DEPARTMENT OF COMPUTER SCIENCE

Which properties make a modelling notation easy for untrained users to understand?

Carol Britton
Sara Jones

Technical Report No. 284

May 1997

Which properties make a modelling notation easy for untrained users to understand?

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

Abstract

One of the essential components of interactive system development is a model or representation that can be understood by all parties. Although in some development projects this requirement is fulfilled by an early prototype, there are many situations in which paper-based models have to be used. Clients and users are often unfamiliar with notations for modelling software systems and therefore find it difficult to understand models produced using such notations well enough to contribute effectively to the development process. In this paper we identify and discuss properties of modelling notations that contribute to the production of models that untrained users can readily understand.

1 Introduction

Although the relationship between notations, models and the quality of the system development process is not fully understood, the impact of the choice of notation on successful performance of many system development activities has long been recognised (Green 89 and 91a, McCluskey et al 95, Modugno et al 94). In the case of interactive systems, there is general agreement that an essential component of successful development is a model or representation that can be understood by all parties, including clients and users who may be untrained in the use of notations for modelling software. This is because successful development of interactive systems demands active user participation, and this can only be achieved through a shared understanding of representations of the system.

In many interactive system development projects, the need for user understanding is addressed by building up a skeleton prototype to offer options which can then be refined in line with the user's requirements. However, in certain cases, the prototyping approach may be neither feasible nor adequate. For example, early design ideas may need to be discussed with clients and users before even a first prototype is built. For safety critical systems, development based on a prototype would be unlikely to gain the client's confidence, and for large complex systems of any kind the development of complete system prototypes is unlikely to be cost-effective.

Where prototypes cannot be used, the main alternative is to use paper-based models constructed from notations designed for modelling software. The problem for the developer is how to select a notation which is easy for users to understand. The problem of which modelling notation to choose is exacerbated for the developer of interactive systems by the fact that many notations designed to model this type of system are relatively immature, their theoretical underpinnings are often weakly documented and some are still undergoing evolution stimulated by practical experience. In this situation it is difficult to assess whether a particular notation will encourage models that are readily understandable by untrained users, yet this is an essential property of a modelling notation for interactive systems.

With notations that are not easy to understand, novice clients and users are forced to put effort into deciphering the notations, rather than concentrating on the content of the model, a state of affairs described most eloquently in (Green 89): "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."

2 Criteria for choosing effective notations

The topic of criteria for effective modelling notations, has been addressed by authors from both the academic and industrial communities. Farbey (93) covers criteria relating directly to notations, such as readability, modifiability and lack of ambiguity, and criteria relating to the notation in use, such as the production of a well-presented specification, the cost in time to produce the model, and the amount of support available. Green (89) suggests that a notation should be able to support what he terms 'opportunistic planning', where high-and low-level decisions may be mingled, work may frequently be re-evaluated and modified and commitment to different decisions may be strong or weak. Although Green is writing about notations for programming, his point is equally relevant to the study of notations which are used earlier in the development process. In an article on hypermedia design, Garzotto (95) evaluates notations in terms of what can be described: a useful notation should be able to model information content and presentation, system structure, and interaction with the user. Davis (93) suggests a list of criteria pertaining to the effectiveness of models and the choice of notations. Davis' list includes criteria relating directly to modelling notations, such as that the notation should permit annotation and traceability, facilitate modification, and some that are expressed in terms of the software requirements specification. These include the criterion that "proper use of the technique should result in an SRS that is understandable by customers and users who are not computer scientists".

Among publications from authors in industry, criteria for modelling notations in the STARTS guide (87) 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 notation 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 notation supports consistency checking of the requirements themselves. Suitability for end-user agreement refers to ease of understanding of the notation by an untrained user, and assistance with structuring the requirements assesses the extent to which the notation supports hierarchical decomposition and separation of concerns in the model. The STARTS guide also regards the range of requirements covered by the notation as important, including functional, performance, interface, system development and process requirements. Admiral Training's (95) guide to interactive multimedia development is similar to the STARTS guide in placing emphasis on ease of understanding and on what the notation should be able to model. Effective notations, according to Admiral, should be able to describe the current situation, the target audience, the actual and required level of user performance, the overall aim of the system, the environment, possible constraints and details of specific functions.

