The Odd Couple
or
Formal Notations and the Development of Information Systems

Technical Report No. 100

Carol Britton
Karry Omer

February 1990

Submitted to
The Journal of Information Technology
Volume 5, No 1

ABSTRACT

The development of information systems and the introduction of Formal Notations have tended to be taught as separate subjects. In this paper we hope to show that it is both possible and desirable to place Formal Notations in the context of systems analysis and design. By adopting an essentially pragmatic approach we feel that we have, to some extent, bridged the gap between the world of theoretical mathematics and the practical task of building information systems.

THE ODD COUPLE - FORMAL NOTATIONS AND THE DEVELOPMENT OF INFORMATION SYSTEMS.

WHY FORMAL NOTATIONS?

This paper describes our experience of introducing the ideas that are embodied in what is broadly called Formal Notations into a course entitled 'The Development of Information Systems', consisting of 2 hours per week contact time. This course was taught to two separate groups: first as an option to a group of 12 students who were in the second year of a three year part-time MSc in Computer Science, and secondly to the final year of the combined honours BSc. None of the students had any prior knowledge or experience of using Formal Notations, although many realised that they were current 'buzz words', and felt that it would be useful to know more about their meaning. All students indicated a willingness to learn, appreciating that the ability to understand Formal Notations would be a useful addition to their list of skills.

At this point we wish to note that we use the term 'Formal Notations' in preference to the more commonly used 'Formal Methods', as we feel that the current state of the art in this field does not constitute a method as such. Whilst work is being done to remedy this situation, in academic establishments and commercial institutions alike, what exists at the moment is essentially a means of description that is mathematically rigorous and precise.

Both lecturers on the course were new to teaching, one of us having completed an MSc, and the other being halfway through a PhD, at Hatfield. We both had an interest in the use of Formal Notations, arising on the one hand from work with the formal notation OBJ to specify abstract data types, and on the other hand from a research interest in the use of mathematics to capture system requirements. We were keen to introduce a little of what we had learned into our teaching, since we felt that precision and rigour were often missing from existing established methods of system development, such as SSADM (CCTA, 1986).

In addition, we were aware of recent changes in software development, such as the Ministry of Defence's introduction of its standard 00-55, which was announced in April 1989 and pertains to the development of all systems deemed by the MOD to be safety critical. The standard requires that the functional specification for a system is written down using mathematics. Such specifications have normally been written using natural language, which is typically vague and open to interpretation. (Meyer, 1985). We felt that, in the light of such developments as 00-55, we had a responsibility to our students to give them at least some idea of what Formal Notations looked like, and to see how they could be used advantageously in the initial stages of the development of a system.

Our intention was that we would show how a formal language can be used to capture and specify requirements. We wanted to concentrate on the very early stages of information system development, understanding that this stage of system development poses for the developer perhaps the hardest problem of all, and that errors made here result in software that is late, over budget and that does not meet the customer's needs (Boehm, 1983). We did not intend to look at topics such as program design or formal proofs in respect of formal notations, since these topics were already covered in other courses at Hatfield.

We also felt that mathematics had a poor image in the field of information systems development; many regarded its use as 'too difficult' and of little relevance to the design and development of systems. We wanted to show our students that, whilst they might not become mathematicians in the strict sense of the word, they could, with a sincere desire to improve the quality of the software they developed, usefully employ some degree of mathematical formality. We hoped that introducing the topic of Formal Notations in the context of system development,

rather than as a completely separate subject, would make the subject more accessible to the students.

WHAT WAS THE CONTENT OF THE COURSE?

The first part of the course concentrated on teaching what we called 'traditional' systems development. We did this for three reasons. First, because structured methods of analysis and design are widely used in industry, with some measure of success. Being pragmatists we wished to provide our students with marketable skills. Second we were aware of shortcomings in certain techniques used in some of the proprietary methods, notably the variety of interpretations which may be given to a single model. We hoped that by taking a critical approach to the teaching of such techniques our students could use them more wisely. Finally, we intended that using those methods that are most commonly used as a yardstick would offer a contrast and useful basis for comparison later in the course when we introduced alternative approaches.

