The Roles of Formal Specification in the System Design Process

7S−2

Tim Gleeson and Toyofumi Takenaka

ATR Communication Systems Research Laboratories

1 Introduction

Incomplete and ambiguous problem requirements are inherent to system design. Two fundamental parts of the design process are *understanding* and *elaborating* these requirements [Gui90]. These activities are carried out continuously during the design process, but two of the more important mechanisms used are simulations within the problem domain and construction of solutions. These are both effective because they generate questions about the problem which helps the designer to discover the incompleteness, ambiguity, and perhaps contradictions within the problem requirements.

However, there is another well known, but infrequently practiced, method to structure problems: we can write a formal specification of the problem.

This paper shows how formal specifications can be a useful tool in the psychological investigation of the design process and how existing psychological results can illustrate the roles of formal specification and how they can be better integrated and supported. We will illustrate this interaction by examining the activities of writing and using formal specifications. The use of formal methods to ensure correctness is not explicitly addressed.

2 Writing formal specifications

Writing a formal specification can be seen as an externalisation of the internal process of understanding. Understanding partly involves the construction of an internal model which reflects the external problem. It is difficult to build such an internal model because of certain cognitive characteristics:

- simple oversight of issues
- deliberate omission of issues due to poor prioritization
- use of (non-logical) induction in model construction
- cognitive overload leading to forgetting

The use of a constricted notation, e.g. a formal one, forces more complete elaboration of these issues and the externalisation of the method, on paper or with computer support tools, helps to overcome the last problem.

This forces the question: at what *level* do we exercise the problem in order to understand and elaborate it? The answer is: at the highest level possible. Formal specifications, at the highest level of abstraction, thus have an important role to play, but many automated exercises are still technology limited, e.g. specification execution, animation, theorem proving.

2.1 Selection of notation

The large range of formal (and informal) specification notations which exist clearly demonstrates that there is no single, ideal notation, nor will there ever be. The process of choosing a notation is characterised by the question: "in what terms do we specify this problem?" This decision is of fundamental importance because once the choice has been made, the problem is then analysed in those terms: using the Z [Spi89] notation, we look for sets, sequences and functions; when using CSP [Hoa85], we look for processes and communication.

Apart from the obvious advantages of rigour, the use of formal notations poses a problem in this way because they limit the informal, and far more flexible internal creative analysis processes. Nonetheless, the existence of formal notations at least allows us to psychologically investigate how they are chosen, something which would be very difficult to do with unstructured internal processes.

Furthermore, the investigation of how notations are selected against problem structures may reveal evidence about how previous experience is drawn upon in later stages of design [Gui90].

2.2 Elaborating requirements

When developing programs we should not only record the intermediate steps but also the decisions, justifications and rejected alternatives involved. Such decisions are an essential part of the design process. Recording them also increases the potential for the reuse of the constructed items and aids maintenance which requires this information for decision reassessment. Similarly, in the step between informal requirements and the top level formal specification, we should record the reasons for the decisions we have made which seem to be fundamentally different in character from those refinements made between later formal domains. For example:

- Making implicit requirements explicit
- Adding assumptions
- Resolving ambiguities
- Removing inconsistencies

All of these activities can be aided by the construction of a formal specification, particularly if the formal notation has a calculus of properties, as illustrated by the work of Hoare et al. [H+87]. It is the existence of such a calculus that not only allows us to ask questions about the problem (and get objective answers) which increases our understanding, but can also suggest those questions.

Refinement during solution production is generally from one formal domain (a higher-level specification or program fragment) to another (a lower-level specification or program fragment). However, writing a formal specification can be seen as an informal refinement from *several* domains, including:

- the problem statement
- problem domain knowledge (e.g. lifts)
- real world knowledge (e.g. flow of time)
- computing knowledge (e.g. pragmatics and computability theory)

3 Using formal specifications

We stated that two major goals in systems design were understanding and elaborating requirements. If we construct a system from a formal specification, then we only have the understanding left to do: the formal specification should completely elaborate the system requirements.

The designer still needs to understand the formal specification, even though it may be complete and unambiguous. The internal model that the designer constructs will itself be incomplete and ambiguous. Thus we need a psychological study of the

relationship between internal and external models and how they interact. This will be far more tractable when the external model is formal, complete and unambiguous rather than informal.

