for user understanding of representations is addressed by building a skeleton prototype to offer options which can then be refined in line with the user's requirements. A prototype facilitates communication between developers and clients or users, allows users to relate what they are shown to their own experience of tasks, and enables them to give meaningful feedback on the design ideas which are embodied in the prototype.

There are, however, certain cases where the prototyping approach may not be feasible or not, on its own, adequate. For large complex systems of any kind

"exploits the capabilities of the output medium and the human visual system", rather than expressiveness: whether a language "can express the desired information". In this paper, we assume that a language has appropriate coverage (i.e. that it's semantics are appropriate for representing the things which the developer wants to represent). We are interested in what aspects of a language, over and above an appropriate semantics, are likely to facilitate the production of representations which may be easily understood by untrained users.

Ease of understanding of software systems representations is crucial for successful development of interactive systems, but evaluating such representations in themselves is fraught with problems. A representation written in a particular language may be easy to understand for a variety of reasons: for example, the developer may have many years of experience in producing this type of representation, or the context of the representation may help the reader to understand it. It is widely agreed that there are many ways that a particular system or problem may be specified, some more elegant than others. The difficulty of evaluating representations directly has been recognised by other authors. These include Stenning and Oberlander (1995) who note problems with an approach which emphasises "differences between token representations, rather than the differences of expressive power of the systems the tokens are drawn from". Scaife and Rogers (1996) also highlight the problem. In their survey of the literature on how graphical representations affect performance they note that it is difficult to draw general conclusions from the findings of specific studies, and that "It is often hard to separate general claims about graphical representations *per se* from factors that have to do with individual differences in ability in the subject or understanding of the domain-specific genre of the diagrams involved". It is precisely because of the problems of evaluating representations, that we have chosen, in this paper, to focus on properties of the languages themselves that support understanding. We are concerned, here, with the essential properties of languages, rather than representations, which are the products of the languages being used in different ways, by different people, in different situations.

The paper aims to address the question of how to evaluate languages in terms of ease of understanding, in particular, whether it is possible to predict the extent to which a language will be readily understandable by users who are new to it. The detailed objectives of the paper are:

- to explore research in related areas;
- to articulate ease of understanding in terms of properties of languages, focusing particularly on the context and restricted scope described above;
- to investigate the possibility of measurement of the properties identified.

Our intention is that results from this work will be used in three ways. First, our findings will help to classify existing languages in terms of ease of understanding, without the need to carry out a full experimental investigation of each language. Second, they will provide a rational basis for predicting ease of understanding in proposed new languages. Finally, they will help developers to use current languages in more imaginative ways so that they can produce representations that are more easily understandable by users who are unfamiliar with languages for software specification.

1998) or 'models' (for example object models, dynamic models Rumbaugh et al., 1991). A number of different diagrams (or models) may often be used in combination. Such representations may eventually be carried forward to form part of the 'specification' which defines the performance of the system to be developed. The conventions according to which they are composed are typically referred to as 'techniques' (Davis, 1993) or 'languages' (Fenton & Hill, 1993; Jackson, 1995).

2.2 Use of Terms in Relevant Literature in Cognitive Psychology

The software developer's 'diagrams' or 'specifications' are examples of what some research in cognitive psychology has referred to as 'external representations'. There has recently been considerable debate in cognitive psychology about the relationship between internal, or mental, representations, and external representations (see, for example, Scaife & Rogers, 1996). The way in which we represent the world 'inside our heads' is an important subject of on-going research, and there are a number of different views on, for example, the relationship between symbolic and distributed representations, analogical and propositional representations, and the role of mental imagery (Eysenck & Keane 1990). Since internal representations are not yet well-understood, it is currently impossible to reach any general conclusions regarding the links between internal and external representations, and the support which particular forms of external representations provide for working or reasoning with different kinds of internal representations. We agree with Cox and Brna (1993) that the debate about internal representations is unlikely to point directly to practical solutions at this stage. In this paper, we make no claims about the relationship between internal and external representations. Instead, as described in section 1, we focus mainly on the extraction of information from external representations and the kind of low-level cognitive processing which may precede the encoding of information into internal representations. For the purposes of this paper, we ignore the more difficult and less well-understood aspects of understanding, such as the way in which information from external representations may be encoded into internal representations, or the way in which reasoning about the relationship between new and existing internal representations may take place. Where we refer simply to 'representations', this should be taken as shorthand for 'external representations'.

Within cognitive psychology, a distinction is commonly made between graphical (or pictorial) representations and propositional (or linguistic, textual, sentential) ones. As we have already said, many different representations are used in providing a means for clients and users to check the requirements of a software system. These range from representations such as rich pictures (Patching, 1990) which can be characterised as broadly graphical, to formal logic, which is basically propositional. There are also representations of software systems which fall somewhere between the two extremes of graphical and propositional: for example data flow diagrams have some graphical elements, but also rely heavily on the use of propositional components. Thus the representations of interest in this paper include both graphical and

which they suggested should influence the developer's choice of requirements techniques and languages. These factors included the scope of a project, the need for understanding by various parties, and the volatility of requirements. More recently, Sommerville and Sawyer have listed some generic guidelines for choosing representations and methods (Sommerville & Sawyer 1997) and the RESPECT project has made some general recommendations as to the stages of system development for which certain techniques and representations are most appropriate (Maguire 1997). The RESPECT approach is data driven and based on user-centred design. A range of different methods are suggested to capture the required data in three main stages: user context analysis, feasibility and prototyping, and user requirements synthesis and validation. Other projects such as RESCUED (O'Neill, Johnson & Johnson, 1997) and the Evaluation Framework for Representations in Requirements Engineering (Sutcliffe, Maiden & Bright, 1997) are currently investigating the choice of representations at various stages of the software development process, while the work of Haywood and Dart (1996) and von Knethen et al. (1998) focuses on criteria for languages to specify software requirements. Finally, Brun and Beaudouin-Lafon have presented a taxonomy for the evaluation of formalisms intended specifically for specifying interactive systems (Brun and Beaudouin-Lafon, 1995). Their taxonomy includes three types of criteria, relating to expressive power, which they define as 'the ability to support the description of the largest possible set of characteristics of an interactive system', generative capabilities, defined as the ability to produce, for example, code, or interface functions, and extensibility and usability. The definition of ease of understanding which we adopt in this paper corresponds to one aspect of Brun and Beaudouin-Lafon's notion of usability.

