DIVISION OF COMPUTER SCIENCE

Experience of using Coad and Yourdon Object-Oriented Analysis and Design

Technical Report No. 148

A. Mayes & R. Barrett

November 1992

Experience of using Coad and Yourdon Object-Oriented Analysis and Design

Audrey Mayes and Ruth Barrett

November 92

1 Introduction

This document presents an analysis of the use of the Coad and Yourdon object-oriented methods detailed in the two editions of their book on analysis [1], [2] and the next volume in the series concerning design [3]. The purpose is to assess the suitability of the approach to object-oriented analysis and design and to clarify the differences between the two editions of the book covering analysis. The reader is assumed to be familiar with a version of the book on analysis, so only brief notes about the method are included. The following sections describe in detail the work carried out.

Section 2 a brief overview of the methods and the underlying philosophy.

Section 3 a comparison of the two editions of the method, summarizing the differences between them.

Section 4 details of the case study used for assessing the method.

Section 5 the results obtained using the first edition. The original candidate objects are named and reasons for their inclusion or exclusion from the final design given. A class dependency chart showing the final design is also included.

Section 6 the results obtained when using the second edition. Any differences between the versions are noted with the effect on the end product identified.

Section 7 comments on any differences between the two versions with respect to the reusability of the classes and the styles of program.

Section 8 comments on the effectiveness of the method and the notation used.

2 The method

The Coad and Yourdon view of objects is that they encapsulate attributes and exclusive services. Objects represent an abstraction of something in the problem space, reflecting the ability of the system to store information about it and/or interact with it. Classes describe groups of objects with the same attributes and also describe how to create new objects of the class. The object-oriented definition used includes the use of objects and classes, classification, inheritance and communication with messages.

Analysis is said to be the study of the problem and application domain. The emphasis of their method is heavily biased towards stored data and the operational procedures which are part of that domain and seems to ignore the actual application being developed. The method is intended to build on the best concepts from information modeling and object-oriented programming but is concerned with analysing the domain rather than systems analysis.

Initially the method, published in 1990, concerned only the analysis phase of development. Design and implementation are said to be an expansion of the results of object-oriented analysis. Both editions of the analysis divide the process into five stages, all of which feed into the final documentation set. The five stages of analysis are :

1. identifying objects
2. identifying structures,
3. identifying subjects,
4. defining attributes,
5. defining services.

The second edition of the method uses the same headings for the five analysis stages but contains some significant differences within the stages, together with changes in terminology and notation to aid the understanding of the method.

The book covering object-oriented design was issued about the same time as the second edition of Object-Oriented Analysis. This uses the same notation as version 2. Four components of the design are identified. These are

1. Problem domain component,
2. Human interaction component,
3. Task management component,
4. Data management component.

The results of the analysis feed in to the problem domain component. Nothing has been learnt about the other design components during the analysis. This is caused by the definition of object as an abstraction of something in the problem space.

The book gives little help with developing these other components beyond general advice obtainable in other, general design texts. The second component, the human interaction component involves studying the people who will use the system and what they will need the system to do. This phase appears to be part of the analysis of the required system and to require a repeat of the analysis using the human interaction as the problem domain. The separation into four components divides the problem well and should allow the components to be used in different systems. For example the classes used for the human interaction component could be designed to be generally useful.

3 Changes in the method

The differences between the two versions are considered to be important because different results are obtained when following the analysis guidelines. The short time span between the editions means that users might not realise the significance of the second edition and suggests that the method may not be fully developed.

3.1 Major changes

The changes listed here all affect the results of the analysis.

1. Operational procedures were added to the list of potential Class-&-Objects. This allows operational steps which must be remembered to be modelled. The attributes *'might include the operational procedure name, required authorisation level and a description of the steps in the procedure'*.

   However, this idea conflicts with other changes introduced in the second edition. The requirement for an object to contain stored data as attributes has been strengthened. In the first edition, it is stated that:

   > *an Object may have required Services, but no required remembrance (Attributes).*

   Whereas the second version states:

   > *Attributes describe values (state) kept within an Object.*

and

> *Conceivably an Object could have required Services, but no required remembrance.....The only time the authors observed Class-&-Object symbols with only Services inside, the analyst had been trying to use OOA notation with DFD style thinking;*

The conflict between the definition of an object and the definition of an operational procedure when represented by an object, causes confusion and indicates that the method is not fully developed.

2. An extra stage was added to the identification of subjects. This part of the analysis is only required for large systems, but the examples given show changes as a result.