Authors who have explored the topic of criteria for modelling notations differ in their approach and in the weightings that they give to different properties; in general, and as might be expected, publications from industry focus on practical criteria aimed at producing an effective model, whereas criteria chosen by academics tend to be more theoretically based. There are, however, certain criteria that are generally agreed by most authors to be important for effective modelling notations; one of these is the criterion that a notation should be as easy as possible for untrained users to understand.

3 Ease of understanding

In order to build a model that is to be the basis for discussion and communication, a notation must be used that is readily understandable by all parties concerned. This is particularly relevant in the development of interactive systems, where involvement of all user groups is essential and where it is unlikely that all clients and users will be familiar with software modelling notations. It is difficult to generalise about what makes one notation easier for an untrained user to understand than another. Inevitably, users who are shown a model will have personal preferences, different backgrounds and experiences and different attitudes to the model and to the unfamiliar notation. One user may be instinctively attracted to diagrams, while another may feel more at home with a text-based notation. Without detailed knowledge of a particular user, it is not possible to predict whether he or she will find a specific notation easy to understand. However, in trying to evaluate notations in terms of ease of understanding, it is useful to identify properties of notations that make models built using them readily understandable by untrained users. Building on the work of authors such as Green (80, 89, 91a and b, 96), Petre (95) and Sampson (85) we have identified the following properties that contribute to ease of understanding of a notation:

- The number of different symbols in the notation
- The discriminability of the symbols
- The degree of motivation of the symbols
- The extent to which the notation is perceptual or symbolic
- The potential for 'redundant recoding' in the notation
- The amount or structure inherent in the notation

Each of these properties is discussed below.

3.1 The number of different symbols in the notation

Green (80) 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 modelling notations, we can see that a notation such as storyboard with 2 symbols, is obviously going to be easier for novices to understand than notations such as the Z specification language, which has 77 symbols (Myers et al 96). It is not surprising that storyboards are widely used in the development of interactive multimedia systems, where not only the users, but also members of the development team, are likely to be unfamiliar with software modelling notations (Britton et al 96).

The number of symbols in a notation is a fairly crude measure, consideration should also be given to the amount of information carried by each symbol; for example the function arrows in Z carry considerably more information than the arrows in a data flow diagram. Z and other mathematical notations are difficult for untrained users to understand not only because they have large numbers of different symbols, but also because many of the symbols represent complex operations and concepts. Developers of systems also need to be aware of the trade-offs between a small and large number of different symbols in a notation. A small number of symbols will be easier for novices to grasp initially, but the restricted vocabulary of the notation may hamper their understanding in the long term.

3.2 The discriminability of the symbols

Discriminability refers to the ease with which different symbols in a notation can be distinguished from each other; this depends largely on how physically distinct each symbol is from others in the notation. Discriminability has been identified by both Green (80) and Sampson (85) as an indicator of how easy an unknown notation will be for novices to understand. As an example, we can consider the Z symbols shown below:

$\nrightarrow$	$\mapsto$
partial function	maplet

To someone who is not familiar with the Z notation 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 novices trying to understand a specification written in Z. Discriminability of symbols is a problem with many mathematical languages. In contrast, diagrammatic notations, 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 notations 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 1 below).

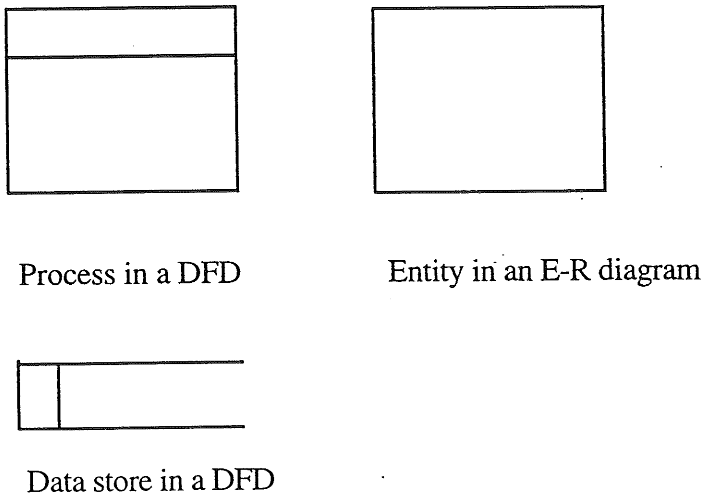


Figure 1: Symbols for process, data store and entity.