Looking in more detail at the particular criteria or characteristics of languages which different authors have identified as being relevant in determining the suitability of a particular language for a particular project, we once again see that there is considerable variation in approach. Criteria for specifying languages in the STARTS guide (Department for Trade and Industry & National Computing Centre [DTI & NCC], 1987) incorporate qualities such as rigour, suitability for agreement with the end-user and assistance with structuring the requirements. Rigour comprises four separate features: how precisely the syntax of the language is defined, the extent to which it is underpinned by maths and logic, whether the meaning of individual symbols is defined, and the extent to which the language supports consistency checking of the requirements themselves. Suitability for end-user agreement refers to ease of understanding of the language by an untrained user, and assistance with structuring the requirements assesses the extent to which the language supports hierarchical decomposition and separation of concerns in the representation. The STARTS guide also regards the range of requirements covered by the language as important, including functional, performance, interface, system development and process requirements. Farbey (1993) covers criteria relating directly to languages, such as readability, modifiability and lack of ambiguity, and criteria relating to the language in use, such as the production of a well-presented specification, the cost in time to produce the representation, and the amount of support available. Davis (1993) also suggests a list of criteria pertaining to the effectiveness of representations and the choice of languages. Davis' list includes criteria relating directly to specifying languages, such as that the language should permit annotation and traceability, facilitate modification, and some that are expressed in terms of the software requirements specification (SRS). These

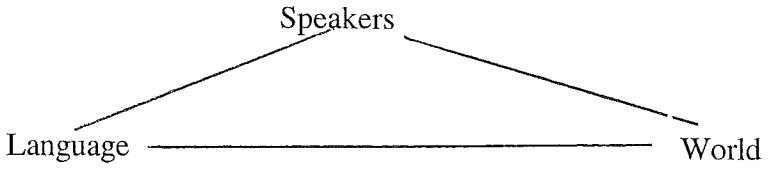

Figure 1: Key elements of understanding

Figure 1 denotes the fact that language is typically used by speakers to express their thoughts about the world. A person with a thought (such as 'I'd like a cup of coffee') can only communicate that thought to another person, such as the waiter in a café, through the use of some language - either verbal or visual (as in sign language). We say that the waiter is able to understand such a request if he can attach the intended meaning to it, or in other words, if he can know what thought or desire it expressed.

According to this view, understanding relies on a shared system of thoughts and meanings, one which all members of the linguistic community have (at least roughly) in common. In the above example, both the waiter and the original speaker would normally be familiar with the idea of a request and the fact that the words 'I'd like' may be used to express one. They would normally also have roughly the same idea about what kind of thing the words 'a cup of coffee' are used to denote. Thus Blackburn talks about a 'dog-legged' theory of meaning in which understanding between different members of a linguistic community is possible because that community shares a set of conventions about the mappings between words and things in the world, and also between words and thoughts.

For the purposes of our discussion, we may substitute the more general notion of 'representations' in place of 'language' in Figure 1. We may limit ourselves to the domain of software systems as the part of the world in which we are particularly interested, and think of our speakers as being software developers, clients, and potential system users, in other words, the stakeholders in a system development (see Figure 2).

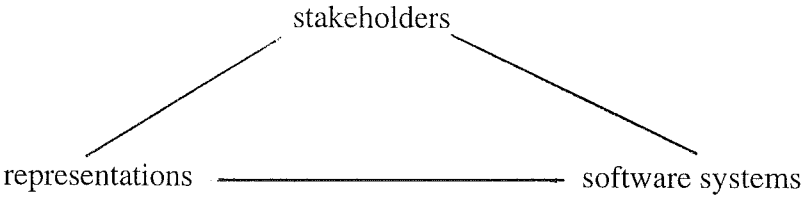

Figure 2: Key elements of understanding in interactive system development

Following on from the discussion above, we would say that understanding between the stakeholders would have to rely on a shared system of thoughts and meanings for representations; or in other words, a shared set of mappings between

The initial distinction is between semasiographic and glottographic languages. Semasiographic writing systems are means of visible communication which are independent of any particular spoken language. Semasiographic systems can be found in areas of public communication, such as road signs, cleaning instructions for clothes and directions for storing frozen food. Although it is not possible to read aloud a sign such as the one shown below, motorists, regardless of nationality, can see clearly that the sign indicates a no through road.

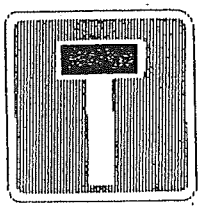

Figure 4: A semasiographic road sign

Well-developed semasiographic writing systems are often associated with primitive peoples, but this is not always the case. Mathematics is a highly sophisticated form of semasiography. As an example, we can consider the mathematical representation 91000, which translates as ninety-one thousand in English, and quatre-vingt-onze mille (four twenties and eleven thousand) in French. All three representations (maths, English and French) have different structures; there is repetition in the maths, but not in the other languages since 000 translates into one word in each. Mathematical symbolism is not tied to a specific spoken language. It is a semasiographic language that articulates thought directly and independently rather than merely standing for its spoken articulation.

In contrast to semasiographic, glottographic languages provide visible representations of spoken-language utterances. Glottographic writing systems are further divided into logographic (based on meaningful units) and phonographic (based on units of sound). Examples of logographic language are %, meaning 'per cent', and & meaning 'and'. It is clear that the logographic symbol & bears no relation to the phonographic a, n, d , since it is not meaningful to replace words such as land or candy with l& or c&y. One of the most common examples of a phonographic language (based on units of sound) is the international phonetic alphabet.

According to this classification, we would say that languages for specifying software systems are largely semasiographic, since their symbols relate directly to concepts, not to units in a specific language. However, text items which form part of a language (such as names of processes or data items) will be classed according to the natural language in which they are expressed and are most likely to be phonographic units.

6. A VIEW OF THE PROBLEM FROM COGNITIVE PSYCHOLOGY

The impact of different forms of representation on cognitive performance has long been recognised. This phenomenon is described by Simon as follows: 'Representations may be equivalent in the knowledge embedded in them without being equivalent in the power and speed of the inference processes they enable. They may be informationally equivalent without being computationally equivalent.' (Simon 1995)

We have said in section 2 that we focus in this paper on what have been called 'external representations'. External representations come in many different forms including maps, photographs, tables of figures, shopping lists, architects' blueprints and designers' doodles. As described in section 2, such representations have often been divided broadly into two categories: graphical and propositional. The distinctive characteristics of propositional and graphical representations are characterised in a little more detail by Eysenck and Keane (1990) as follows. Propositional representations are made up of discrete symbols (words or letters, depending on the level of granularity which is required), whereas in graphical representations, there are no obvious discrete symbols (for example, in a picture of a face, there is no obvious way of deciding what should be seen as a discrete symbol - should it be the representation of the face as a whole? or of the eye? or of the pupil?). In propositional representations, explicit symbols are used for everything which is to be represented, whereas in graphical representations much, for example the relation of 'on-ness' or 'beside-ness' is implicit. In propositional representations, sentences are arranged according to a grammar, whereas graphical representations usually follow no set grammar. Finally, propositional representations are in some sense abstract with respect to the form of perception (visual, auditory, tactile etc.) through which the information they portray could be communicated, whereas graphical representations are tied more firmly to the visual modality. Larkin and Simon (Larkin & Simon, 1987) also characterise the difference between what they call 'sentential' and 'diagrammatic' representations, drawing particular attention to the contrast between sequential nature of sentential representations and the indexing of information by location in a plane in diagrammatic representations.

