An Object-Oriented Database Methodology

for application development with extended relational or object-oriented DBMS

A thesis presented in partial fulfilment of the requirements

for the degree of

Master of Science in Computer Science at Massey University

Benny Liew

1992

Acknowledgement

Firstly, I would like to thank Mr Roger Tagg, my thesis supervisor, for helping me to draft out the contents and using the two case study examples from his 57.ODB(Object-Oriented Database) paper. His advice and guidance throughout is greatly appreciated. Many books and articles were borrowed from him for use in this thesis.

Next, I would like to express my appreciation to Dr. Daniela Mehandjiska-Stavreva, my alternate supervisor for her helpful comments on the structure, written style and presentation of this thesis.

I would like to thank Massey University Library for using the facilities for my literature search, especially the Library Interloan staff for their excellent services. A total of eight books and more than a dozen of journal articles were requested within New Zealand and overseas. Some of these articles came as far as Canadian universities' libraries.

Thanks go to Mr. Colin Eagle and Mr. Richard Rayner for their excellent support of Postgres on our Sun Microstations network; and also to Mr Todd Cochrane, our PhD student of Computer Science Department, for his past assistance. I wish Todd every success for his PhD research work.

Lastly, I would like to thank Mum and Dad for so many years of upbringing.

Benny Liew, DipSc(CompSc), TechDip(Elect Engg), MSIET

Master of Science(Computer Science) candidate,
Massey University,
Dept. of Computer Science,
Palmerston North,
New Zealand.

3/12/92

Abstract

Recently development methodologies have been proposed which describe themselves as "Object Oriented". While all of them offer approaches to extended data and behavioural modelling, none of them seem fully adequate to address the total concept of object-oriented development. They often do not provide constructs which lead to the use of databases, nor do they always recognise the shift from sequential to prototyping style which is inherent in much object-oriented technology.

The objective of this thesis is to lay a framework for an object-oriented methodology suitable for OODBMS. Details of conventional methods for developing database applications, and of the recent OO methods, have been examined and compared in order to propose a coherent set of tasks and deliverables. Account has also been taken of designing for re-use, which has been one of the main selling points of the OO approach.

The proposed methodology attempts to address related side issues, with particular focus on object concurrency, which seems particularly thinly covered in many of the current proposals. Many other side issues are also mentioned, but due to time constraints, they are not given any further discussion. The topic is an extremely multi-disciplinary one, and a very wide range of expertise would be necessary to do justice to all these aspects.

Mapping of the new methodology has been tried on two case study examples using Postgres and Ontos. Postgres is an extended relational DBMS developed as a research prototype at University of California, Berkeley. Ontos is the commercial object-oriented DBMS marketed by Ontos Incorporated, Burlington, Massachusetts. Some details of these implementation examples are included.

Rationale for the Research

Object-oriented technology has gained much popularity recently, but methodologies for its use are still at an immature stage. There are many proposed developments of the OO paradigm by pioneers in this area. Examples are Booch[4], Coad & Yourdon[7,8], Shlaer & Mellor[55], and Meyer[19]. These methodologies are often fairly general in nature and do not specifically address the needs of the OO paradigm to some special areas, such as databases.

On the other hand, pioneers in OODBMS like Zdonik and Maier[30], Stonebraker[56-58], Won Kim[15] and Lochovsky[14] and Rolland & Brunet[52] concentrate more on the requirements and implementation of a specific kind of OODBMS.

The concepts of Object Repository and reusability of software have also been subjects of discussion lately. There are many advantages associated with OO prototyping[20].

So far, there has not been an OO paradigm that covers the whole development cycle of an OODBMS, although there exists many OODBMS tools. This thesis aims to propose a total, unified paradigm applicable to OODBMS from feasibility through analysis and design to implementation stage. It emphases particularly on prototyping and reusability through the use of class libraries and repositories so as to support modern practices.

One way of doing this is to review all the currently proposed OO methodologies to gain an understanding of each in terms of techniques and diagrams used. Sometimes, different conventions and terms are used by different authors to represent the same semantic meaning. It is necessary to understand why such individual approaches are used.

An OO methodology should also have stages of development just like conventional software development using the functional approach. In addition, steps for each phase of development is prescribed.

Extended relational and object-oriented databases are examined, and their common features extracted. This is necessary for the formulation of an OODBMS methodology of general applicability.

The topic of object-oriented prototyping as applied to application development in OODBMS is also discussed. OO prototyping enables quick development of OO database applications and this technique should be used.

Thesis structure

This thesis is made up of eight chapters.

The first chapter of the thesis takes a look at past methodologies for software development and the evolution of present ones. It briefly describes the existing methodologies that are well accepted and practised by current software houses. It then describes the emerging methodologies of the 1990s such as RAD and the IBM AD/Cycle-Repository. Finally, some of the better-known OO approaches are briefly introduced and summarised.

Chapter 2 discusses the required features of an OODBMS methodology. These concepts are taken from various sources and each one is given a brief description. Later, in Chapter 4, some of them are selected to be applied in the proposed methodology.

Chapter 3 gives a brief description of existing methodologies using the object-oriented paradigm. It is important to note that not all of them are equally suitable for all types of implementation. For instance, Rolland & Brunet's O* Model is particularly suitable for OODBMS because it supports a lot of database concepts. A comparison is made on the methodologies covered in the literature search. The similarities, differences, strength and deficiency of each is pointed out in a matrix.