The lectures were based on a life-cycle model of development and introduced structured techniques such as data-flow diagrams and entity-attribute relationship models, which are found in many systems development methods. The course was taught using a case-study approach, as developed by Juliet Brown and Clare Tagg at Hatfield, (Brown and Tagg 1989). Throughout the year the students, working either in groups or alone, analysed and designed a system to computerise the delivery side of a small dairy, using a cut-down version of SSADM. We concentrated on the main modelling techniques offered by this method: data flow diagrams, entity models and entity life histories. The lecturers acted as both clients and advisors, guiding the students along their way, using the contact time for lectures and tutorials. Little of the emphasis was placed on implementation, although all of the students managed to get at least a part of their system up and running.

During the second semester we began to look critically at the traditional way of building systems, discussing the software crisis, its causes and possible solutions to it. Through their own experiences with the case study the students identified problems and basic flaws in the approach they had been using, and established that there was indeed a need to look for complementary techniques. Throughout our teaching we tried to stress that different approaches, such as Prototyping and Formal Notations, should not be regarded as an alternative to the traditional approach, but rather as additional tools for solving problems. We tried to get the students to see that they should build up a 'toolbox' of techniques and to use any one as it seemed appropriate. (Benyon and Skidmore, 1987.) We suggested, for example, that Prototyping might well be especially useful in the design of the user-interface, or that using a Formal Notation could be useful to capture or specify a particularly complex part of the problem requirements. An example of the latter from the Dairy system which we discussed was the three types of order that existed: Standing, Exception and Promotion Orders.

Our own experience of using Formal Notations was that they were useful in the analysis stage as a 'doodling tool' to help the analyst to understand just what it was in the problem domain that was to be modelled and encompassed in the computerised solution to the problem. The use of Formal Notations requires a mental discipline that, once learned, enables the analyst to ask a different kind of question from that which is usually asked during structured analysis. The act of forcing oneself to be precise also raises what we called 'what if?' questions during analysis. An example of this that we used from the Dairy was the relationship between a customer and the three different types of order that exist in the system.

The EAR data model of this relationship may look like this:-

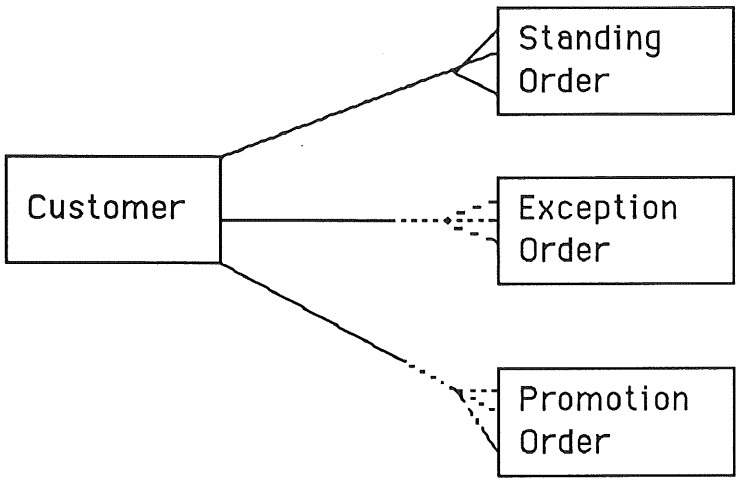

Figure 1: EAR Model of Customer - Orders Relationship

There is a basic underlying assumption in this diagram that a customer only exists if he or she has a regular Standing Order. When the students modelled this part of the problem using only data flow diagrams and EAR models, they all failed to realise the implications of their diagram with respect to a customer who goes on holiday for a fortnight or abroad on business for six months. Our own model was equally imprecise, until we came to specify this part of the system formally. When defining the realationship between Customer and the various types of Order ourselves, using the algebraic specification language OBJ, we felt that we were led to ask, "What happens if the customer goes away on holiday or business for some time and cancels their Standing Order?" This supported our instinctive feeling that one of the main advantages of using Formal Notations is not so much the answers they provide, but the questions they force the developer to ask. We believe that the use of Formal Notations encourages a mental discipline that becomes alomost subconscious after a relatively short period of time. This discipline forms in the developer the habit of asking questions which are of a different nature and quality in the analysis phase of development from those which are asked using structured analysis techniques. It is as though one's view of the world changes, with the result that the analysis has a clarity and precision that is often lacking when using more traditional techniques.