3. The emphasis on the use of inheritance and aggregation has been changed. The first edition suggests that inheritance can be used to express commonality whereas the second places more emphasis on the is-A relationship. The idea of using aggregation to model the has-A relationship between classes is also introduced.

4. The advice on the positioning of attributes in a Gen-Spec structure has changed. The first version states:

   > *If an Attribute applies to a majority of specialisations, put it in the generalisation, then override it for the specialisations that do not need it.*

   The second edition states:

   > *... put an Attribute at the uppermost point in the Structure in which it remains applicable to each of its specialisations.*

   This obviously leads to different analysis results in the case studies and can cause confusion.

5. The view on multiple inheritance has also changed. The first edition states:

   > *the net effect (of multiple inheritance) is a cumbersome model that in practice obscures the actual problem space structure.*

   The second edition states:

   > *the lattice structure (showing multiple inheritance) could become unwieldy with more and more specialisations. This has not come up in practice.*

   This change in view leads to different results in the example case studies.

6. The definition of instance connections has changed. In the first version the definition was

   > *a mapping from one instance to another.*

   The second definition states:

   > *An instance connection is a model of problem domain mapping(s) that one object needs with other objects, in order to fulfill its responsibilities.*

   The revised definition seems to mean almost the same as the definition of a message connection identified as part of the service layer.

   > *A message connection models the processing dependency of an object, indicating the need for services in order to fulfill its responsibility.*

   The differences appear to be that a message connection has explicit direction whereas an instance connection has information about the multiplicity. It might be better to combine the two notations to reduce the number of lines on the diagram.

7. The method for defining services shows many changes. The object life histories are no longer considered. The first stage is to identify the object states which cause a change in behaviour. The only other consideration is the identification of the required services, that is the occur, calculate and monitor services. This is the same as in the first edition. The case studies in the two editions of the book show differences in the services required. As a result of all the previous changes it is difficult to ascertain the reasons for this.

3.2 Terminology and notation changes

1. It can be seen above that the first stage of analysis is the identification of objects. This caused confusion when, at the design and implementation stages, the objects became classes. The second version identifies Class-&-Objects instead of objects. A class is defined as:

   > *a description of one or more Objects with a uniform set of Attributes and Services, including a description of how to create new Objects of the Class.*

   This concept helps ease the transition from analysis to design but does not affect the results of the analysis.

2. Another change concerns the naming of structures. The Classification Structures of the first version are now called Generalisation-Specialisation or Gen-Spec structures and assembly structures are called Whole-part structures. This has no effect on the method.

3. As part of the definition of attributes in the first version, each attribute is categorised as descriptive, definition, always derivable or occasionally derivable. This information was not used further and the phase has been dropped from the second version.

4. Some of the notation used in the diagrams has been simplified in the second version as can be seen by comparing the notation in Figs 1 and 3. The instance connections and Whole-part connections are labelled with numbers instead of bars and crows feet. The parts of a whole part structure are also shown as joined separately to the whole. This makes the diagrams both easier to draw and easier to understand.

4 The Case Study

The system is required to store all the information required to allow a gardener to plan the growth of crops. There is a requirement to plan the maturation of crops, for example to avoid crops maturing when the gardener is on holiday or when there is a strong likelihood of attack by pests. The variation in planting time and growth period of a crop depending on the region of the country needs to be catered for. The gardener also wishes to be able to divide the garden into plots to allow for crop rotation to be carried out. Reports should be produced to give relevant information about the productivity of the vegetable garden. The gardener also requires to be able to add or change the information concerning crops.

5 Experience using version 1

5.1 The Analysis

The five stages were not followed sequentially. It was found necessary to define the attributes before the structures could be identified. There was a large amount of iteration needed to produce the final model shown in Fig 1.

The following list contains all the candidate objects identified by looking for structure, other systems, devices, events remembered, roles played, locations and organisational units. The fate of these objects during development is noted.

Crop This was retained as an object through to the final design. It underwent some changes and finished as part of an inheritance hierarchy with plant as the base class. This is shown in Fig 1.

Pest This was retained as an object.

Plan This became part of the inheritance hierarchy derived from plant.

Growing crop This also became part of the inheritance hierarchy derived from plant.

Report This was discarded as an object because it did not meet the criterion of needing remembrance as it is generated from other data.

Garden This was retained as an object. It became an assembly structure with growing areas as its component parts.

Growing area This was retained as an object.

Calendar This was retained as an object.