As we have already said, many different representations are used in providing a means for clients and users to check the requirements of a software system. These range from representations such as rich pictures (Patching, 1990) which can be characterised as broadly graphical, to formal logic, which is basically propositional. In rich pictures, the symbols used may be drawn from an infinite set of possibilities, and it can be difficult to make a definite judgement about where one symbol begins and another ends. Rich pictures rely heavily on visual communication, and the location of symbols within the plane of the picture is often used in a significant way. Information can remain implicit in many parts of the representation. Finally, the grammar according to which rich pictures are drawn is only weakly defined. For formal logic on the other hand, there is a pre-defined set of symbols which may be used in writing a specification, all information to be communicated by a specification must be included explicitly,

arbitrary. The way in which analogy can be exploited in representations of software systems is at first sight not clear: as Fred Brooks has famously pointed out (Brooks, 1986), much of a software system is invisible, so direct visual analogies are not possible. The representations used in software development often aim to show, for example, the ordering of events, or the dependency of different outcomes on a range of alternative inputs, none of which correspond to obvious visible objects in the world around us - they are therefore what Larkin and Simon termed 'artificial diagrams'. Even in these cases, certain conventions of representation are so ingrained within the culture, that particular arrangements of symbols can immediately suggest what they represent, without the existence of a simple analogy with a physical object. For example, Winn (1993) has discovered that the simple vertical or horizontal alignment of symbols can affect whether readers judge one symbol to represent the 'cause' of the other, or a 'characteristic' of the other. He also draws attention to hierarchies, flow diagrams and tables as being forms of representation which are so familiar in our culture as to convey meaning directly without relying on a mapping to an actual physical object or arrangement.

As well as analogy or cultural familiarity of form, Winn also identifies the prominence or perceptual discriminability of symbols within a representation as being an important determinant of how easily meaning can be extracted. Discriminability relies on preattentive perceptual processes and is not affected by the reader's characteristics or goals. The phenomenon of certain symbols 'popping out' of a visual display because of their contrasting colour, shape or size has been demonstrated empirically and explained by Triesman's 'feature integration theory' (Triesman,1988). Winn (1993) suggests that the visual distinctiveness of symbols determines the order in which readers process components of a representation, their attention being drawn automatically to the most prominent symbols first.

Finally, both Larkin and Simon (1987) and Winn (1993) have identified the importance of spatial layout in facilitating the search for relevant information within a representation. Preattentive processes may also play a role in the reader's identification of groups of symbols which are clustered together in the same location. Thus the particular arrangement of individual symbols may lead to the representation as a whole possessing certain 'emergent properties' which may themselves carry meaning.

The way in which different representations support particular kinds of reasoning is more complicated and less well understood. Bauer and Johnson-Laird (1993) showed that particular forms of diagrams can facilitate reasoning about double disjunctions if they are laid out in such as way as to preserve important topographical features of the problem (note that this relates to the point about spatial layout above), and also assisted reasoners in keeping track of alternative possible solutions to a problem by making those possibilities explicit. Cox and Brna (1993) have looked at constraint-satisfaction problems and suggest that an important factor here is the extent to which a representation is capable of representing an appropriate amount of abstraction or indeterminacy. Finally, Stenning and Oberlander (1995), investigating syllogistic reasoning, have identified the important characteristic of representations in supporting reasoning as 'specificity': the extent to which representations limit abstraction and thereby aid processibility. Stenning and Oberlander claim that the advantage commonly

The framework of cognitive dimensions (Green 1989, 1991) was developed for the analytical evaluation of information structures and has been used successfully in a number of cases, both by Green himself (Green, 1991; Green, Petre & Bellamy, 1991; Modugno, Green, & Myers, 1994; Green & Blackwell, 1996), and by other authors (Shum, 1991; Petre, 1995; Stacey, 1995; Roast, 1997; Yang et al.,1997). We have, ourselves, found cognitive dimensions to be useful and effective in evaluating two different mechanisms for structuring specifications written in Z (Britton, Jones & Lam, 1998).

Cognitive dimensions are intended to support the evaluation of any type of information structure and can be applied to specification and programming languages, musical scores or even telephone numbers (Modugno, Green & Myers, 1994). The dimensions are not a set of criteria which may be satisfied to various degrees by different information structures; rather they are aspects of information structures which may be important and useful in specific situations. Green's work in this area has not, as yet, been theoretically validated; nor does it address the question of objective measures of cognitive dimensions. The framework does, however, provide a pragmatic approach to considering information structures (in our case, specification languages). Cognitive dimensions are tools for thinking about information structures, rather than detailed guidelines; they provide a ready-made vocabulary which makes them useful, both as a thinking tool and an index to the professional literature.

Ease of understanding is not, itself, part of Green's framework, but several of the cognitive dimensions helped us to identify ideas in the cognitive psychology literature that are relevant to ease of understanding. This was helpful in our research precisely because we wanted to break down the concept of ease of understanding in order to get a grasp of how it may be supported by specification languages. We found the dimensions to be especially helpful in focusing on the activities that a reader has to carry out in the context which is the subject of this paper (the validation of representations of software requirements) and the properties of languages which may support these activities. The dimensions of visibility and redundant recoding highlighted relevant work in the cognitive psychology literature that helped us to formulate two properties that may help readers to distinguish elements of a representation clearly: discriminability of symbols and the extent to which a language allows exploitation of human visual perception. The dimensions of closeness of mapping and role expressiveness led to consideration of two of the tasks that a reader has to perform: relating elements of the representation to elements in the domain and inferring the purpose of individual components in a representation. In order to provide support for these activities, we formulated the property of motivation of symbols in a language. On the other hand, the dimensions of hard mental operations and hidden dependencies relate to activities that we would not wish to impose on a reader of a representation; this led us to thinking about properties of languages which could help reduce the need for these activities, such as the number of different symbols in a language, the extent to which a language allows exploitation of human visual perception and the amount of structure in the language. The property of structure also resulted from consideration of the dimension of abstraction gradient and the associated task of decomposing a representation into manageable chunks. Finally, the dimension of consistency (where similar information is expressed in similar ways) lead directly to the property of consistency of symbols in a language.

actually makes them easier to reason with. To achieve a complete view of a system, a range of languages has to be used in order to make explicit all the different dependencies.