We saw our approach on the course as essentially pragmatic, attempting to bridge the gap between the 'us and them' of the theoretical mathematicians on the one hand, and people who have a real system to build, with limited time and resources with which to do the job, on the other.

We found that the sessions in class became increasingly animated, with the students being keen to express their own opinions, based on their experiences. Their input became more and more sophisticated, as they considered alternative ways of viewing the development process. This was rewarding for both students and lecturers alike.

HOW WERE FORMAL NOTATIONS INTRODUCED?

We began to introduce the idea of using formality somewhat generally at first, mentioning the sound mathematical principles underlying it such as Z-F Set Theory and First Order Predicate

Calculus (There are a number of textbooks available on discrete mathematics for Software Engineers, for example see Denvir,1986; or Woodcock and Loomes,1988). We went on to outline some of the advantages that are claimed for Formal Notations, such as the assertion that using a formal language forces the developer to raise more searching questions and thus leads to a better understanding of the problem. Other strengths of the use of Formal Notations that we suggested were that the specification can be proved to be internally consistent, and that reliable predictions can be made about a formal model where they cannot about one which is not formal in the mathematical sense.

We then considered any objections to these arguments, many of them coming from the students themselves, such as the concern about the use of formal specifications as a means of communication between the analyst and the customer. Another objection that was raised was the feeling that a formal specification would be inappropriate for many applications, for example predominantly people-oriented, or 'soft systems' (Checkland,1981), or those which were to be implemented in Fourth Generation Languages.

These introductory sessions on the subject were inevitably fairly conventional lectures, but we encouraged the students at all times to participate and challenge what we were saying. Several of them made it abundantly clear at this stage that they thought that the use of Formal Notations was a waste of time and had little or nothing to offer in the development of systems. (It is interesting to note that many of these students had changed their opinions by the end of the year.)

The students were then at a point where they could see how Formal Notations might be useful, and were keen to get to grips with a formal language. We looked briefly at three languages: OBJ (Burstall and Goguen 1977), Z (Abrial 1980) and CCS (Milner 1980). In each case we used small examples which could be easily understood by the students, since we did not want problem complexity to impede the students' understanding of the way the languages work. For each language we gave two introductory lectures, two hours of practicals or exercises and a final discussion session where the students were encouraged to consider how these small examples might or might not scale up to cope with large, real-life problems. The students also considered whether each language could have been useful when they were working on the Dairy case study, or any other systems in which they had been involved.

An example from the Dairy case study which we used was the idea of looking at what happens to the customer file if the user tries to enter the same customer details twice. Using data flow diagrams the students had already analysed and modelled processes such as 'Handle Customer Order'.

When exploded at a lower level, this gave diagrams such as:-

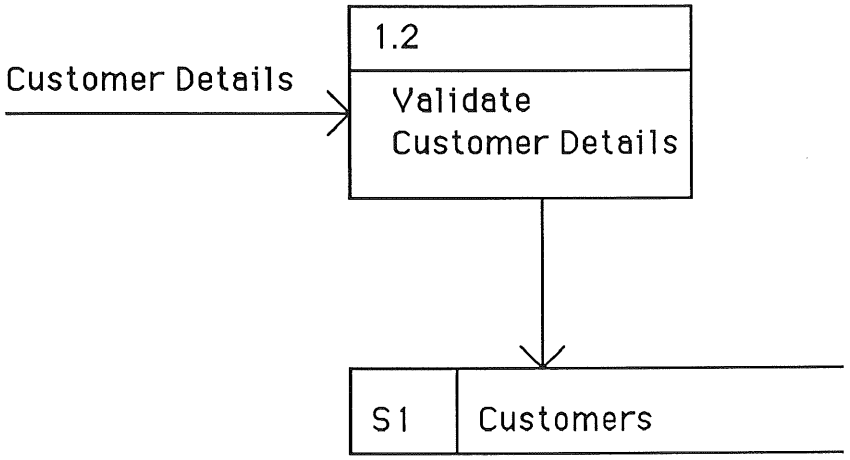

Figure 2: Data Flow Diagram of Process 'Validate Customer'

A Process Specification for validate customer might have included such pseudocode as:-

If CustID is found in Customer File
Then error
Else add CustDetails to CustomerFile

The executable formal specification language OBJ provided an interesting contrast to the structured approach. Here, we modelled Customer as an Abstract Data Type (Guttag and Horning,1978) and specified it in terms of legal operations that could be performed upon it in the system.