Different variations of standard notations, such as Yourdon and SSADM for data flow diagrams, may also bewilder novice users.

The level of discriminability in a specific model can be increased by use of techniques such as different sizes, fonts and use of colour, although these are typographic devices, rather than an intrinsic part of the notation itself. Figure 2, below, illustrates how even simple shading can help to distinguish data stores from processes in a data flow diagram; use of colour for shading or outline would be even more effective (Tufte 83).

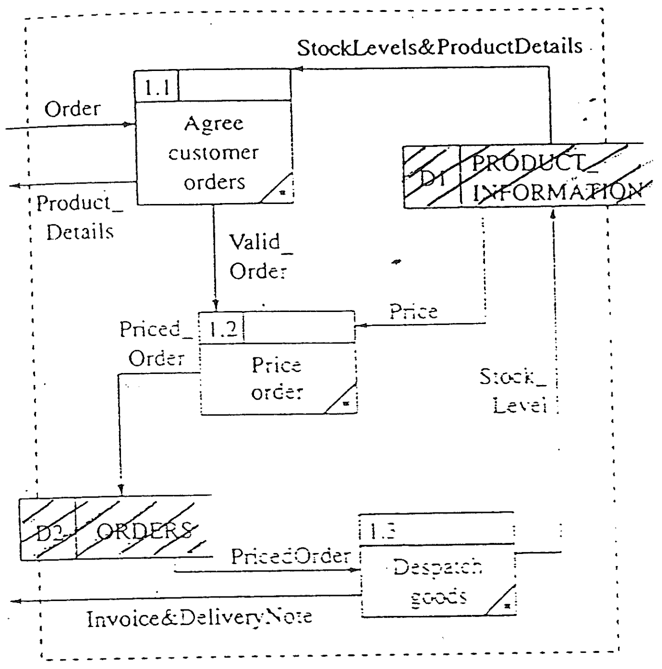
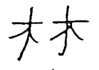


Figure 2: Use of shading to distinguish symbols in a data flow diagram

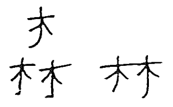
3.3 The degree of motivation of the symbols

The concept of motivated and arbitrary notations has been described by Sampson (85). A notation may be considered to be motivated if there exists a natural relationship between the elements of the notation and objects or ideas that they represent. Many of the characters in Chinese script are motivated, for example:



meaning tree

and



meaning forest

Not only does the form of the symbols suggest trees, but the symbols also preserve the idea that a forest is made up of many trees.

In an arbitrary notation there is no natural relationship between the object and the representation; the symbol +, for example, bears no obvious relation to the notion of conjunction.

Disciplines where the artefact being modelled exists in physical form , such as a bridge or a building in civil engineering, are more likely to have notations with motivated symbols, such as the drawings of an architect or structural engineer. The problem in software systems development is that the effects and interface of the artefact are perceived rather than the thing itself; this makes it difficult to provide motivated symbols for use in models: a shape can represent a room and a curve a bridge, but what realistically represents a process or an interaction?

Modelling notations for software systems tend to be arbitrary, although some diagrammatic notations do include motivated symbols, such as a line representing a link in entity-relationship diagrams, and an arrow signifying direction in data flow diagrams. Inclusive (such as Venn) diagrams and connected (such as state transition) diagrams may each be considered as motivated in certain situations. For example, a diagrammatic-inclusive notation can represent the positions of different elements on a screen, and a diagrammatic-connected notation may be effective for specifying possible routes through a system.

3.4 The extent to which the notation is perceptual or symbolic

Perceptual representations are those in which we perceive meaning directly without having to reason about them, for example the use of colour in electricity cables, or diagrams in various contexts, such as road signs. Most modelling notations contain both perceptual and symbolic elements. Graphs are perceptual in that they are diagrammatic, although they often contain labels (symbols) for the various nodes. Notations which are mainly symbolic rather then perceptual are frequently laid out to show certain aspects perceptually: Z's use of schemas, formatting of pseudocode and introduction of 'white space' to aid comprehension are examples of this. Text-based notations, which are symbolic, generally use perceptual devices to aid comprehension; these may include section headings, paragraphs, variations in size and font, bullets, emphasis and white space.