Chapter 4 is the proposal of a new methodology for OODBMS. The new proposal stresses 4 stages of development and the exploitation of object-oriented prototyping for object iteration. The techniques and diagrams adopted in each step have been described in Chapter 3.

Chapter 5 examines the application of the proposed paradigm as applied to extended relational database. Postgres is chosen as the extended relational database used to illustrate a case study example.

Chapter 6 examines the application of the proposed paradigm as applied to OODBMS. Ontos is used as the object-oriented database to illustrate a case study example.

Chapter 7 offers some conclusions. It also comments on the application of the proposed methodology to the two different types of DBMS. Further possible work on the enhancement of the new methodology is also suggested.

Five sections are included in the Appendices.

Section A gives a brief description of existing fourth generation languages(4GL) for OODBMS. Samples of the user interfaces of O2, GemStone, and GOOSE are shown. GOOSE is a graphical interface for an OO database schema environment created at Georgia Institute of Technology.

Section B discusses concurrency control protocols in OODBMS.

Implementation details of Postgres Case Study example are provided in Appendix C. Implementation details of Ontos Case Study example are provided in Appendix D.

Finally in Appendix E, current research areas relating to both types of DBMS are discussed.

The bibliography contains all the books and journal articles used in the formulation of the proposed methodology.

Table of Contents

Chapters			Page
1.	Review of Software Development Methodology		1
	1.1	Introduction	1
	1.2	Mainstream Methodologies	1
		1.2.1 STRADIS	1
		1.2.2 Information Engineering	1
		1.2.3 SSADM	2
		1.2.4 JSD	3
		1.2.5 MERISE	3
		1.2.6 SSA	3
		1.2.7 Deficiency of mainstream methodologies	3
	1.3	Current Trend	4
		1.3.1 Rapid Application Development(RAD)	4
		1.3.2 IBM AD/Cycle - Repository	7
	1.4	Object-Oriented Methodologies	8
		1.4.1 Booch Methodology	9
		1.4.2 Rolland & Brunnet O* Model	9
		1.4.3 Coad & Yourdon OOA and OOD	9
		1.4.4 GE Labs Object Modelling Technique	9
		1.4.5 Bertrand Meyer OO Methodology	9
		1.4.6 Ivar Jacobson Object-Oriented Development	10
		1.4.7 Henderson-Sellers Object-Oriented Life Cycle	11
		1.4.8 Summary of Object-Oriented Methodologies	12
	1.5	Conclusions	13
2.	Required Features of an OODBMS Methodology		14
	2.1	Support for development in stages	14
	2.2	Class Identification	14
	2.3	Relationships Identification	14
	2.4	Behaviour modelling	14
	2.5	User Interface Development	15
	2.6	Digramming conventions	16
	2.7	Object-Oriented CASE Tools	16
		2.7.1 Tools for analysis and design(front-end)	16
		2.7.2 Tools for implementation(back-end)	16

2.8 Object-Oriented Prototyping 17
2.9 Object Repository 17
2.10 Support for Reusability 17
2.11 Support for use of OOPL 19
2.12 Support for use of OODBMS features 19

3. Review of Current Object-Oriented Methodologies 20
3.1 Booch Methodology 20
3.2 The Database Object Model by Rolland & Brunet 23
3.3 Coad & Yourdon's Methodology 25
3.3.1 Object-Oriented Analysis 26
3.3.2 Object-Oriented Design 29
3.4 Object-Modelling Technique(OMT) 30
3.5 Comparison of Methodologies 32

4. A Proposed Object-Oriented Methodology for OODBMS 35
4.1 Feasibility Study 37
4.1.1 Overall application purpose 37
4.1.2 Statement of interactions 38
4.1.3 Performance requirements 38
4.1.4 Failure conditions 38
4.1.5 Cost/Benefit analysis 38
4.2 Object-Oriented Analysis 38
4.2.1 Generating a description of the problem domain 39
4.2.2 Constructing the Analysis Model 39
(a) Identify Classes 39
(b) Identify Relationships 41
(c) Structure the Static Aspect 41
(d) Structure the Dynamic Aspect 44
(e) Structure the Static/Dynamic Interaction 47
4.2.3 Object-Oriented Prototyping 47
4.3 Object-Oriented Design 49
4.3.1 Identification of supporting classes 49
4.3.2 Identification of reusable library classes 50
4.3.3 Tailoring the class structure for reusability 50
4.3.4 Choosing a concurrency control protocol 50
4.3.5 Iteration of classes 50
4.3.6 System Design 52

4.4 Implementation 52
4.4.1 Mapping to the target language 53
4.4.2 Implementing the application 53
4.4.3 Querying the database 54
4.5 Maintenance of the application 54
4.6 Summary 54

5. Application of the proposed methodology to Postgres Case Study 55
5.1 Features of Postgres 55
5.2 Feasibility Study 55
5.2.1 Overall application purpose 55
5.2.2 Statement of interaction 56
5.2.3 Performance requirements 56
5.2.4 Failure conditions 56
5.2.5 Cost/Benefit analysis 56
5.3 Object-Oriented Analysis 57
5.3.1 Generating a description of the problem domain 57
5.3.2 Constructing the Analysis Model 57
5.3.3 Object-Oriented Prototyping 59
5.4 Object-Oriented Design 60
5.4.1 Identification of supporting classes 60
5.4.2 Identification of reusable library classes 60
5.4.3 Tailoring the class structure for reusability 60
5.4.4 Choosing a concurrency control protocol 60
5.4.5 Iteration of classes 61
5.4.6 System Design 61
5.5 Implementation 61
5.5.1 Mapping to the target language 61
5.5.2 Implementing the application 61
5.5.3 Querying the database 61
5.6 Summary 61