Visibility

The ability to view components of a representation easily and to distinguish them from each other is valuable in providing the reader with confidence that he or she will be able to understand the representation. First impressions can have a significant effect on the way in which a reader (particularly one who is unfamiliar with the language) approaches the tasks of searching the representation, recognising relevant information and validating the content of the representation. One important factor in such first impressions is the perceptual discriminability of symbols within the representation, as discussed in section 6. A textual representation which is presented in a single block is much more off-putting than the same text split into separate sections according to the different topics covered. Among mathematical languages, the schema boxes in Z help to increase visibility by physically packaging together associated parts of the representation. Graphical languages, such as entity-relationship or data flow diagrams, appear to offer the greatest potential for visibility, but frequently produce diagrams which are a cluttered mish-mash of icons and symbols, because not enough care has been taken to emphasise visibility of components in the representation. One of the most useful aspects of the cognitive dimensions approach to evaluation of languages is that it highlights the trade-offs that have to be made between dimensions in different situations. Visibility of components is often a casualty in cases where priority is given to ensuring that the representation carries as much information as possible. This cognitive dimension implies that, in cases where visibility is important in a representation, a language should be used which has symbols that are simple to recognise and easy to distinguish from each other. We discuss this further in section 8.

Redundant Recoding

A cognitive dimension which is related to visibility is that of redundant recoding: the ability to express information in a representation in more than one way. Examples of redundant recoding in text-based representations include headings, font size, and use of upper and lower case. Colour or the shape of the representation itself may be used in graphical or textual representations to highlight separate parts of a system or to emphasise aspects of its functionality. In certain languages redundant recoding is provided by indentation and layout. Figure 6(a) and (b) below shows two versions of the same process description: one using unformatted, lower case structured English and one using comment, indentation and a mixture of upper and lower cases. In the second example the decisions to be taken are conveyed by both the content and the layout of the representation.

provided by a language. According to Sengler (1983) reading representations of any size involves the activities of decomposition (to split the representation into manageable chunks) and abstraction (to identify the most important features) as described in section 6. A language which scores highly on the cognitive dimension of abstraction gradient is one which encourages developers to produce representations for which the reader does not have to expend effort in working out how to abstract the most important information from it. This again relates to Stenning and Oberlander's point about the importance of 'specificity', or clarity in abstraction. One important way of imposing a particular view of abstraction on the reader is by physically structuring a representation into visible chunks. The amount of structure is an important property in a language; it is discussed below in section 8.

Consistency

A consistent language is one where similar information is expressed in similar ways. This not only simplifies representations produced using the language, but means that untrained users can make reasonable guesses at symbols which they have not seen before. One example of consistency in mathematics is the use of the symbol $<$ to mean "less than" and the symbol $\leq$, to mean "less than or equal to". In section 8 we discuss how consistency of symbols can alleviate problems of understanding in languages which have a large number of symbols.

Role Expressiveness

In languages which score well on the dimension of role expressiveness the reader can readily infer the purpose of individual components in a representation. Closeness of mapping between language and problem domain provides support for role expressiveness, since the reader will recognise the purposes of different components of the representation based on his or her own experience of the problem domain. As already mentioned above, there is rarely a direct mapping between symbols used in languages for software development and the domain elements they represent. This means that role expressiveness depends largely on how names and labels are used in representations. In a data flow diagram, for example, the name given to a process or data store is crucial in conveying the purpose of that process or store to the reader. In a Z specification the purpose of a schema is encapsulated in its name, so it is important that care is taken to make sure this is meaningful.

Hard Mental Operations

Some languages require readers of representations to perform mental operations that are extremely complex, especially for people who are unfamiliar with the languages. Examples can be found in many of the symbols used in Z, and in the aggregation and inheritance relationships used in object-oriented modelling. Hard mental operations are a further barrier for readers of a representation who are unfamiliar with the language in which it is written.

8.1 The Role of Measurement

It is our view that an approach based on measurement helps us to understand and elucidate the concept of understandability of languages. The process of attempting to measure something is valuable in itself, in that it forces the would-be measurer to articulate exactly what is being measured. In the words of Lord Kelvin (1889) "When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it in numbers, your knowledge is of a meagre and unsatisfactory kind". In the area of software specification, Farbey (1993) and Davis et al. (1993) have attempted to measure aspects of the software requirements specification itself. In (Davis 1988) languages used in software specification, including natural language, are measured against a set of criteria, scored on a subjective scale of 0 to 10. In our own work on understanding in software specification languages, we have, to date, already identified some direct and empirical ways of measuring languages in terms of the six properties listed above. At the very least, we can measure on a nominal scale, to ascertain whether or not a language possesses a certain property. In certain cases, we can use an ordinal scale to compare languages, to indicate the relative degrees to which two or more languages possess a particular property. While these basic measurements are useful in themselves, rapid advances in metrics in a wide variety of areas lead us to believe that there is considerable potential for more sophisticated measurements of languages in the future, which will, in turn, lead to a deeper knowledge of concepts such as ease of understanding.

8.2 Properties of Languages for Specifying Software

The Number of Different Symbols in the Language

One obvious property of languages which can be measured is the number of symbols they contain. The number of pre-defined symbols in a language has an impact on three of the cognitive dimensions described in section 7.1: redundant recoding, consistency and hard mental operations. Green (1980) makes the point that programming languages with a large number of symbols and features are more difficult to learn and understand than languages with fewer features. If we apply this to languages used in software specification, it follows that languages such as storyboard or rich pictures, each with 2 symbols, are apparently going to be easier for new users to understand than languages such as Z, which has 76 symbols (Myers et al 1996). It is not surprising then that storyboards are widely used in the development of, for example, interactive multimedia systems, where not only the users, but also members of the development team, are likely to be unfamiliar with software specification languages (Admiral Training 1995, Britton et al 1997). Figure 7, below, shows the huge difference in the numbers of symbols found in software specification languages.

language, Z, has 76 different symbols (see Figure 7, above); of these, 28 symbols are consistent in that they have a direct connection in form and meaning with at least one other symbol. The consistent symbols include = ("is equal to") and ≠ ("is not equal to"), ◁ ("domain restriction") and ⩤ ("domain subtraction"), ⊆ ("subset") and ⊂ ("proper subset"). However, 48 of the Z symbols do not exhibit any consistency; there is no obvious way of grouping them according to meaning and form. We would expect that this contributes to the difficulties that new users have in understanding representations written in Z.

We should also be aware, however, that it may be difficult for users to discriminate between symbols which are consistent, such as the examples in the preceding paragraph. For example, a user who is not familiar with the Z language may recall that ◁ and ▷ mean "domain restriction" and "range restriction", but may be unable to remember exactly which is which. In some contexts there is a trade-off between the properties of clear consistency of symbols and discriminability (see below). Links and trade-offs between properties of languages are identified as future work in section 10.

The Discriminability of the Symbols in a Language

Discriminability, which is related to the cognitive dimensions of visibility and redundant recoding, refers to the ease with which different symbols in a language can be distinguished from each other; this depends on how physically distinct each symbol is from others in the language. Discriminability has been identified by research in cognitive psychology (see section 6), and by both Green (1980) and Sampson (1985) as an indicator of how easy a language will be for untrained users to understand. As an example, we can consider the Z symbols ⇸ (meaning partial function) and ↦ (meaning maplet). To someone who is not familiar with Z, the two symbols look very similar, yet they have completely different meanings. They are not easy to distinguish from each other and so are a potential cause of confusion for readers trying to understand a specification written in Z.