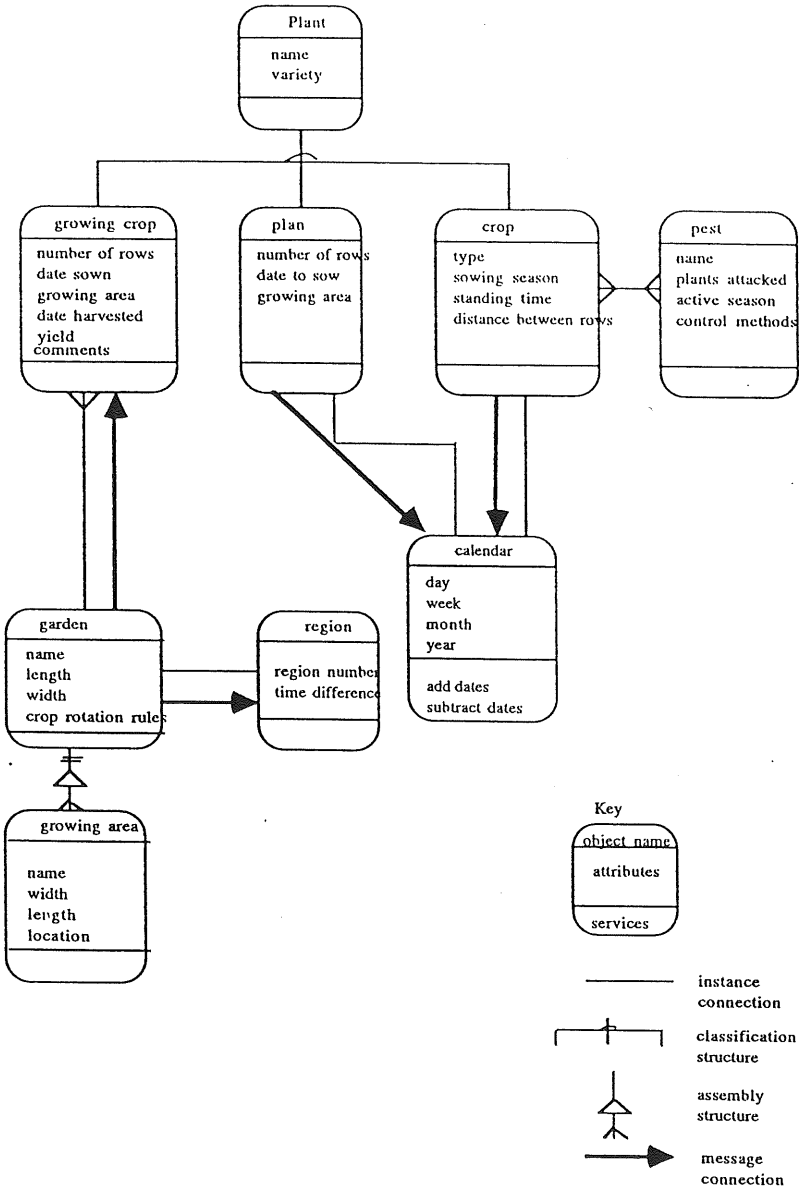
Input and output The decisions about this were deferred to the design stage because they involve hardware devices that may be changed.

Gardener This was discarded because no information needed to be stored about the gardener.

Region This was retained as an object.

Seed sown, Crop transplanted and Crop harvested These three objects concerned events remembered and were discarded because they duplicated the information held elsewhere.

Fig 1 Result of Object-Oriented analysis1st version



The inheritance hierarchy was discovered when the attributes of the objects were defined. 'Crop', 'growing crop' and 'plan' objects were found to require some of the same information. The common attributes were separated out into a new class, 'Plant', from which the others were formed by inheritance. This follows the advice given which is to use inheritance *'to explicitly express commonality of Attributes and Services'* but leads to the use of inheritance without an is-A relationship. A plan is definitely not a plant. There is no suggestion from Coad & Yourdon in the first edition that this use of inheritance should be suspect.

Also included under the heading of defining attributes is the identification of instance connections. These are defined as *'a mapping from one instance to another'*. This was assumed to mean the relationships between objects. The lines were drawn as if the model was an entity relationship diagram. For example, the relationship between crops and pests is a many to many relationship. The attributes were also categorised as descriptive, definition, always derivable or occassionally derivable. For example, the attribute 'type' in the object class is a definition attribute because the value can be applied to more than one instance of the object. The other attributes are all descriptive. There were no derivable attributes as they were discarded in compliance with the statement that *'derived results muddy the picture'*. However, the information gained by categorising the attributes was never used.