Next, a collection, or file of Customers, was specified. This included such operations as adding a Customer to a file, verifying that a Customer was already in the file and so on. These operations are essentially the set operations which we find in set mathematics.

Part of the OBJ specification of CustomerSet is found below:-
Comments are enclosed in * *.

OBJ

CUSTOMERSET / CUSTOMER *Customer set has access to all the operations on Customer*

SORTS

CustomerSet *New type or Sort declared*

OPS

nullCustomerSet :-> CustomerSet	*the empty set*
custSetPlusCustomer : CustomerSet Customer -> CustomerSet	*2 operations to
custAdded : CustomerSet Customer -> CustomerSet	add a customer*
custSetMember : CustomerSet Customer -> BOOL	*the member of set operation*

VARS

cSet : CustomerSet	*variables for the customer set and 2
cust, cust2 : Customer	customers*

EQNS

(custSetMember(nullCustomerSet,cust) = F)

(custSetMember(custSetPlusCustomer(cSet,cust),cust2) = T

IF(cust) == (cust2))

(custSetMember(custSetPlusCustomer(cSet,cust),cust2) = custSetMember(cSet,cust2)

IF not((cust) == (cust2))

The above equations define the meaning of the operation CustSetmember

(custAdded(cSet,cust) = custSetPlusCustomer(cSet,cust)

IF not(custSetMember(cSet,cust)))

This equation ensures that duplicate customers cannot be added to the set

JBO

As a final example, we specified the same 'add' operation in the specification language Z using the following schema:-

File Add

CustFile, CustFile' : Key Record

CustID? : Key Record

---

dom (CustID) dom (CustFile) =

CustFile' = CustFile CustID?

Glossary.

=	partial function
=	set intersection
=	the empty set
=	conjunction
=	set union

The top half (Declaration) of this schema says that both the old and the new files (CustFile and CustFile') are collections of mappings from keys to records (containing customer details). The input (denoted in Z by '?') is also a collection of key-record pairs. The bottom half of the schema (Predicate) expresses the pre-condition that the new pairs to be added must not already be in the old file. It then states that the new file is equal to the old file with the new pairs added.

This example of adding customers to a file was not appropriate to demonstate the language CCS, which deals specifically with concurrency. We were able to point out, however, that knowledge of one Formal Notation only may not be sufficient. If the Dairy case study had involved some processing that required communication to take place, then at least two Formal Notations would have been necessary, CCS and one other.

We did not suggest that the two versions using Formal Notations provided information that was intrisically different from the data flow diagram and the process specification, rather that they said it more concisely and precisely, leaving less room for error. The 'Customer-add' example is clearly a simple one, but it can be seen that with more complex problems the difference in the level of precision between the formal and structured specifications becomes increasingly relevant.

We did not have time to go into any one language in depth, indeed this was not our aim. We wanted to introduce the concept of Formal Notations in some sort of context, to whet the students' appetites for more information, and to encourage confidence in their ability to use some degree of mathematical formality and rigour in their work.

DID WE SUCCEED IN OUR AIM?

In addition to the coursework, which was presented as a folder at the end of the year, the students were assessed by way of a three-hour written exam which consisted of essay-style questions. (Some of these are included in the appendix.) A large proportion of both groups chose to answer the questions we had set on the role and use of Formal Notations, and these questions were answered extremely well, indicating a sound grasp of the topic. Three students from the MSc. group have chosen to develop their knowledge of the subject by way of their third year project. All of the students on both courses said that they were appreciative of the fact that the topic had been taught, although some had had initial reservations. They felt that they had gained confidence in the area, some of them overcoming a previous 'fear' of mathematics, and also that they had added to the size of their 'toolbox'.