Discriminability of symbols is a problem with many mathematical languages. In contrast, diagrammatic languages, such as entity-relationship and data flow diagrams, generally have symbols, such as lines, arrows and boxes, that are clearly distinguishable from each other. However, confusion may arise when someone who is unfamiliar with the languages has to look at more than one type of diagram. An untrained user may well find that the symbol for a process in a data flow diagram is not easy to distinguish from the symbol for a data store or the symbol for an entity in the E-R diagram (see Figure 8 below).

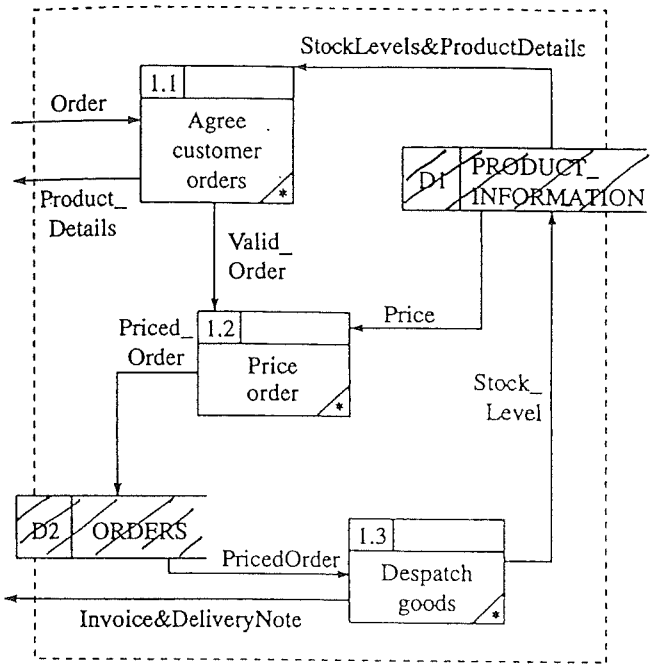

Figure 9: Use of shading to distinguish symbols in a data flow diagram

As we mentioned above, there is frequently a trade-off between a language in which the symbols are easily discriminable and one in which the symbols show a high degree of consistency. It is not possible to advocate a general rule as to which of the two properties contributes more to the ease of understanding of a language, but it is important that developers are aware of the issue when choosing languages and constructing representations.

In section 9, below, we report on a study to measure discriminability in the Z formal specification language.

The Degree of Motivation of the Symbols in the Language

The concept of motivated and arbitrary languages comes from the work of Sampson (1985) (see section 5, above). Motivation is also identified as an important characteristic by many authors in the discipline of cognitive psychology (see section 6). A language may be considered to be motivated if there exists a natural relationship between the elements of the language and objects or ideas that they represent. The property of motivation of symbols is

Although it is not currently feasible to measure motivation directly, we can, however, measure motivation of symbols in a language indirectly by means of experiments, such as those used in choosing icons, where subjects are asked to identify the images which map most closely onto concepts of interest. An example of this is discussed in section 9, below. Motivation is an important property of a language, since, by definition, it means that the symbols in the language will be closely related to the concepts that they represent and therefore that their meaning will be clear, even to untrained users. We would expect, therefore, that a language with a large number of motivated symbols will produce representations that are easy for users to understand, even if the users are unfamiliar with the language.

The Extent to which the Language allows Exploitation of Human Visual Perception.

Cognitive dimensions relevant to this property of languages are redundant recoding, hidden dependencies and visibility . This property is also related both to Mackinlay's (1986) notion of effectiveness: whether a language "exploits the capabilities of the output medium and the human visual system", and to what Scaife and Rogers (1996) term 'computational offloading'. Computational offloading is the extent to which a representation of a problem can reduce the amount of cognitive effort needed to solve the problem by providing the means for direct perceptual recognition of important elements in it. In this paper we refer to languages which allow exploitation of human visual perception as 'perceptual'.

Perceptual representations are those in which we perceive meaning directly, for example the use of colour in electricity cables, or diagrams in various contexts, such as road signs. Most languages used to specify software contain both perceptual and non-perceptual elements. Graphs are perceptual in that they are diagrammatic, although they often contain (non-perceptual) labels for the various nodes. Representations in languages which are not perceptual in themselves are frequently laid out to show certain aspects perceptually: the placing of schemas in Z specifications, formatting of pseudo code and introduction of 'white space' to aid comprehension are examples of this. Text-based languages generally use perceptual devices to aid comprehension; these may include section headings, paragraphs, variations in size and font, bullets, emphasis and white space.

The question of how to exploit human visual perception is one which has long concerned designers of representations in all fields. As long ago as 1935, Jan Tschichold noted that ".... readers want what is important to be clearly laid out. They will not read anything that is troublesome to read, but are pleased with what looks clear and well arranged, for it will make their task of understanding easier", (cited in Williams 1994).

Perceptual and non-perceptual languages each have advantages and disadvantages in software specification. Representations in languages which are not in themselves perceptual, such as mathematical specification languages, are generally able to hold much more information than representations which rely heavily on perceptual features, but are frequently beyond the understanding

entities and data stores can be easily confused by readers who are new to the language, and it can be difficult to distinguish between names of data flows, such as 'Order', 'Valid_Order', 'Part_Order, Incomplete_Order'.

Repetition means repeating some aspect of the design throughout the representation to give the impression of coherence and organisation. In a set of data flow diagrams this can be achieved by labelling each level in exactly the same way. In individual diagrams, the same font should be repeated for names of the same type of elements, such as processes. This can cause a problem, however, if a process has a particularly long name and the process symbol needs to be made larger, since it may wrongly appear to be particularly important.

The technique of alignment aims to ensure that nothing is placed randomly in a representation. Each element should, where possible, have a strong visual connection with something else in the representation and the overall structure should appear to be a cohesive unit. In complex data flow diagrams space is often at a premium, which means that components of the representation are placed where there is room, rather than in a position which emphasises their relationship to other elements in the representation. In Figure 9 (above) it would be clearer if the three processes 'Agree customer orders', 'Price order' and 'Dispatch goods', were aligned to reflect the path of an order through the system. They have had to be placed unaligned on the page to allow room for the 'ORDERS' data store. If we attempt to move the 'ORDERS' data store to the other side of the diagram, aligned with the 'PRODUCT_INFORMATION' store, we get crossed data flows, which are prohibited in a data flow diagram.

Proximity means that elements of a representation that are related to each other should be grouped close together, as mentioned above. Again, in data flow diagrams, lack of space means that this clustering is often impossible. A further problem arises from the fact that external entities are only shown on the top level diagram; this means that, even if an external entity is very closely related to a lower level process, they would not appear on the same diagram.