The final stage is to define the services. The instructions state that all objects require 'occur' services, that is to add, change, delete and select. Some require calculate services and/or monitor services. In the case study it was decided that only the 'calendar' had to provide calculate services for use by other objects. There were no monitor services required. The object life histories are also defined to help identify services. The decision was made that most of the objects have simple life histories relying only on the standard occur services. The only object required to provide services is the calendar object. The final strategy in defining services is to look at the state-event-responses required by the system. This stage involves looking at the different actions that are required when the system is in different states. It was decided that the system did not have any states which required a different response.

Message connections are then added. These are mappings from one instance to another to show where a message is sent, for example from plan to calendar. The instance connections are said to require a message connection between the objects. In the case of pests and crops, it was decided that a message connection was not needed because the two objects do not need to communicate. It is also necessary to add message connections to ask an object to carry out some processing, for example, the calendar object would have messages sent to it to ask it to perform calculations on dates.

The results of the analysis are shown in Fig 1. The diagram combines the results of the five stages of analysis, showing the classification structures and assembly structures identified as well as the instance and message connections

between all the objects. The diagram resembles an entity relationship diagram combined with a data dictionary and some processing requirements. The use of inheritance is another addition. This diagram ties in exactly with their view of the object-oriented approach.

5.2 The Design

The process of object-oriented design is said, by Coad and Yourdon, to be *' a progressive expansion of the model'*. This was not the case. The model from the analysis stage gave little hint of the major functions required by the system. The only way found to identify the functional requirements was to draw high level data flow diagrams. The objects identified during analysis became classes at this stage. New classes were added to provide the high level functions identified by the data flow diagrams. These classes are called 'record', 'amenddata' and 'plan'. They have no stored data and therefore no state. The information required to carry out these functions was provided by the 'occur' services of the objects.

The system structure is shown in Fig 2. It can be seen that it closely resembles a functional breakdown of the problem. Most of the classes identified during analysis are simple abstract data types (ADT's) providing the only standard occur services to access the attributes. The exception is the calendar class which provides extra services. Most of the messages are passed down in a hierarchy from the higher level objects to the ADT's at the bottom. The message connections between the lower level objects reflect the fact that instances of objects are nested.

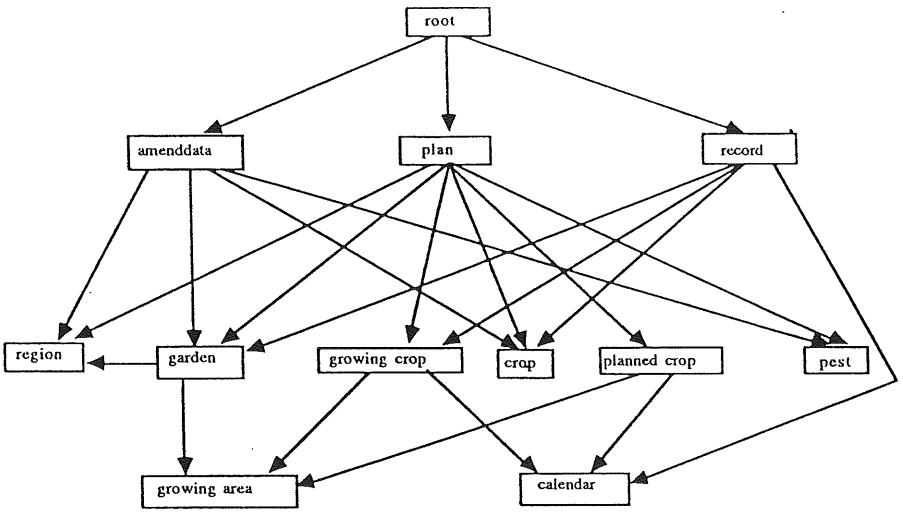
6 Experience using version 2

6.1 The Analysis

The main difference between the two versions at the initial stage is the advice to include operational procedures in the list of potential Class-&-Objects. This suggests that 'report ' might be a Class-&-Object this time. Despite the confusion noted in section 3, it was decided that 'report' is a valid object. This decision allowed more of the requirements to be captured. The same arguments apply to the production of a plan and the carrying out of updates to the crop and garden data. This gave three Class-&-Objects - 'report', 'plan' and 'update'

which were not present in the first attempt at the analysis.

Fig 2 System design 1st version



The inclusion of these Class-&-Objects had most effect on the process of identifying services. This was because the production of plans and reports require services to be provided. The changes at this stage lead to changes in the attribute layer and structure layer.