6. Application of the proposed methodology to Ontos Case Study 62
6.1 Features of Ontos 62
6.2 Feasibility Study 62
6.2.1 Overall application purpose 62
6.2.2 Statement of interaction 63
6.2.3 Performance requirements 63

6.2.4 Failure conditions 63
6.2.5 Cost/Benefit analysis 64
6.3 Object-Oriented Analysis 64
6.3.1 Generating a description of the problem domain 64
6.3.2 Constructing the Analysis Model 65
6.3.3 Object-Oriented Prototyping 67
6.4 Object-Oriented Design 68
6.4.1 Identification of supporting classes 68
6.4.2 Identification of reusable library classes 68
6.4.3 Tailoring the class structure for reusability 68
6.4.4 Choosing a concurrency control protocol 69
6.4.5 Iteration of classes 69
6.4.6 System Design 69
6.5 Implementation 69
6.5.1 Mapping to the target language 69
6.5.2 Implementing the application 70
6.5.3 Querying the database 70
6.6 Summary 71

7. Conclusion 71
7.1 Author's comment on the newly proposed methodology 71
7.2 Comparison of Development for Postgres and Ontos 72

Appendices
A. OO Prototyping Tools 73
B. Concurrency Control in OODBMS 75
C. Implementation details of Postgres Case Study 80
D. Implementation details of Ontos Case Study 98
E. Future Directions of OODBMS 119

Bibliography 123

List of Figures

		Page
Fig. 1.1	Stage Framework of Information Engineering Methodology	2
Fig. 1.2	The Rapid Iterative Production Prototyping	6
Fig. 1.3	IBM AD/Cycle - Repository	8
Fig. 1.4	Class/Module Life Cycle	10
Fig. 1.5	Object-Oriented Systems Development	11
Fig. 1.6	Fountain Model	12
Fig. 2.1	Model of Reuse in Object-Oriented Development	18
Fig. 3.1	Booch's Class Diagram	21
Fig. 3.2	Template for the class Alarm	21
Fig. 3.3	State Transition Diagram for the class Alarm	22
Fig. 3.4	Booch's Object Diagram	23
Fig. 3.5	Overview of Rolland & Brunet's Object Definition	24
Fig. 3.6	A sample of the O* Model textual description	25
Fig. 3.7	Using a class as a generalisation	27
Fig. 3.8	Using a class object as a generalisation	27
Fig. 3.9	Person Gen-Spec structure, as a lattice	28
Fig. 3.10	"Part-of" structure of Aircraft & Engine	29
Fig. 3.11	"Part-of" structure of Organisation & Clerk	29
Fig. 3.12	Four components and five layers	30
Fig. 3.13	Matrix for the comparison of the methodologies	34
Fig. 4.1	Development stages of the new methodologies	36
Fig. 4.2	Effort as a function of time	39
Fig. 4.3	Association Object	40
Fig. 4.4	Extended E-R diagram	42
Fig. 4.5	Class Descriptor for the class Reservation	43
Fig. 4.6	Object Communication Diagram	44
Fig. 4.7	State Transition Diagram	45
Fig. 4.8	Event Trace Diagram	47
Fig. 4.9	Mapping Principles for Analysis	48
Fig. 4.10	Mapping Principles for Design	51
Fig. 4.11	Module Diagram	52
Fig. 4.12	Mapping Principles for Implementation	53
Fig. 5.1	Class Diagram for Postgres Case Study	58

Fig. 5.2	Class Descriptor for Postgres Case Study	59
Fig. 6.1	Class Diagram for Ontos Case Study	66
Fig. 6.2	Class Descriptor for Lakes	66
Fig. 6.3	State Transition Diagram for class Measuring_point	67
Fig. 6.4	Object Communication Diagram for Ontos Case Study	67
Fig. B.1	Dynamic Interrelations Diagram	75

Chapter 1 : Review of Software Development Methodologies

1.1 Introduction

Early 1960s' information systems were not built according to any formal methodology[1,25,26]. Analysis work was limited and the emphasis was towards programming. Implementation of information systems was mainly restricted to programming and was based on fixed file structures.

In the late 1960s and 1970s, software development was based largely on function-oriented design, whereby the design is decomposed into a set of interacting units, each having a clearly defined function. Large software systems have been built using this technique and thus it has stood the test of practice. However, the need to develop and maintain large complex software systems using advanced techniques such as databases in a competitive and dynamic environment drove interest in better approaches to software design and development. In the 1980s, this led to a batch of formal "methodologies", which have incorporated some blend of function-oriented and data-oriented approaches.

1.2 Mainstream Methodologies Description

Some of the well-known methodologies that have gained widespread acceptance for information systems development today are introduced below :

1.2.1 STRADIS : Structured Analysis, Design and Implementation of Information Systems

This is based on the work of Gane & Sarson. The development of this structured systems approach to analysis came as a result of the earlier development of a structured approach to design. The structured design concepts were first proposed in 1974 by Stevens, Myers and Constantine (1974) and were later developed and refined by Yourdon and Constantine (1978), and Myers(1975, 1978). Data flow diagrams are constructed to represent the existing system and its interfaces.