One of the principal claims of data flow diagrams is that they are easy for users to understand; data flow is, in fact, so widely considered as easy to understand that it is still the first specification language taught to students on many system development courses. From this brief assessment of the data flow language in the light of the four techniques for making representations more perceptual we can see that, in spite of its graphic components, data flow gives the developer very little freedom to exploit human visual perception. Although, in theory, the data flow language is flexible enough to produce perceptually rich representations, in practice it is often difficult for developers to incorporate these four techniques into their diagrams.

The Amount of Structure Inherent in the Language

As discussed in section 6, above, finding, decomposing and abstracting information have been identified as key activities in reading a representation (Larkin & Simon, 1987, Sengler, 1983). Finding information involves searching the representation; this process will be easier for untrained users to carry out if the language used encourages a clear structure in the representation. Decomposition and abstraction are important because a reader can only cope

relevant to ease of understanding, and then to find a way of measuring the languages in terms of these properties. In the previous section we identified and described six properties of specification languages that, in our view, may contribute to the ease of understanding of representations. In this section we describe a small-scale pilot study , which we have carried out to explore the potential for empirical evaluation of the languages in terms of the properties identified in section 8. The aims of the study are, first, to investigate whether three of the properties (consistency, discriminability and motivation) can be measured empirically in a reliable and repeatable way, and, second, to obtain some first (empirical) indications as to whether our properties may plausibly be predictors of ease of understanding of representations written in Z.

There are a number of approaches that we could take in an empirical study based on our work. We could, for example, select a specific problem, apply different specification languages to it and then analyse and compare the resulting representations of the problem. However, as we have already said in the introduction to this paper, we share the opinion of certain other authors (Stenning & Oberlander, 1995; Scaife and Rogers, 1996) that evaluating representations themselves is too problematic. There are too many unknown and uncontrollable variables associated with a representation for the results of such an evaluation to be of real use. Our response to this is to focus on the empirical evaluation of the languages themselves, concentrating, initially, on the extent to which the properties are amenable to objective measurement. We intend, in future, to carry out more empirical studies which will investigate whether our properties really are good indicators of understandability of representations, for example by applying the properties analytically to predict understandability and then testing out our predictions in empirical studies.

For the purposes of this exploratory study, we have chosen the language Z as an example of the kind of language whose properties we would hope to be able to measure, and ultimately to use, in predicting the extent to which it will support the construction of representations which are understandable for untrained users. Although the choice of Z has been made partly for pragmatic reasons (as students in our department are taught Z on a routine basis, we have access to a large number of potential subjects), we argue that the choice can also be justified on theoretical grounds. Z is generally agreed to be a very difficult language for untrained users to understand. However, we can often learn more from 'failures' or difficult cases than from cases in which everything goes well. In Z, we may find many things which make a language difficult for untrained users: we believe we may learn more by observing students' problems with trying to understand representations written in Z than we would from considering representations in which understanding is simply intuitive. Furthermore, although Z would usually not be the obvious choice for writing software specifications which are to be checked by users, such checking of formal specifications is sometimes required in the development of safety-critical systems (see, for example, McCluskey et al., 1995; Johnson, McCarthy & Wright, 1995).

We have focused, in our pilot study, on the properties of consistency, discriminability and motivation, which relate to individual symbols in the languages. The number of symbols in a language can be measured directly as described above, and will not be discussed any further here. The previous

Symbol presented with meaning	Options presented with totals for each		
1. If $\in$ means "is a member of", which of the following symbols do you think means "is not a member of"?	$\subset$	$\subseteq$	$\notin$
	1		**24**
2. If ? represents "input", which of the following symbols do you think represents "output"?	??	!	$\rightarrow$
	1	**9**	13
3. If $\mathbb{N}$ represents positive numbers including 0, which of the following symbols do you think represents positive numbers not including 0?	$\mathbb{Z}$	$\mathbb{R}$	$\mathbb{N}^+$
	8	5	**11**
4. If $\subseteq$ means "subset", which of the following symbols do you think means "proper subset"?	$\subset$	$\notin$	$\mathbb{Z}$
	23	2	

Figure 12 : Numbers of students choosing symbols to represent meanings relating to a given example (correct answers are highlighted).

In questions 1 and 4, consistency apparently provided a strong cue for the meaning of new symbols, as given the meaning for one symbol, most subjects could correctly identify the symbol corresponding to a related meaning (although it should be noted that some of the subjects may have encountered these symbols during school maths programmes and may have been acting on the basis of memory for their meaning, rather than relying purely on the visual consistency of the symbols).

In questions 2 and 3, the responses are much more divided. Our hypothesis regarding question 2 is that use of the symbol $\rightarrow$ to represent the idea of "output" is motivated and that for a large proportion of students, this was the determining factor in this choice of symbol over and above any consideration of consistency. This suggests that the effects of consistency may need to be considered in conjunction with those of motivation, as well as discriminability.

In question 3, the relationship between the $\mathbb{N}$ and $\mathbb{N}^+$ symbols was apparently not strong enough to guide subjects to the correct symbol for the meaning given. This suggests that attempts to exploit consistency must be treated with caution: some relationships between symbols which may be obvious to the designer of a language may not be apparent to all its intended users.

	Options presented with numbers of guesses									
⩤	is defined as	domain subtrac-tion	**range subtr-action**	domain restrict-ion	member of	subset of	override	range restrict-ion	not a member of	proper subset of
		5	**6**	3						
⊆	subset of	domain restrict-ion	is defined as	range restrict-ion	member of	not a member of	domain subtract-ion	override	range subtract-ion	**proper subset of**
	6									**8**
⩥	range restrict-ion	domain restrict-ion	subset of	not a member of	domain subtract-ion	member of	override	proper subset of	**is defined as**	range subtract-ion
			1	1		1	1		**10**	
◁	member of	is defined as	proper subset of	not a member of	domain subtractio n	**domain restrict-ion**	range restrict-ion	override	range subtract-ion	subset of
					2	**5**	6		1	
⊕	range restrict-ion	domain restrict-ion	subset of	not a member of	domain subtract-ion	member of	**override**	proper subset of	is defined as	range subtract-ion
							14			
∉		domain subtract-ion	range subtract-ion	domain restrict-ion	member of	subset of	override	range restrict-ion	**not a member of**	proper subset of
									14	
▷	subset of	domain restrict-ion	is defined as	**range restrict-ion**	member of	not a member of	domain subtract-ion	override	range subtract-ion	proper subset of
		6		**5**			2		1	
∈	range restrict-ion	domain restrict-ion	subset of	not a member of	domain subtract-ion	**member of**	override	proper subset of	is defined as	range subtract-ion
						14				
⩤	override	is defined as	**domain subtra-ction**	range restrict-ion	member of	subset of	range subtract-ion	domain restrict-ion	not a member of	proper subset of
			6	4			4			
⊂	**subset of**	domain restrict-ion	is defined as	range restrict-ion	member of	not a member of	domain subtract-ion	override	range subtract-ion	proper subset of
	10									4

Figure 13(b) : Numbers of students choosing meanings for symbols given in Z (correct answers are highlighted)

The aim of the questions in Figure 13 (a) was to see if subjects confused either the meanings of perceptually similar symbols in Z, such as the arrows ⇸ (partial function), → (total function), ↦ (maplet) and ↔ (relation), or the meanings of perceptually dissimilar but conceptually related symbols such as ? (input data) and ! (output data), and Δ (denoting an operation which will change data) and Ξ (denoting an operation which will not change data).