The suggested strategy for defining services is initially to identify object states. An object state is defined as an attribute value which reflects a change in object behaviour. The 'growing area' has states. It can be full or not-full and has a valid crop associated with it. The gardener should be allowed to enter information which these states suggest are invalid. For example crops can be

planted closer together than recommended which would mean that the area was not full. Crops can also be planted in the wrong place. The processing of these conditions was allocated to the 'plan' class.

An attempt was made to analyse the garden case study without requiring a change in state to cause a change in behaviour. The result was a model with tight coupling between two pairs of classes, planned with planned section and growing with growing section, as can be seen in Fig 3. Each member of the pairs of Class-&-Objects contains a variable of the other member as one of its attributes. This is required to make a link between where a crop is grown and the crop itself. Tight coupling of this type can lead to problems when considering updates to the data in the system. From this case study it seems unwise to neglect the requirement for a change in state to cause a change in behaviour.

A further attempt was made to produce a model without the tight coupling. The crop object was simplified to provide the standard "occur" services only. The garden and growing areas remained whole-part structures. The definition of a section was changed to include an attribute crop. Again the section was used as the base class for two subclasses, namely growing section and planned section. These added the required details.

There are several possible ways to model the information required by 'planned section' and 'growing section'. The extra information could be represented by abstracting it into a separate Class-&-Objects or by adding each piece of information as an attribute. The abstractions represented by the extra Class-&-Objects were not realistic in the domain so the simpler model was chosen. Fig 4 shows the results of this analysis.

6.2 The design

The chosen implementation languages were Eiffel or C++. The results of following Coad and Yourdon's suggested design strategy [3] are:

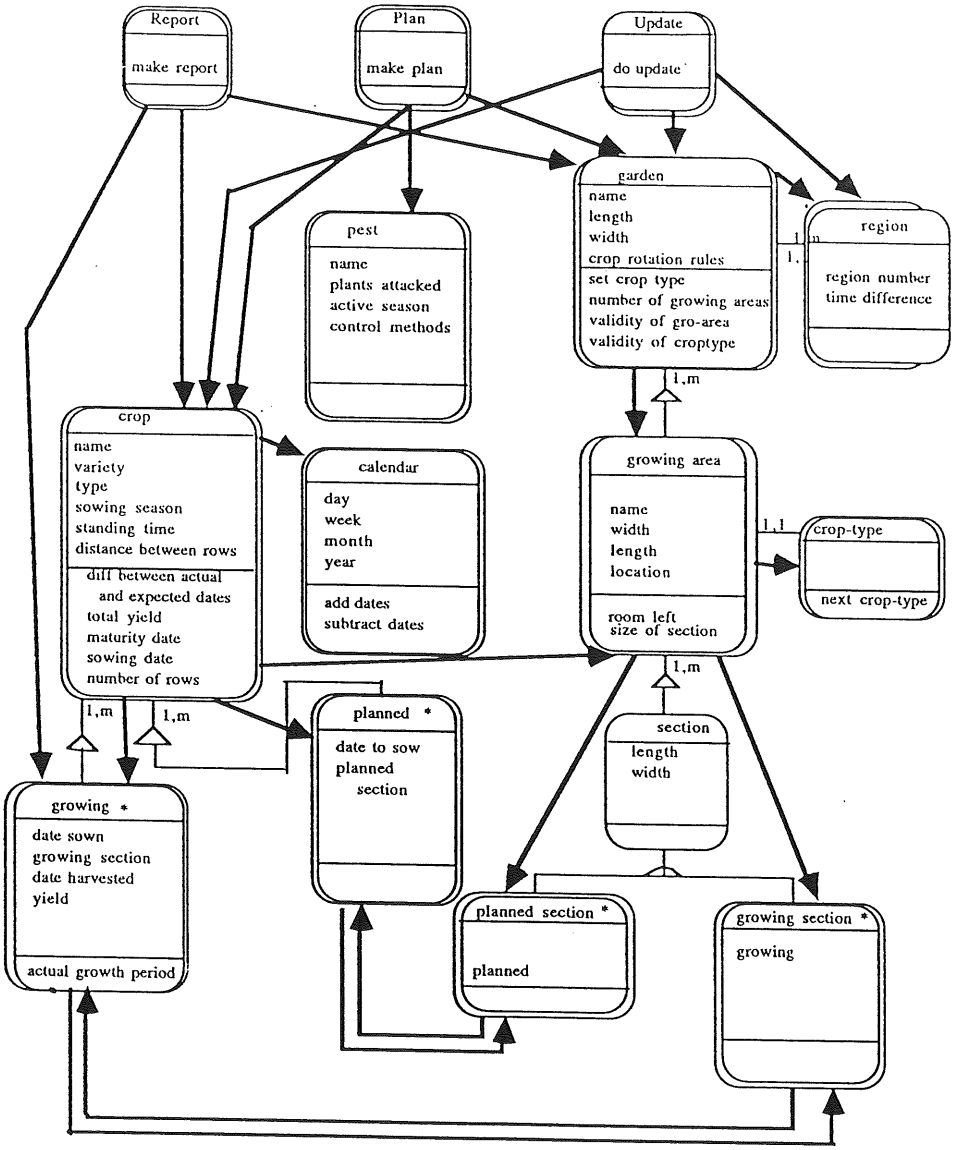
1. The Problem Domain component. This builds directly on the OOA results with any necessary changes to accommodate any limitations of the language, such as inheritance not being available. There are no language problems of that type when using Eiffel or C++.