1.2.2 Information Engineering

The term Information Engineering[17,18] originates from Clive Finkelstein who described a data modelling methodology he developed in Australia in the late 1970s, although the details have developed from a variety of sources including Ian Palmer of CACI in the UK, and James Martin in the USA. Information Engineering is now a comprehensive methodology covering all aspects of the software life cycle. It is evolving in the area of automated tools and the development of the methodology to support 4GL. The methodology is divided into four levels, within which there are seven stages, each with different objectives as shown in Fig. 1.1.

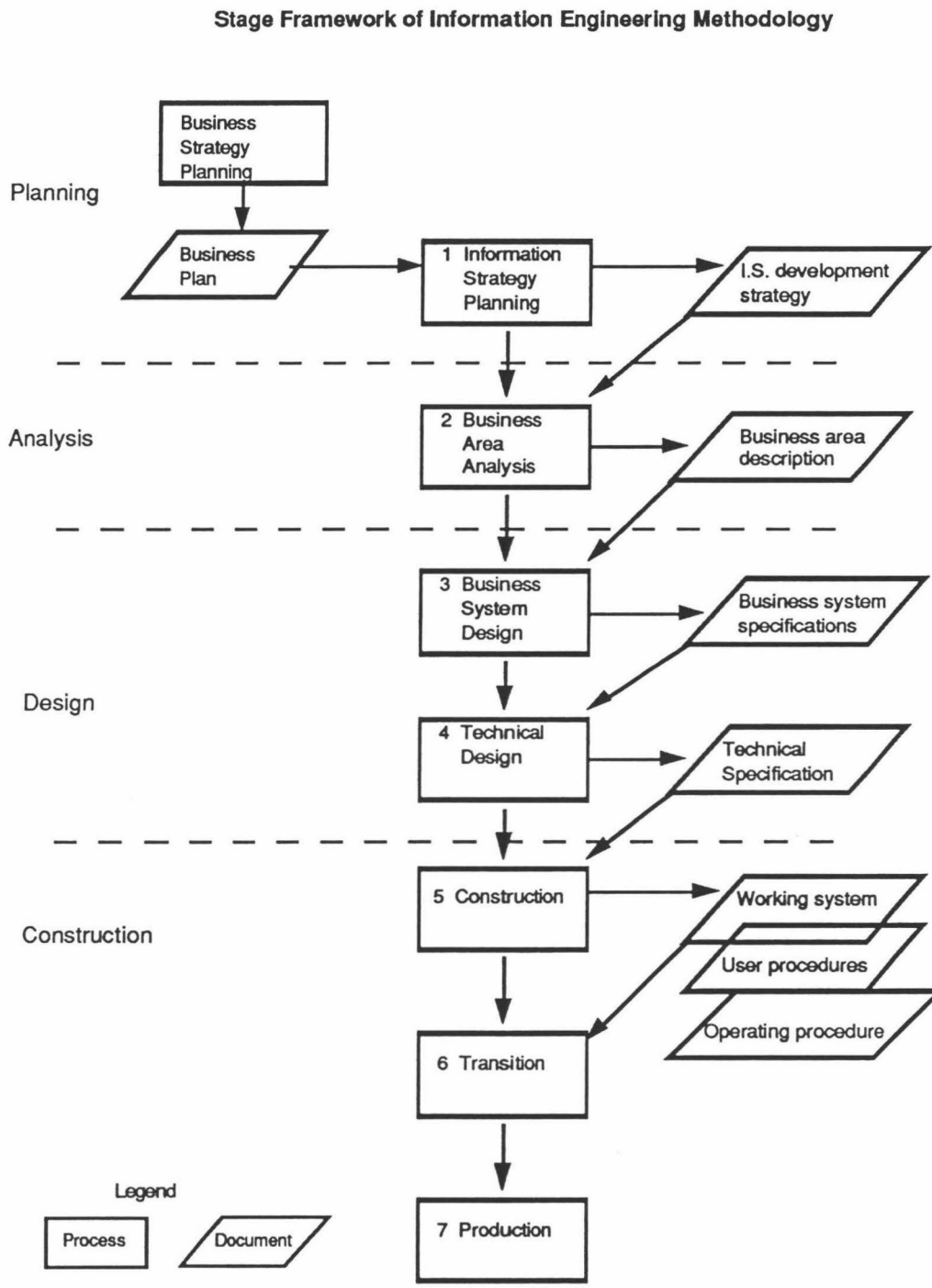


Fig. 1.1

1.2.3 Structured Systems Analysis and Design Methodology(SSADM)

SSADM[21] is a data-driven methodology developed originally by U.K. consultants, Learmonth and Burchett Management Systems and the U.K.Central Computing and Telecommunications Agency(CCTA). There are six phases in SSADM,

in which the first three phases are classified into systems analysis and the last three are systems design. They are :

(a) analysis of the current system,
(b) specification of the required system,
(c) user selection of Service Levels, including technical options,
(d) detailed data design,
(e) detailed procedure design,
(f) physical design control.

Data flow diagrams and entity models are needed to represent the static views of the system and a function/event matrix and an entity/event matrix are used to show the effects of time on the system.

1.2.4 Jackson Structured Design(JSD)

JSD[11] emphasises on the developing of maintainable software systems, and less on organisational need. Topics such as project selection, cost justification, requirements analysis, project management, user interface, procedure design or user participation are not addressed. JSD does not deal in detail with database design or file design. The major phases of JSD are :

(a) entity step action,
(b) entity structure step,
(c) initial model step,
(d) function step,
(e) system timing step,
(f) implementation step.