It can be seen from the first, third, fifth and last rows in Figure 13 (a) that perceptually similar symbols were in fact confused; in each case, the subjects either identified the correct meaning for the symbol given, or selected a meaning that belongs to one of the other similar symbols. However, what is not clear at this stage, is whether this is due to problems with the perceptual

Meaning	Options presented with numbers of guesses								
and	⇒	–	∨	⇔	∧	∃	¬	&	∀
					1			**24**	
if .. and only if	∃	⇒	∨	–	&	∀	¬	⇔	∧
	2	4				4	6	**6**	2
not	∀	⇔	¬	∃	–	&	∧	⇒	∨
		1	**2**	5	**4**		8	1	4
all	&	–	∀	⇒	∃	¬	∧	∨	⇔
	1	2	**9**	3	1	1	2		6
there exists	⇒	⇔	∧	¬	∀	∨	∃	–	&
	3	3		2	4	3	**6**	4	
or	∨	&	⇒	⇔	∧	¬	∃	∀	–
	8		2	8		1	1		2
if .. then	–	∧	&	∨	⇔	¬	∀	⇒	∃
	4	3		1	2	7	1	**5**	

Figure 14 : Numbers of students choosing symbols in logic to represent meanings given (correct answers are highlighted)

It is noticeable from Figure 14 that all but one subject in Group 1 chose the symbol & as an appropriate representation for "and", and that & was chosen by only one student to mean anything other than "and". Unfortunately however, the symbol used in Z to represent 'and' is '∧'. This appears not to be very motivated as only one subject chose this symbol to mean 'and', and opinion appears also to have been quite divided regarding the other meanings with which it was associated. On this basis, we may suppose that the use of the symbol '&' in logic is highly motivated, but that other symbols used in logic and Z will not easily be associated with their intended meanings by untrained users.

Group 2 had followed a course on formal languages the previous year, in which they were taught Z as well as one of the versions of logic from which symbols were drawn - in the version of logic students were taught, the symbol '∧' was used to denote 'and', and the symbol '¬' was used to denote 'not'. These subjects were given the same set of symbols as were shown to Group 1, and were asked to identify their meanings. For example, given the symbol ∃, subjects were asked to say whether this means "and", "for all", "there exists", "if .. then", "or", "if .. and only if", or "not". Results from this question are shown in Figure 15.

identified than those of untaught symbols which are not motivated. Furthermore, attempting to teach the meaning of symbols which are unmotivated may not be as effective as starting with symbols which are motivated in assisting subjects to identify their correct meanings.

We also note that all subjects in Group 2 correctly identified the meanings of $\exists$ and $\forall$. From the results of Group 1, the symbols $\exists$ and $\forall$ do not obviously appear to be motivated (these symbols were only correctly associated with their meanings by around one quarter and one third of the subjects respectively). However, the motivation of these symbols ($\exists$ is a backwards E, as in "Exists", and $\forall$ is an inverted A, as in "All") had been pointed out to the subjects in Group 2 during their course the previous year. The results shown in Figure 15 suggest that, once attention has been drawn to the motivation of a symbol, its meaning may quite readily be recalled, even if, at first sight, it does not appear to be motivated. Once again, these results provide interesting pointers to further work on the nature of motivation and its effect on ease of understanding.

9.4 Summary of Findings

The results of our pilot study indicate that we have achieved some success in our attempts to derive measures of languages which may be used to predict understandability of representations. We have some measures, though we do not yet know exactly how to use them, nor how to combine them. For example, we do not know at present how exactly ease of understanding of representations written in Z is affected by the fact that two sets of four symbols amongst the 76 available in Z are easily confused with each other. Considering the properties in combination, we do not know whether consistency of symbols is more or less significant than discriminability or motivation, nor what the relative weightings of each of the properties should be in any calculation of understandability. However, our study has produced some interesting results that encourage us to have some confidence that our properties might reasonably be indicators of ease of understanding.

The results of the study suggest that the kind of consistency which exists between some symbols in Z may be useful in helping untrained users to correctly identify the meanings of some symbols once they know the meanings of others. On the other hand, the results suggest that lack of discriminability between symbols causes problems in Z. In particular the arrows, such as $\twoheadrightarrow$, $\rightarrow$, $\mapsto$ and $\leftrightarrow$, and symbols for domain and range restriction and subtraction, $\rhd$, $\lhd$, $\rhd$ and $\lhd$ appear to be sources of confusion, although it is not yet clear whether this confusion may be due mainly to the perceptual similarity between symbols, or to their close relations in meaning. Finally, none of the symbols from Z which were included in our pilot study appear to be very motivated for completely untrained users, although once meaning has been taught and motivations identified, their meanings can be correctly recalled.

We have already seen, in section 8, that Z has a relatively high number of symbols, that it provides little basis for exploiting the capabilities of visual

and choices made. For each of the properties of languages identified, we have suggested ways in which direct or empirical measures relating to these properties can be obtained. For the number of symbols in a language and the degree of consistency of the symbols we have identified metrics against which languages can be measured. We have presented empirical evidence from our study to suggest that problems with discriminability can lead to inaccurate memory for meaning and therefore decrease the possibility of understanding a representation. Results from the study also suggest that high motivation is associated with improved recall of meanings of symbols, and will therefore contribute to ease of understanding. We have identified certain properties of software specification languages that exploit human visual perception (connectedness and inclusion), and we have also suggested how we may check whether a language gives the developer freedom to exploit recognised design techniques. For the amount of structure in a language we have referenced existing research on measuring and comparing tree structures in different languages.

From the literature and from our study we have generated a number of hypotheses about properties of languages which make representations constructed using the languages easier to understand. The impact of the number of symbols in a language is discussed in section 8.2 , which also shows how consistency of symbols may alleviate some of the problems which arise when a language has a large number of symbols, and considers the importance of discriminability of symbols. Although our study identified problems with discriminability of symbols, it is not clear whether this was due to lack of perceptual discriminability or to conceptual confusion; this is discussed in section 9.2. Motivation of symbols is a feature that is considered as important for ease of understanding both in Sampson (1985) and in the cognitive psychology literature (see section 6, above). However, as we note in our discussion of this property, motivation is virtually impossible to achieve in the area of software specification because many of the main elements of concern, such as processes, data and changes of state are invisible and intangible. Perhaps more important in the development of software systems is whether the reader has the relevant concepts of, for example, objects, data flow or state to which he or she can relate components of the language. A further property of a language that is grounded in the cognitive psychology literature is the property of exploiting human visual perception . The concern here is that languages should have the flexibility to allow developers to exploit human perceptual capabilities in a way which is appropriate to the domain in which representations are being used. For example, where a domain requires discussion of categorisation, the language should allow developers to use factors, such as spatial layout, which suggest a conceptual grouping to the reader. Finally, we suggest that it is important for a specification language to provide a clear, easily visible structure for representations, since, in the absence of a given structure, readers will have to waste time and effort in constructing one for themselves. We argue that any structure is better than none, since it helps the reader to 'chunk' the information presented. Over and above this, a structure will be particularly effective if it in some way reflects the structure of information in the domain, or if it provides the reader with useful abstractions to reason with (Stenning & Oberlander 1995).

empirical studies which will confirm this (or otherwise). Only once we are happy that we have a good set of measures, will we be in a position to do such studies in order to fully validate our properties in the sense of checking whether they really are good indicators of understandability of representations.