Perceptual and symbolic notations each have advantages and disadvantages in modelling. Models in symbolic notations, such as mathematical specification languages, are able to hold much more information than perceptual models, but are frequently beyond the understanding of untrained users. Models in diagrammatic notation often embody perceptual concepts, such as connectedness (as used in state transition diagrams) and inclusiveness (as used in Venn diagrams), which most people understand. However, without elaborate 'extras' the amount of information they can contain is restricted. A further problem with perceptual notations is identified by Green et al. (91b). Perceptual cues can be provided in a diagram by placing functionally-related items close together in order to emphasise the relationship; however, this can lead to diagrams which are cluttered and difficult to understand. There is also the risk with untrained users that any adjacent items in a diagram will be seen as related, even when this is not the case.

3.5 The potential for 'redundant recoding' in the notation

A concept that is closely related to that of perceptual properties of notations is what has been called 'redundant recoding' (Green 91a and Modugno et al 94). This involves presenting the same information in more than one way in the same model. Examples of redundant recoding in text-based models include headings, font size, and use of upper and lower case. In notations such as structured English and decision trees redundant recoding is provided by indentation and layout. Figure 3 below shows two versions of the same process description using structured English and a decision tree. The decisions to be taken are conveyed by both the content and the layout of the models.

```
if not local Customer (*lives outside 20 miles radius*)
        charge P&P
else (*local customer*)
        if TotalCost (* of Order*) < £30 then
              set DeliveryCharge
        else if TotalCost (* of Order*) > =£30 then
              if Distance < = 5 miles then
                    no DeliveryCharge
              else (*Distance > 5 miles*) then
                    if CustomerTradeRecord > = 1 year then
                          no DeliveryCharge
                    else (*CustomerTradeRecord < 1 year*)
                          set DeliveryCharge
```

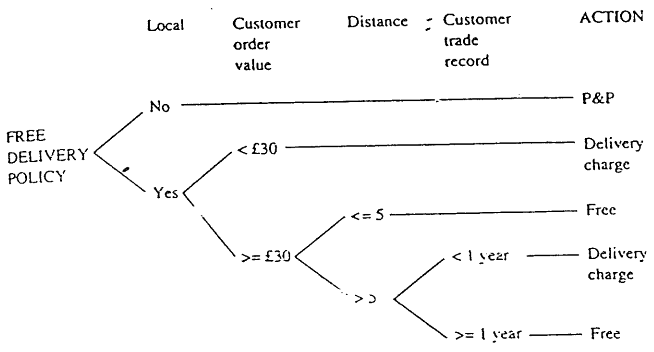


Figure 3: Process description using structured English and a decision tree

3.6 The amount or structure inherent in the notation

Sengler (83) has identified finding, decomposing and abstracting information as three of the most important activities in reading a model. Finding information involves searching the model; this process will be easier for untrained users to carry out if the notation used encourages a clear structure in the model. Decomposition and abstraction are important because a reader can only cope with a small amount of information at a given time. An effective notation should encourage decomposition of the model into sections or ‘chunks’, each of which is intellectually manageable by the reader. If the structure of decomposition is not clear, then the reader will be forced to devise his or her own decomposition of the model, which may well be different from that which the developer had in mind.

Among software modelling notations, the context diagram and imposed hierarchical decomposition of data flow diagrams mean that it is relatively straightforward for untrained users to find information at different levels and to concentrate on only a small part of the model at any given time. Mathematical notations, such as Z, often provide structure at low levels - for example, separation of the specification into schemas can help to identify normal and error cases of an operation - but no mechanism is provided to give an overview of the system as a whole. This adds to the difficulties of novice users of Z who wish to get a feel for the system as a whole before examining the details of the model.

Although diagrammatic notations are often thought to encourage more explicitly structured models then text-based notations, this is not always the case. In an experiment comparing textual and visual programming languages, Green and Blackwell (96) found that readers' understanding of programs depended more on the structure of the information in the program than on whether the program was written in a visual or text-based language.