1.2.5 MERISE

MERISE[21] supports four stages of information system development. It combines an entity-relationship approach for data and a Petri-net based approach for processes.

1.2.6 Structured Systems Analysis(SSA)

SSA[21] was developed by Exxon in 1978, combining functional decomposition, data flow, relational data modelling and Jackson Structured Programming(JSP) techniques. Some information systems planning capability is also included.

1.2.7 Deficiency of mainstream methodologies

The 1980s have witnessed a growth in the number and variety of information systems methodologies. This increase in number of methodologies has caused much confusion. Many are the same(or very similar) and yet they have different 'brand names'. Some of them emphase in the techniques, the role of the computer, the documentation or the role of the people using the system. Some methodologies emphasize the importance of data and the development of a database. Some concentrate on analysis, others on design or implementation.

The classical waterfall software development life cycle, which is extensively used, is sometimes treated as a process in which work proceeds from one phase to another. It would be more difficult to return to the previous phase when the specification changes in comparison with OO development. Reasons why the traditional life cycle is inadequate for software development are :

(a) It assumes a relatively uniform progression of discrete steps, which includes little or no iteration,
(b) Due to the low cohesion and high coupling nature of program modules, it is difficult for the software to accomodate change which is a very desirable factor because each system is built from scratch and maintenance costs account for a large share of development cost,
(c) It does not accomodate the sort of evolutionary development made possible by rapid prototyping tools and 4GL,
(d) It does not allow future modes of software development like automatic code generation, module code transformation and 'knowledge-based' software development assistance,
(e) There is no emphasis on re-use of the software developed.

1.3 Current trends

In the early 1990's, there have been two new developments in the marketplace. One is Rapid Applications Development(RAD); the other is the IBM AD/Cycle applications development framework.

1.3.1 Rapid Application Development(RAD)

RAD[18] may be defined as the process of building and refining a working model or prototype of the final software system during the development process. The main purpose of prototyping is to refine functions, inputs and outputs during the design phase without having to wait for development to be completed. However, prototyping is not a

substitute for good analysis and design, but rather it is another way of producing results. If used properly, prototyping can be an effective tool and an aid in developing systems that allow closer user participation in the process, leading to information systems that meet the needs of the business.

Prototyping has been an informal methodology for quite some time. However, over the years, more experiences are gained in this area, and now it is possible to come up with some form of requirements or standards. The reason for prototyping is that the formal lifecycle is actually delaying the delivery of the final product. It is becoming the major cause of the application backlog. Moreover, the elapsed time between requirements and a delivered product erodes a customer's confidence. Perhaps, people are more impatient and pragmatic these days and would like to see some form of results earlier on. Gladden[18] suggests delivering any form of a prototype as quickly as possible. This approach is typified by Gilb[9] and Martin[17,18].

An approach to making prototyping successful was developed by Du Pont in 1985, called RIPP[3]. The approach was developed around the use of a CASE tool - CorVision from Cortex. A proposal and definition report was drafted between 10 to 15 days before proceeding to prototyping. The timebox is basically an iteration development process of the prototype limited to a maximum of 90 days before being evaluated again. Du Pont's first project using RIPP was completed in 5 man months compared with the 28 to 36 months using traditional approaches. This approach has saved them $2.3 million over 3 years, in 15 systems at 9 sites.

The RAD lifecycle has 4 phases[3] as applied in RIPP :

(a) Requirements Planning
(b) User Design,
(c) Rapid Construction,
(d) Transition.

During the first phase, developers create an outline model of the chosen area and define the scope of the planned system. Business executives, users, and developers take part in workshops(called the Joint Requirements Planning Workshop - JRP) that progress through a structured set of steps. All the results of the workshops are recorded using an integrated CASE(I-CASE) tool. The I-CASE tool is a repository for requirements and specifications. This stage usually takes one to three weeks.

