Chapter 8

Conclusions

This chapter reviews the objectives of the thesis, shows how the work presented in this thesis meets these objectives and makes suggestions for future work.

8.1 Review of Thesis Objectives

The main objective of the thesis is to show that combining object oriented and formal methods is a practical and effective way of improving the software development process. To meet this objective the thesis addresses five separate goals:

- To record an understanding of software devlopment and, using this understanding, to formulate an *ideal* software development environment.
- To show that a formal object oriented development method is a step towards achieving such an *ideal*.
- To remove the ambiguity and informal nature of object oriented terminology by developing an object oriented semantics.
- To construct a formal object oriented development (FOOD) framework based on these object oriented semantics.
- To illustrate the effectiveness of FOOD by applying it in the development of a non-trivial software system.

8.2 Meeting Objectives: The Contributions of the Thesis

Chapter one establishes the complementary nature of object oriented and formal methods within the domain of software engineering. The thesis is developed on the premise that *correctness* is the most important property of software, and argues that *software engineering* must be based on mathematical models. Object oriented methods are presented as providing a practical solution to the synthesis and analysis of mathematical models of computer systems, in general, and software in particular. LOTOS is proposed as a good language for implementing object oriented semantics at all stages of

software development. The ADT part of LOTOS is shown to be suitable for implementing abstract requirements models, whilst full LOTOS is put forward as an ideal language for incorporating these abstract requirements in a more concrete design model.

The integration of a process algebra and ADT, within LOTOS, is the main reason for its use within this thesis. It provides a smooth transition from requirements to design by facilitating the implementation of a class at various levels of abstraction within the same semantic framework: as an abstract data type in the requirements models to a process in the design models. This novel approach to using LOTOS is a major contribution of this thesis.

Chapter two introduces object oriented methods by first considering object oriented analysis and its relation to other analysis methods. It motivates the development of a formal approach to analysis and requirements capture. The chapter provides a rationale for the success of object oriented development methods and, based on this rationale, proposes a set of object oriented models which can be used throughout software development. A contribution of the thesis is a recognition of the importance of the way in which these models are co-ordinated. Further, the thesis identifies the importance of establishing a semantics for object oriented terminology which is consistent throughout the development process. As a system moves from the abstract to the concrete it is fundamental to successful development that leaps between different semantic frameworks are curtailed. The thesis proposes a mechanism for constructing design models in which the requirements are still present. In chapter 2, abstract data types are shown to provide a level of abstraction suitable for the specification of object oriented requirements. However, as types are more general than classes, we argue that it is necessary to develop a more abstract view of objects and classes.

Chapter three includes one of the major contributions of the thesis: an abstract object oriented semantics based on the modelling of state transitions. We argue that the state transition view is ideal for communicating object oriented requirements. An *object-labelled state transition system* (O-LSTS) semantics is developed, and the object oriented notions of classification, subclassing, composition, configuration and interaction are formally defined. The O-LSTS semantics are then used in the definition of an object oriented analysis language (OO ACT ONE), which, as its name suggests, is syntactically similar to ACT ONE (but with a distinct object oriented 'sugaring'). An important contribution of this thesis is the formulation of two different types of subclassing, namely extension and specialisation, and the provision of language mechanisms for defining class hierarchies based on these relationships. The thesis argues that these two mechanisms are sufficient, during analysis, for the definition of all subclassing relationships. In conclusion, chapter three identifies the need for the requirements models to be executed: *customer validation* is argued to be dependent on the ability to step through a dynamic execution of the requirements models. A translation to ACT ONE provides a means of stepping through the dynamic behaviour of an OO ACT ONE specification.

Chapter 4 considers how the formal object oriented models can be successfully used in the initial stages of software development. In particular, customer-analyst communication is identified as the most important aspect of requirements capture. This chapter examines how the analyst, using the formal object oriented models, can achieve a mutual understanding of the problem domain with the customer. The difficult question of how the analyst can and should alter the way in which a cus-

tomer conceptualises their needs is considered. The chapter concludes by re-iterating the importance of constructive specification in an object oriented development strategy. The thesis argues that the structure of the problem domain should be recorded in the requirements model: it improves understanding and acts as a framework upon which design can begin. Designers must be given the option of reproducing the problem domain structure in their designs. Without this option, object oriented design can be very difficult.