A criticism that was levelled against us was that the examples we used seemed trivial. The reason for choosing such small examples was that we were concentrating upon the principles embodied in Formal Notations, which were new to the students and rather complicated to grasp at first. We did not wish to complicate things further by introducing complex aspects of a problem to illustrate the use of Formal Notations. With hindsight we feel that it may have been better to have taken a more complicated part of the Dairy case study, such as Billing, and used Formal Notations to capture its requirements.

Overall, however we feel that the course was successful. Our students can now understand the increasing number of references to Formal Notations in the literature on Information Systems. They know that Z is more than the last letter of the alphabet, that OBJ isn't yet another buzz word connected with things 'object-oriented' and that CCS is not a gas that destroys the ozone layer! Some have found a genuine interest in the subject ,and will want to develop this to further their careers. It is certain that the skills and knowledge that they have acquired will be of interest to employers.

We hope that all our students have seen that there is an alternative way of analysing and designing information systems, and that even where they are using 'traditional' methods, such as SSADM and JSD, they will approach these from a more rigorous standpoint.

APPENDIX

Examples of questions in the examination papers for the Systems Development courses.

1. Many managers resist using formal methods because they believe that the solution to their software problems lies in the proper use of informal structured methodologies.

   a) To what extent would you agree with the managers' viewpoint?

   b) Do you think that knowledge of at least one formal specification language would be of use to a system developer? Give reasons for your answer.

2. 'Traditional methodologies are largely "finger in the dyke" solutions to halt the software crisis, but are unlikely to solve it since they address the symptoms, not the problems'. (Martin Loomes)
   a) To what extent would you agree with the above statement?

   b) Discuss alternative or complementary ways of addressing the problems of the software crisis.

3. The most important thing about the use of Formal Notations is the questions they make the designer ask about the problem.

   Discuss the above point of view. Illustrate your answer with examples from a case study with which you are familiar.

REFERENCES

Abrial, J.R. (1980), The Specification Language Z: Syntax and Semantics, Oxford University Computing Laboratory Programming Research Group.

Benyon, D. and Skidmore, S. (1987) Towards a Tool Kit for the Systems Analysts, The Computer Journal, Vol. 30, No. 1, 1987.

Boehm, B. (1983). Software Engineering Economics, Engelwood Cliffs, NJ, Prentice Hall.

Brown, J. and Tagg, C.(1989), A Case Study Approach to Teaching Systems Development, Presented at the 1988 ISTIP Conference, Sunningdale.

Burstall, R.M. and Goguen, J.A. (1977), Putting Theories Together to make Specifications, IJCAI '77, Invited Paper.

CCTA, (1986) SSADM Reference Manuals, Version 3, Issue 1, March 1986.

Checkland, P. (1981) Systems Thinking, Systems Practice. Wiley, Cichester, 1981.

Denvir,T. (1986) Introduction to Discrete Mathematics for Software Engineering. Macmillan 1986.

Guttag, J. and Horning, J. (1978) The Algebraic Specification of Abstract Data Types. Acta Informatica 10, 53-66, 1978.

Meyer, B. (1985) On Formalism in Specifications, IEEE Software January 1985.

Milner, R. (1980), A Calculus for Communicating Systems, Springer-Verlag LNCS vol 92.

Woodcock, J. and Loomes, M. (1988), Software Engineering Mathematics. Pitman 1988.

BIOGRAPHICAL NOTES.

Karry Omer has a BAHons degree in Social Administration from Nottingham University, and has worked as a social worker for a number of years, both in Nottingham and in London. After taking a break from paid employment to look after her son she obtained a conversion MSc in Computer Science from Hatfield Polytechnic. She is currently employed at Hatfield as a Research Assistant, looking at the applicability of Formal Notations for use during the early stages of the development of Information Systems.

Carol Britton's first degree is in French, which she taught for some years. After a ten year break from employment while her children were small, she decided to retrain by studying part-time for an MSc in Computer Science at Hatfield Poytechnic. She is currently a Senior Lecturer in the Division of Computer Science at Hatfield, with special interest in Formal Notations, the Development of Information Systems and Object-Oriented Design.

Address for correspondence:
Hatfield Polytechnic,
College Lane,
Hatfield,
Herts AL10 9AB.