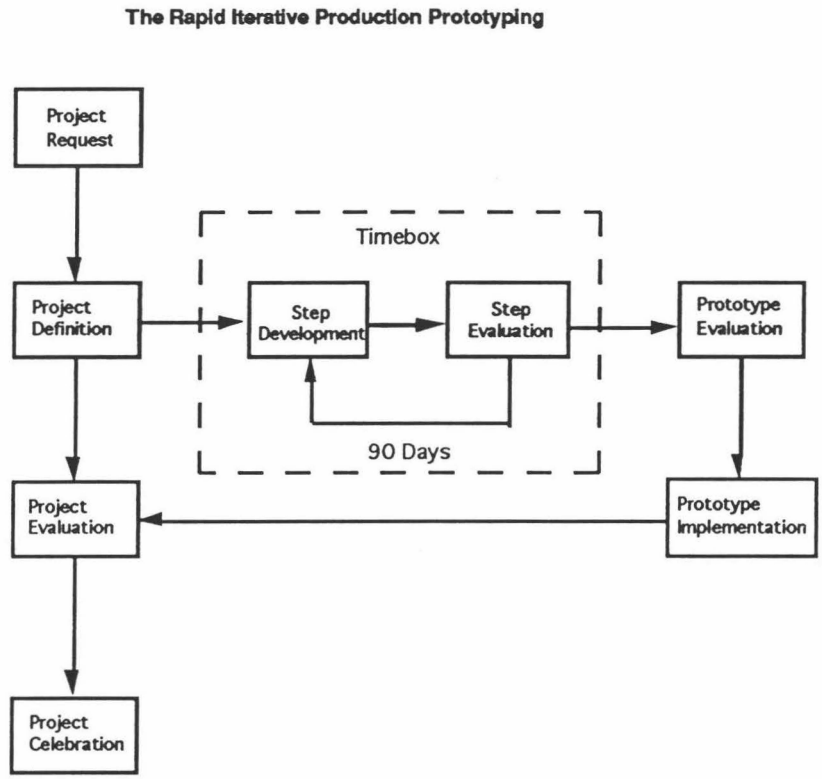


Fig. 1.2

The User Design stage requires that end-users participate strongly in the nontechnical design of the system under the guidance of an IS developer. User Design is done in a Joint Applications Design(JAD) workshop, which completes the detailed analysis of business activities and develops the outline design of the system. The information recorded in the I-CASE tool is used as input and is further refined. This stage usually lasts three to five weeks.

The third stage involves the design and implementation of the proposed system, which was outline in the previous stage. The software is constructed using an iterative technique. Finally this stage includes activities needed to prepare for cut over to production status. The I-CASE tool is used to generate the application code from database definitions.

When the system is cut over in the last stage, a variety of actions is needed, including comprehensive testing, end-user training, organisationanal changes and operation in parallel with the previous system until the new system settles in.

Prototyping approaches have the following advantages:

(a) improved developer user communications
(b) increased developer productivity
(c) working model versus a paper model
(d) model iterations

(e) user specification is changeable at any stage
(f) reduction in user training due to early participation
(g) production of error-free applications

However, the disadvantages are :

(a) configuration management and version control of prototypes is more difficult than with conventional development. Prototyping can result in many trial systems. It is possible to get versions mixed or to be unable to recover an earlier prototype version. Configuration management software can reduce this problem
(b) keeping documentation up to date may be difficult because of its rapidly changing and iterative nature
(c) maintaining discipline and objectives in the development team is difficult because it is possible to become distracted from the legitimate goals of the prototype due to the fluid nature and constant demands of prototyping
(d) Planning and allocating resource is difficult in an environment dealing with uncertainty and unknown
(e) ultimate testing may be neglected and left to the users.

Incidentially, a RAD approach has also been integrated into Information Engineering by Texas Instruments (James Martin Associates).

1.3.2 IBM AD/Cycle-Repository

In Sep 1989, IBM became a standard bearer for the computer-aided software engineering(CASE) industry by laying out its plan for the software development process. AD/Cycle-Repository[38,53,54] is an integrated framework intended for a CASE environment, and compatible with a range of development tools and techniques from many vendors. The goal is to vastly improve productivity in the applications development process. The only way to achieve this is to automate code generation through the use of models rather than conventional programming. Also it standardises repository storage of development objects. All CASE tools from other vendors, in order to link to AD/Cycle, must comply with certain IBM standards. However, no attempt has been made to create a standard in the methodologies themselves.

The primary benefit of the open repository-based environment is that users should be able to plug tools developed by CASE vendors complying with the repository standard into the environment and then use them together. CASE tools supporting various methodologies use the services of the Repository Manager to store user-defined application knowledge. The information contained in these models is stored in standard

format within the Repository Manager, from which it will be ultimately used to drive a code generator.

However, until now, it has not become popular due to a number of reasons. The MVS Repository Manager is not a stable product. Only a few CASE tools are compatible and it is difficult for other vendors to plug their CASE tools into the Repository. There is also problem with LAN configuration which is a important desired feature because today's CASE tool is geographically dispersed. Vendors with CASE tools running under MS-DOS and Unix have to rewrite them for OS/2EE for IBM PC and SAA compliance. One problem is that until now OS/2EE has not been popular.

While IBM is promoting integrated CASE in a mainframe environment, Digital Equipment Corp is following a more distributed path[53]. DEC's integrated CASE standard is known as ATIS(A Tool Integration Standard) and CDD/Repository in the VAX/VMS and Ultrix enviroments.