2. The Human Interaction component. A simple menu driven interface was chosen. The code for this is to be located in the root class for the system.

3. The Task Management component. The system has no concurrency so a task management component is not required.

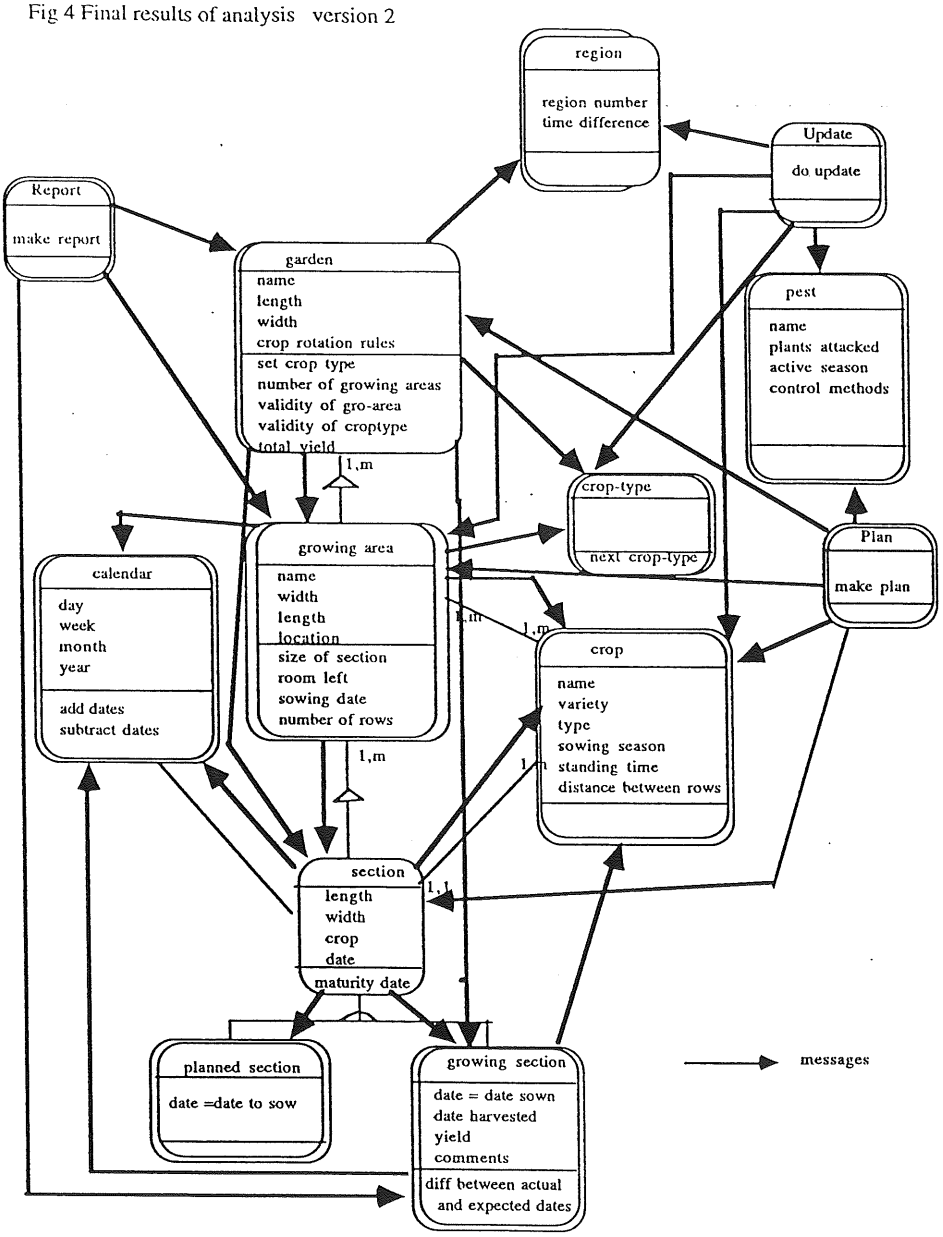
4. Data Management component. Eiffel's environment class will be used to store any necessary objects. The C++ storage facilities are yet to be explored.