One way in which the internal and external models interact is by the process of testing, or comparison, where the internal model is exercised against the external one. When writing formal specifications, a calculus of properties within the specification was one of the advantages of formal specifications mentioned. In the case of using formal specifications, this can be done with both a calculus of properties and a calculus of refinement. The designer can use the internal model to suggest refinement steps, but will always be able to check these against the external model when the formal steps must be made. This is a self correcting system: the external, formal refinement step is necessarily correct, but it leads to the correction of internal model errors. Thus we get the best of both worlds: external formality and rigour and internal non-logical, creative reasoning which is corrected. We must build systems which promote this synergy.

4 The importance of formality

Why is *formality* important?

A formal specification notation should be a relatively simple thing, with a limited number of concepts, yet it should be able to express everything in its domain of applicability. This makes writing specifications easier: the pattern matching process between problem element and suitable specification element is simpler. This also makes refining specifications into solutions easier: e.g., take the subset of a set, decompose a function. This kind of calculus of refinement is well illustrated by Gries [Gri80].

Another benefit of formality is the constructive externalisation of information. Externalisation is important because it relieves many of the cognitive problems involved in design. Many non-formal methods also encourage externalisation, but formal methods are particularly constructive in that they have objective criteria for what must be externalised. In addition, of course, the externalisation allows for the use of the formal calculi we have already mentioned.

However, a limited domain notation, is not a sufficient condition of utility. If we look at a Pascal program we can regard it as a string of characters: this is a very simple notation but entirely unhelpful. We can look at it as a string of tokens, which is more helpful, or better still as an abstract parse tree. Though the more abstract notation of formal methods, such as sets and functions from the Z [Spi89] notation may capture the essence of what is required better than implementation structures, such as arrays and jumps, it is by no means clear that they are universally ideal.

5 Difficulties of formal specifications

One of the claimed benefits of formal specifications is that they force the specifier to be open and explicit, forcing a complete binding contract between client and implementer. This can also be problematic because they may force problem choices that the intended client doesn't yet want to make. The client may want to hide certain aspects in decent privacy. Formal methods should allow for well defined areas of doubt and uncertainty and still be usable.

This raises social questions of why we write programs: sometimes we write them to understand the world, rather than because we have a specific goal in mind. This may make it very difficult to write a formal specification before we construct a system, an article of formal methods dogma.

For example, we may implement a new programming language system because we want to investigate how useful from a human viewpoint that language is — irrespective of whether we have a formal semantics for it. This should come as little surprise to us: there is a world of difference between proving properties about a system (or designing them into a system) and discovering properties about it, whether the system is number theory, a programming language or a spreadsheet application.

Further, even if we knew what we wanted, not all properties can be stated formally. Formal methods are still very weak in many areas, particularly those concerning performance, real-time issues and concurrency.

Formal specifications, though perhaps mathematically and legally complete may be grossly overdetailed from certain psychological viewpoints. "Superabstractions" which omit much of the detail of a system may be useful. The purpose of superabstractions is not to be accurate or complete but to help the user construct and use mental models of the system. The psychological study of "advance organisers" should help us to better structure our specifications and to produce more useful superabstractions.

6 Conclusions

From the psychological viewpoint of the study of the design process, the writing of formal specifications provides a valuable tool because it enables us to largely separate the activities of problem elaboration from problem understanding. We can separately investigate how people generate formal specifications and how they use them to produce solutions. Important questions we should ask are what forms of knowledge or expertise are used in these two stages.

From the practical viewpoint of building design support tools we can use psychological results to help us to better integrate formal methods into the design process, for example, to aid in the selection of appropriate notation and to remove much of the symbolic tedium of formal systems.

Current design tools, such as gIBIS [CB88], allow designers to record their ideas and understanding of a problem, and how these are elaborated and altered as the design progresses. Such tools can also be used to help design solutions, but the problem elaboration issues may be mixed up with solution design issues. We should attempt to separate the two because they are very different activities and need different kinds of support. We need separate but integrated tools to aid us in the different roles we have identified for formal specifications: elaboration and understanding of requirements through writing formal specifications and using them in developing solutions. We need mechanised support for both property and refinement calculi, both of which can be used to improve the designer's mental model.

References

[CB88] Jeff Conklin and Michael L. Begeman. gIBIS: A hypertext tool for exploratory policy discussion. *Transactions on Office Information Systems*, 6(4):303–331, October 1988.

[Gri80] David Gries. *The Science of Programming.* Springer-Verlag, 1980.

[Gui90] Raymonde Guindon. Knowledge exploited by experts during software system design. *International Journal of Man Machine Studies*, 33(3):279–304, September 1990.

[H+87] C. A. R. Hoare et al. Laws of programming. *Communications of the ACM*, 30(8):672–686, 1987.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[Spi89] J. M. Spivey. *The Z Notation: A Reference Manual.* Prentice-Hall, 1989.