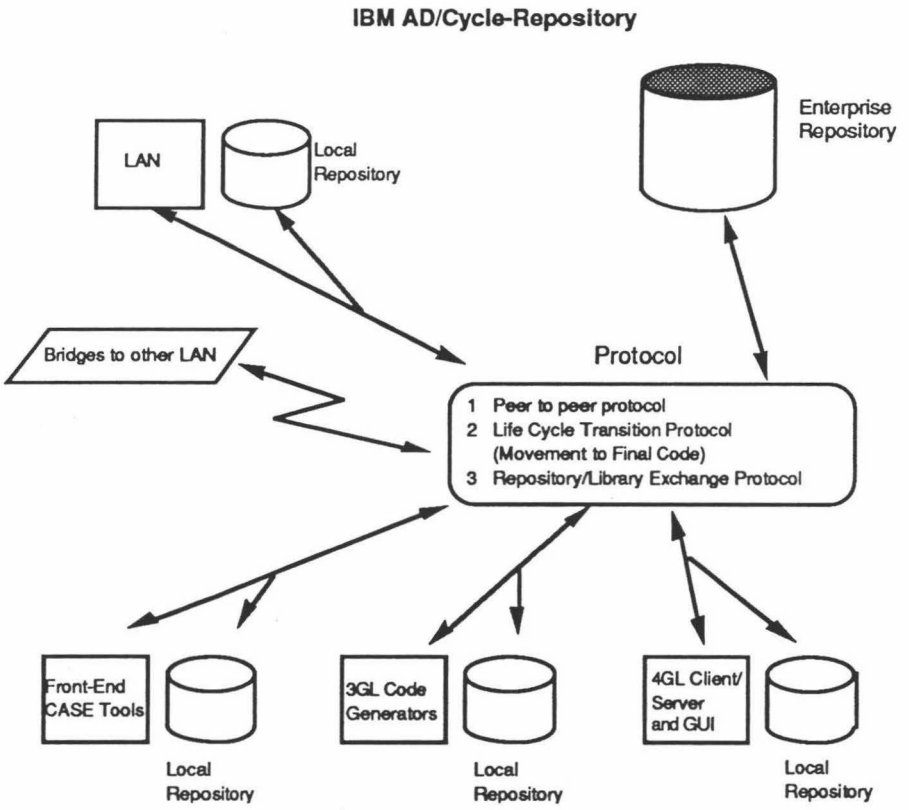


Fig. 1.3

1.4 Object-Oriented Methodologies

Recent suggestions[27] have been made that methods based on the paradigm of functions acting on data should be superseded by object-oriented approach. Object-

oriented methodology is defined as an application development strategy that models both requirements and software solutions as collections of objects that contain both data structure and behaviour.

However, many software organisations have developed standards and methods based on the functional approach and are understandably reluctant to embark on some design techniques that are still immature and unproven. Hence, any migration to new methods is likely to be a gradual one.

Current application of the OO paradigm has been limited to Design and Implementation due to the widespread use of C++ and Smalltalk in a small scale environment. Less has been done on the Analysis, although this is crucial for the construction of large and complex OO Information Systems.

The Object-Oriented development cycle is covered, in particular, by Booch[4], Budd[5], Henderson-Sellers[10,40], Korson[43], Jacobson[12,41], Bailin[31] and Coad & Yourdon[7,8].

1.4.1 Booch Methodology

Early versions of the methodology, proposed by Grady Booch were centered around Ada. In his most recent book, Booch introduces four models to capture OO semantics, which are then mappable to several target OO software environments.

1.4.2 Rolland & Brunet O* Model

This metholodogy[52], by the two authors at the University of Paris, concentrates on development for OODBMS, particularly the O2 system.

1.4.3 Coad & Yourdon OOA and OOD

This methodology[7,8] has been widely published through two books, one each on Analysis and Design, and a CASE tool has been developed.

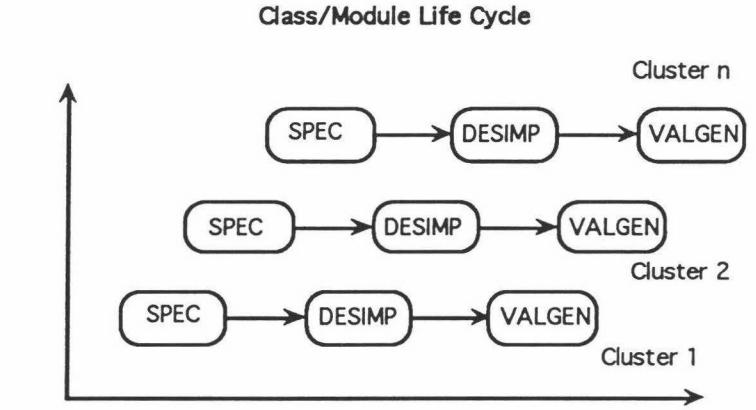
1.4.4 GE Labs Object Modelling Technique(OMT)

This technique[24] is developed by Rumbaugh, Blaha, Premerlani, Eddy and Lorensen at General Electric R&D Center, Schenectady, New York. Originally, this technique[34] was meant for use with relational database but has been modified to suit the object-oriented one.

1.4.5 Bertrand Meyer OO Methodology

Meyer's object-oriented methodology is centered around his OOPL, Eiffel. Not much is discussed about OOA. However, he claims that Eiffel language can both handle OOD and implementation[19]. The reason being the items of interest in each phase are

the same : objects. Objects and relationships between objects are identified in both the analysis and design phases. The cluster model has been proposed by Meyer as a life cycle for a tightly related group of classes, or cluster, in which three phases are identified.



Cluster model proposed by Meyer.
SPEC = specification; DESIMP = design+implementation; VALGEN=validation+ generalisation

Fig. 1.4

First, a specification is written by the systems designer(SPEC), then this is designed and implemented(DESIMP)(one process in a language like Eiffel) and finally it is validated and generalised(VALGEN). This life cycle occurs for different clusters of classes at different times. For example, a window cluster and a graphics cluster of classes could be specified, designed and implemented and then validated and generalised at different times. These phases are also iterative with refinements added.

1.4.6 Ivar Jacobson Object-Oriented Development

Ivar Jacobson come out with an early version of OO systems development in 1987. This technique originates from his work at Ericsson Telecom and since then has been used extensively within the whole Swedish telecommunication industry.

Basically, this paradigm describes a system as a set of properly interconnected blocks - each building block representing a packaged service of the system. A block may itself be made up of other, low-level blocks or by components. Components are standard modules which can be used for many different applications. The lowest-level blocks are made up of components only. Blocks as well as components are naturally implemented as classes using object-oriented programming. The designers are consequently provided with a set of components when building applications by means of blocks.