4 Conclusion

In order to make an informed choice of modelling notation, a developer needs first to identify the criteria for effective notations which are relevant in the particular situation. The second stage of the developer's choice consists of assessing available notations in the light of the criteria which are considered important. In the development of interactive systems common understanding of models of the system by all parties is paramount. The aim of this paper has been to identify properties of notations that help to make models produced using them easier to understand for untrained users.

We have already said in this paper that whether or not users find particular notations readily understandable depends partly on their distinctive backgrounds, preferences and experiences. However, for developers faced with a raft of modelling notations to choose from, many of which have been largely untried on 'real' problems, it is helpful to have some idea of whether a particular notation is likely to produce a model which untrained users can readily understand. Each of the properties identified above is, on its own, a fairly crude indicator of the overall ease of understanding of a notation, but taken together they do give some idea of how quickly and well a novice reader will be able to understand a model developed in a particular notation.

References

Admiral Training Ltd (1995) *Multimedia Design - A Newcomer's Guide* Department of Employment, Sheffield.

Britton, C., Jones,S., Myers,M. and Sharif,M.(1997) "A Survey of Current Practice in Multimedia System Development", to appear in Information and Software Technology

Davis, A.M. (1993) *Software Requirements. Objects, Functions and States* Prentice Hall International.

Department for Trade and Industry and National Computing Centre (1987) *The STARTS Guide,* second edition, volume 1, NCC Publications

Farbey, B. (1993) "Software quality metrics: considerations about requirements and requirement specifications" in *Software Engineering: A European Perspective* R. Thayer and A. McGetterick (Eds.), IEEE C.S.Press

Fitter, M.J. and Green,T.R.G. (1981) "When do diagrams make good computer languages?" in Alty,J. and Coombs,M. (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 Vol 38, No.8

Green,T.R.G. (1980) "Programming as a Cognitive Activity " in Smith, H.T. and Green, T.R.G. (Eds.) *Human Interaction with Computers* Academic Press.

Green, T.R.G. (1989) "Cognitive Dimensions of Notations" in *People and Computers (HCI 89)* Sutcliffe and McCauley (Eds.) CUP.

Green, T.R.G. (1991a) “Describing Information Artifacts with Cognitive Dimensions and Structure Maps” in *People and Computers VI, Proceedings of the HCI'91 Conference*, Diaper, D. and Hammond, N. (Eds.), August 1991

Green,T.R.G., Petre,M. and Bellamy,R. (1991b) "Comprehensibility of visual and textual programs: A Test of Superlativism against the "Match Mismatch" Conjecture" in Koenemann- Belliveau,J., Moher,T. and Robertson,S. (Eds.) *Empirical Studies of Programmers* Fourth Workshop, Norwood NJ, Ablex pp121-146

Green, T.R.G. and Blackwell, A.F. (1996) "Thinking about Visual Programs " in *Thinking with Diagrams* IEE Colloquium Digest No: 96/010

McCluskey, T.L., Porteous, J.M., Naik, Y, Taylor, C.N. and Jones, S. (1995) *A Requirements Capture Method and its use in an Air traffic Control Application.* Software practice and Experience, Vol 25 (1)

Modugno, F., Green T.R.G. and Myers B.A. (1994) "Visual programming in a Visual Domain: A Case Study of Cognitive Dimensions" in *People and Computers IX, Proceedings of HCI'94*, Glasgow, August 1994

Myers, M., Britton,C., Jones,S.and Sharif,M. (1996) "An investigation into the measurement of modelling notations" Technical Report No. 257, Faculty of Information Sciences, University of Hertfordshire, to be presented at The Eighth European Software Control and Metrics Conference, Berlin, May 1997

Petre, M.(1995) "Why Looking Isn't Always Seeing, Readership Skills and Graphical Programming" Communications of the ACM, June 1995, Vol 38, No.6

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

Sengler, H.E.(1983) "A Model of Program Undersanding" in Green, T.R.G., Payne, S.J. and van der Veer, G.C.*The Psychology of Computer Use* Academic Press, 1983

Tufte, Edward (1983) *The Visual Display of Quantitative Information* Graphics Press, Cheshire, Connecticut.