Fig 3 Interim result of Object-oriented analysis 2nd version



* shows classes involved in tight coupling

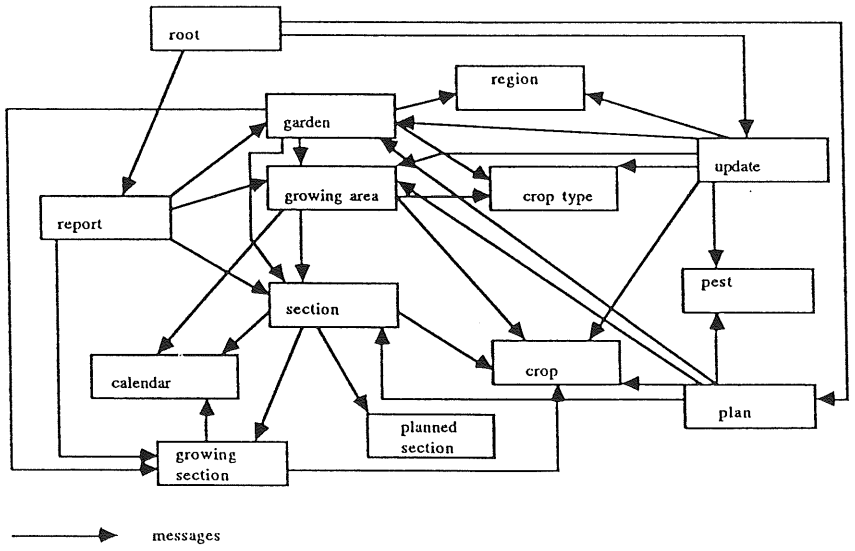
Fig 4 Final results of analysis version 2



The final design of this version of the system is shown in Fig 5. This is, I believe, an object-oriented style having the processing distributed throughout the system not concentrated in a main module. It has the same number of classes as the first version but the processing is distributed between the classes. This results in the classes being more complex and messages being passed between more classes to provide the same functionality. The diagrams reflect this, giving the impression of a more complex system. Many of the functions required to produce reports and plans are provided by other objects. The report and plan classes are, therefore, simpler in comparison to those with similar names in the original design.

In the first design the calculations were performed by the objects requiring the answers, not the ones which had most or all of the required information. The 'intelligence' of the system is more evenly spread throughout the system in the final design. The diagram is very difficult to understand because there are so many messages being passed. The introduction of subject layers simplified the diagram but lost some the information concerning the messages being sent so was abandoned.

Fig 5 System design final version



7 Reusability

The reanalysis and design of the system following the advice in the second edition removed the class hierarchy derived from plant. This was one of the simple classes identified as being reusable in another unrelated system, a pollen count guide. The inclusion of many services in the classes also makes them more application specific and therefore less reusable.

Reusability could be increased by factoring out any attributes and services which are not application specific into separate classes and then forming the application specific classes from them by inheritance. Alternatively, an instance of the general class could be declared in the application specific class. For example, a general class Garden could be produced as follows

Class Garden

Attributes	Services
name	occur services only
width	
length	

an application specific VegGarden class could be formed from this by

1. using inheritance

   Class VegGarden inherit Garden

Attributes	Services
Region	validity of growing areas
GrowingArea	number of growing areas
set crop type in growing area	
	validity of croptype for growing area
	calculate total yield.

2. declaring an instance

   Class VegGarden

Attributes	Services
Garden	
Region	validity of growing areas
GrowingArea	number of growing areas
	set crop type in growing area
	validity of croptype for growing area
	calculate total yield.

In this example the first method would probably be better. There are two reasons for this. The first is that a VegGarden is-A Garden. This is the relationship that inheritance is designed to give. The second reason is that the use of inheritance allows the features of the 'garden' to be accessed directly. For example, when validating the size of a growing area, the garden length could be accessed simply as 'length'. In the alternative method, where a variable of type Garden is declared, the length must be accessed as <garden name>.length.