Chapter five considers the role of design within software engineering. The thesis shows that *constructive* design, based on the application of *correctness preserving transformations*, is the most practical solution to the problem of ensuring that design meets requirements. The way in which the ACT ONE executable model of requirements is incorporated in the full LOTOS designs is an original and effective means of going from analysis to design. We argue that a process algebra is a useful conceptual tool for the specification of communication models, which is fundamental to object oriented design. Chapter five shows that there are a potentially very large number ways of using LOTOS to model objects and classes. These provide different object oriented semantics with respect to the way in which models communicate and interact. The thesis argues that it is the role of designers to choose a model which is best suited to their target implementation environment. A number of different, though equally valid, object oriented communication models are put forward. A major contribution of the thesis is the formulation of a means of generating different designs from the same requirements, all of which maintain *correctness*.

Chapter five also contributes a small number of CPTs for the manipulation of composition structure within object oriented designs. The ability to restructure the composition of classes is shown to be fundamental in object oriented design: targetting design towards classes which have already been implemented is dependent on the ability of designers to restructure their designs whilst maintaining correctness. The transfer of structure in the ADT part of a LOTOS design to structure in the process algebra part is shown to be an important step in the movement from abstract to concrete. Chapter five emphasises the importance of designers understanding the facilities provided by their target implementation environment. The role of designers is defined as restructuring the requirements from being *customer oriented* to being *implementation oriented*. As with requirements capture, we emphasise that the design models provide only a framework for the development of a design method.

Chapter six considers the implementation of OO LOTOS designs. A major contribution of this thesis is the formulation of general strategies for implementing the formal object oriented designs. The importance of having a fundamental understanding of implementation language semantics is emphasised. The thesis shows, in some detail, how appropriately targetted OO LOTOS designs can be implemented in Eiffel. The thesis also illustrates how FOOD is well suited to the development of concurrent software.

Chapters two to six provide a framework of models and techniques for using these models in the development of software. Chapter seven argues that a software development method must evolve from the use of models rather than being an immediate consequence of their formulation. It also argues that practical use of models and methods is the only true means of evaluating their effectiveness. Consequently, as part of this thesis, chapter seven reports on a case study in which FOOD was used

to develop a non-trivial software system. This contributes to the thesis by placing the theory in a more practical domain.

As a whole, the thesis identifies the problems inherent in software development, shows how a formal object oriented method can overcome many of these problems and provides a framework upon which such a method can evolve. In short, FOOD shows that, although much more work needs to be done, it has the potential to result in an efficient means of producing software which fulfils its requirements.

8.3 Future Work

FOOD is a framework of models and methods which provides the basis upon which an *ideal* software development environment can be constructed. To get closer to this *ideal* much more work needs to be done:

- **Analysis and requirements capture**
  A natural step forward is to provide a direct implementation of OO ACT ONE specifications, rather than translating them to ACT ONE. Such an implementation could be incorporated in a comprehensive set of analysis and synthesis tools. As the customer is central to analysis and requirements capture, it is important to further investigate the process of customer-analyst interaction within a formal object oriented framework. The diagramatic representations of the object oriented requirements must be made an integral part of the development environment. It is important that a means of letting the customer directly interact with the requirements models is developed. This interaction can be used in the construction and validation of requirements models.
- **Design**
  In the thesis, LOTOS is used to provide our object oriented design semantics. This is fine in a prototype development environment, but it is important that a cleaner object oriented design language is developed. This should be a superset of OO ACT ONE which incorporates semantics for processes, inter-process communication, nondeterminism and concurrency. Then the CPTs must be translated for designing in this new language. It is vital that the set of CPTs is widely expanded. This can only be done if the process of object oriented design is more thoroughly analysed: the most common design decisions must be identified and CPTs defined to model these decisions. Further, a means for designers to develop, record and re-use CPTs needs to be formulated. This should be incorporated as part of a set of tools for the analysis, synthesis and manipulation of designs.
- **Implementation**
  The implementation stage of FOOD is perhaps the weak point in the whole development strategy. Targetting designs to an informal semantics is not a suitable final step in a formal development method. A natural means of getting around this problem is to build an implementation language on top of the FOOD semantics. Then, the final implementation step can be given a

formal basis. It is important, for future development, that such a language has a concurrent semantics.

There are other areas of work, applicable to all stages of development, which arise out of the thesis:

- **Re-use**
  FOOD promotes re-use at all stages of development. The consequences of developing for re-use and with re-use need to be addressed. In particular, the heuristics for costing software development using FOOD need to be examined. Further, there needs to be some method for controlling the storing of, and access to, classes of re-usable behaviour at different levels of abstraction.
- **Evolving Method**
  Only through widespread application can FOOD become a software development method. Consequently, it is necessary that FOOD is used in a wide range of case studies, each of which learns from previous development. In this respect, FOOD needs to be extended to incorporate many of the *management* aspects which are common in the most popular software development techniques. Future work must atempt to derive a rational for software development method which can be incorporated in FOOD.

In conclusion, we believe that the development of a practical and effective industrial strength software development method, based on FOOD, is feasible in the future.