One approach which we hope to follow up is to compare results from our work with expert opinion on the ease of understanding of different languages. A survey of developers who regularly design systems for clients and users who are unfamiliar with software languages (similar to that reported in Britton et al. 1997) would also provide useful information. We would like to pursue work on how to achieve more exact measures of the properties and how to use the measures to produce precise and useful results. Is it possible, for example, to calculate a reliable 'understandability quotient' of a language? Finally, we have chosen a very specific context for our research to date: the validation of representations of software requirements. We recognise that understanding in related contexts may involve more and different activities on the part of the user, such as the ability to manipulate and modify the representations. Future work will examine whether more properties of languages are relevant in this case and what those extra properties might be.

NOTES

Background. This article is based on a paper presented at the International Workshop on Representations in Interactive Software Development, held at Queen Mary and Westfield College, University of London in July, 1997.

Acknowledgements. The authors would like to thank the organisers and participants in the Workshop and the anonymous referees for their helpful and constructive comments on earlier versions of this paper. We are also indebted to two of our colleagues at the University of Hertfordshire: Richard Young for discussions on various aspects of the paper and Maria Kutar for help in producing the manuscript.

Authors' Addresses. Both authors can be contacted at the Department of Computer Science, University of Hertfordshire, College Lane, Hatfield, Herts. AL10 9AB, UK Email: C.Britton, S.Jones@herts.ac.uk

Davis, A. et al. (1993). Identifying and measuring quality in a software requirements specification. *Proceedings of the First International Software Metrics Symposium*, 141-152. IEEE Computer Society Press.

Department for Trade and Industry and National Computing Centre (1987). *The STARTS guide* (2nd ed.), volume 1, NCC Publications, UK.

Eysenck, M. & Keane, M. (1990). *Cognitive psychology: A student's handbook.*. Lawrence Erlbaum Associates.

Farbey, B. (1993). Software quality metrics: considerations about requirements and requirement specifications. In R. Thayer & A. McGetterick (Eds.), *Software engineering: a European perspective* (pp.138-142). IEEE Computer Society Press.

Fenton, N. & Hill, G. (1993). *Systems construction and analysis: A mathematical and logical framework.* McGraw Hill.

Fertuck, L. (1992). *Systems analysis and design.* Wm. C Brown Publishers.

Fitter, M. & Green, T. (1981). When do diagrams make good computer languages? In J. Alty & M. Coombs (Eds.), *Computing skills and the user interface.* Academic Press.

Garzotto, F., Mainetti, L. & Paolini, P. (1995). Hypermedia design, analysis and evaluation issues. *Communications of the ACM, 38 (8)*

Green, T. (1980). Programming as a cognitive activity. In H. Smith & T. Green (Eds.) *Human interaction with computers.* Academic Press.

Green, T. (1983). Learning big and little programming languages. In A. Wilkinson (Ed.), *Classroom computers and cognitive science.* Academic Press, New York.

Green, T. (1989). Cognitive dimensions of notations. In A. Sutcliffe & L. Macaulay (Eds.), *People and Computers V, Proceedings of HCI'89.* Cambridge University Press.

Green, T. (1991). Describing information artefacts with cognitive dimensions and structure maps. In D. Diaper, & N. Hammond (Eds.), *People and Computers VI, Proceedings of HCI'91.* Cambridge University Press.

Green, T. & Blackwell, A. (1996). Thinking about visual programs. In *Thinking with diagrams* (IEE Colloquium Digest No: 96/010). Institute for Electronic Engineers, London.

Green, T., Petre, M. & Bellamy, R. (1991). Comprehensibility of visual and textual programs: A test of superlativism against the "Match Mismatch" conjecture. In J. Koenemann-Belliveau, T. Moher & S. Robertson (Eds.), *Empirical studies of programmers* 121-146. Norwood NJ, Ablex.

Patching, D. (1990). *Practical soft systems analysis.* Pitman Publishing.

Petre, M. (1995). Why looking isn't always seeing: Readership skills and graphical programming. *Communications of the ACM, 38 (6).*

Riding, R. & Cheema, I. (1991). Cognitive skills: An overview and integration. *Educational Psychology, 11*, 193-215.

Roast, C. (1997). Formally comparing and informing notation design. In H.Thimblely, B. O'Conaill & P. Thomas (Eds.), *People and Computers XII, Proceedings of HCI'97.* Springer.

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. & Lorensen, W. (1991) *Object-Oriented Modeling and Design.*. Englewod Cliffs, N.J.: Prentice Hall.

Sampson, G. (1985).*Writing systems.* Hutchinson.

Scaife, M. & Rogers, Y. (1996). External cognition: How do graphical representations work? *International Journal of Human-Computer Studies, 45*, 185-213.

Sengler, H. (1983). A model of program understanding. In T. Green, S. Payne. & G. van der Veer, (Eds.), *The Psychology of Computer Use.* Academic Press.

Shum, S. (1991). Cognitive Dimensions of Design Rationale. In D. Diaper, & N. Hammond (Eds.), *People and Computers VI, Proceedings of HCI'91.* Cambridge University Press.

Simon, H. (1995). Forward. In J. Glasgow, N. Nari Narayanan & B. Chandrasekaran (Eds.), *Diagrammatic reasoning: Cognitive and computational perspective.* MIT Press.

Sommerville, I. (1995). *Software engineering* (5th ed). Addison Wesley.

Sommerville, I. & Sawyer, P. (1997). *Requirements engineering: A good practice guide.* Wiley.

Spivey, J. (1989). *The Z notation: A reference manual.* Prentice Hall.

Stenning, K. & Oberlander, J. (1995). A cognitive theory of graphical and linguistic reasoning: Logic and implementation. *Cognitive Science, 19*, 97-140.

Sutcliffe, A., Maiden N. & Bright, B. (1997). An evaluation framework for representations in requirements engineering. *Proceedings of the International Workshop on Representations in Software Development.*, 16-32, Queen Mary and Westfield College, University of London.

Triesman, A. (1988). Features and objects (The fourteenth Bartlett memorial lecture) *Quarterly Journal of Experimental Psychology: Human Experimental Psychology, 40a.*