There is one interesting assumption made in this paradigm with regards to object concurrency. It is assumed that there is an infinite processor capacity, the execution speed is immensely high and endless storage volume. In this way, parallelism can be

disregarded and the course of events may be serialised. This assumption may not be adequate for the general case.

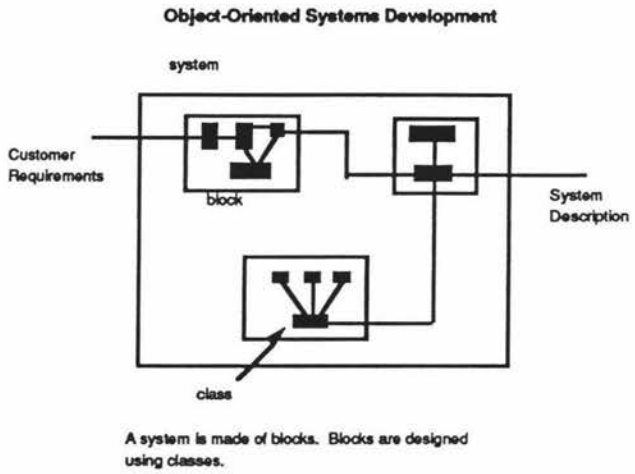


Fig 1.5

1.4.7 Henderson-Sellers OO Life Cycle

This methodology is developed at the University of New South Wales, Australia which describes the life cycle of OO systems devleopment. It focuses more on the front-end and high-level analysis. There are seven proposed steps to follow and earlier efforts made by Coad & Yourdon, Shlaer & Mellor, Bailin, and Wirfs-Brock are applied in these stages :

(a) Undertake object-oriented system requirements specification,
(b) Identify the objects and the services each can provide(interface),
(c) Establish interactions between objects in terms of services required and services rendered,
(d) Analyse stage merges into design stage : use of lower-level entity data flow diagrams/Information flow diagrams,
(e) Consider the bottom-up concerns and use of library classes,
(f) Introduce hierarchical inheritance relationships as required,
(g) Aggregate and/or generalise of classes.

The last step is illustrated using a fountain model rather than the traditional waterfall model.

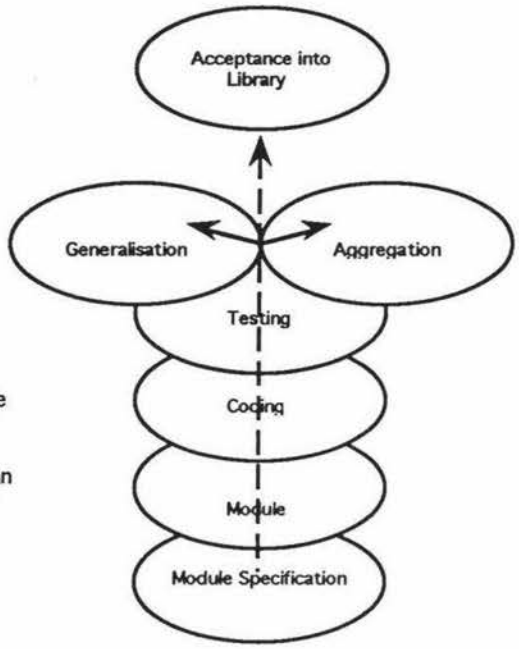


Fig. 1.6 Fountain Model

1.4.8 Summary of OO Methodologies

Booch and Henderson-Sellers have agreed that object-oriented design embodies an incremental, iterative process in between successive stages. Both Jacobson and Henderson-Sellers' ideas are particularly suitable for developing very large object-oriented software systems(>10 man-years). Another interesting point to note is that boundaries of analysis, design and implementation stages in object-oriented software development are blurred. Examples are Coad & Yourdon, who overlap between object-oriented analysis and design. Booch combines design with implementation.

Henderson-Sellers has identified the iterative process and comes out with the fountain model to replace the classical waterfall model. Development reaches a high level only to fall back to a previous level if so needed, to begin the climb once again. This is a better model of reality then the traditional waterfall model. Firstly, it provides a diagrammatic version of the stages present in an software life cycle and a clearer representation of the iteration and overlap made possible by object-oriented technology. Secondly, since the foundation of a successful software development is its requirements analysis and specifications, this stage has been placed at the base of the diagram. The fountain model can also be extended to the life cycle of a module, as outlined in Section 1.4.5.

Some of the advantages of object-oriented paradigm at the Analysis level[7,43] are:

(a) It can handle more complicated problem domains; emphasing more on the understanding of problem domains since it is based on objects, and not just functions or processes alone,

(b) It can improve interaction between analyst and client since it organises analysis and specification using the methods of organisation that pervade people's thinking,

(c) It can increase the internal consistency of analysis results. Object-oriented analysis introduced by several authors have consistent diagramming,

(d) The results obtained in Analysis can be reused on some similar projects.

Some of the advantages of object-oriented paradigm at the Design level[8,27,43] are :

(a) Object-oriented design is actually a continuation of the efforts made at the Analysis stage,

(b) Results and experiences gained during the Analysis stage can be reused,

(c) Object-oriented prototyping is used which increases productivity,

(d) Low life cycle cost,

(e) Modularity,

(f) Maintenability.

1.5 Conclusions

Most of the well-known OO methodologies have been given a brief introduction. While all of them offer approaches to extended data and behavioural modelling, none of them seem fully adequate to address the issues specifically related to OO database applications development. They have also not mentioned the guidelines and the steps involved in the prototyping process.