8 Discussion and Conclusions

8.1 Applicability of the method

The method detailed in the first edition suggests that objects must contain stored information but not derived data. Careful adherence to this principle led to some of the requirements being missed because the need for the derived data was lost and the eventual design to be a top level functional design. The missing requirements included the need to produce plans and reports along with the information they should contain. There is a sentence in the book on analysis,

> *However, the system does need to monitor the user, responding to requests and providing timely information.*

At the time this did not appear to mean the production of plans and reports. A different interpretation would have given a user object with services plan, report and update but no required remembrance and therefore its inclusion as an object would be questioned.

The second edition introduced the concept of operational procedures being potential Class-&-Objects. Required remembrance in a Class-&-Object was made compulsory. In order to comply with this, operational procedures were allocated the procedure name and the description of the steps for their attributes. This conflicts with the idea that attributes '*describe values (state) within an object*'. The inclusion of these procedures as objects did lead to the requirements being defined more completely. As an aside, this also appears to conflict with Meyer's guidelines [3]. He warns against designing classes which are really functions and whose role can only be described as '*This class does ...*'. He states that:

> *'a class should not do something but offer a number of services on objects of a certain type'*

This leaves some confusion about what to include as a Class-&-Object. The inclusion of required output as a Class-& Object would have resulted in the requirements being more fully ascertained. It is also necessary to define what should be contained in a report. A report by its nature will contain mostly, if

not exclusively, derived data and data which forms part of the state of another object. In some systems it is necessary to produce different types of report, for example in the garden planning case study, one report could contain a list of varieties grown in a year and another contain all the information about total yields and problems as well. It is impossible to identify the requirement for this information if derived data is excluded from the analysis. Thus, the data required in the report should be specified as part of the analysis. Any decision about whether to store the information in the report class and update this each time a change is required, or to calculate the result as and when required, is a decision which should be left until the implementation stage. The insistence that derived data should not be stored makes the method similar to the production of entity relationship diagrams.

Another problem with the Coad and Yourdon approach is that the meaning of instance connection is not clear. Their meaning appears to be confused with that of message connections. Communication with messages is one of the features used by Coad and Yourdon to define object-oriented systems, so this confusion should not be allowed to exist.

The garden case study was not well modelled by this analysis and design method. The sample case studies were simulations which have a more clearly defined boundary and require less user input. Some of the objects in a simulation also have states, for example a sensor can have one of three states, off, standby or monitor. This was not the case with the objects in the garden case study. The garden case study allowed the user to choose which crops to grow. The method might have been more successful had the type of crops and the required number of rows of each crop been specified in the requirements instead of allowing a variable number of rows and crops to be defined by the user. The amount of user interaction required would have been reduced. It would then have been possible to set up a running system for the generation of plans from the analysis results.

8.2 Notation

Most of the notation used in the second version is clear and simple. The exception is the notation used to depict inheritance or Gen-Spec structures. All the Class-&-Objects produced by one layer of inheritance from the same generalised class are placed on the same level. They are joined to the generalised class via a single line. This might be taken to imply a relationship between Class-&-Objects on the same level in the hierarchy. They are in fact related to the higher Class-&-Object but not to each other. It might be clearer to adopt the notation used by Meyer [4]. This shows a separate link from each specialisation to the general class.

8.3 Final comments

There are substantial differences between the guidelines in the first and second editions which must be borne in mind when following the method.

Finally, I agree with the statement in the Acknowledgements that

> *'The second edition is far better ...'.*

I would recommend using the second edition and not the first. However, I feel that there are still some inconsistencies and confusing statements which need careful thought in order to be successful when following the guidelines. The following sentence, included in both editions of the book on analysis, should be carefully adhered to.

> *Use this book as a starting point for applying OOA – tailoring and expanding the method to suit your specific organisation or project needs.*

The interpretation of the quotation is left to the reader.

References

[1] P. Coad and E. Yourdon. *Object-Oriented Analysis.* Prentice Hall, Inc, Englewood Cliffs, New Jersey, 1990.

[2] P. Coad and E. Yourdon. *Object-Oriented Analysis.* Prentice Hall, Inc, Englewood Cliffs, New Jersey, second edition, 1991.

[3] P. Coad and E. Yourdon. *Object-Oriented Design.* Prentice Hall, Inc, Englewood Cliffs, New Jersey, 1991.