Object Orientation

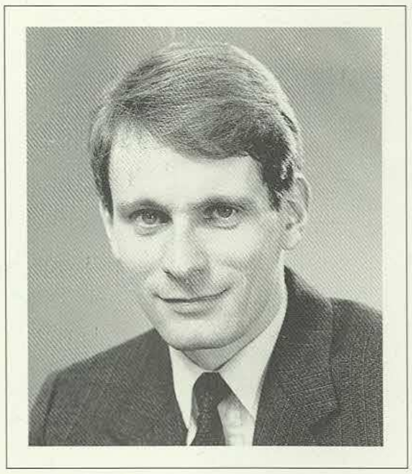
BUTLER COX FOUNDATION

A Paper by Richard Pawson
September 1991

BUTLER COX
FOUNDATION

Object Orientation

A Paper by Richard Pawson
September 1991



Richard Pawson is a senior consultant with Butler Cox, specialising in the tracking of emerging technologies and their potential application within large organisations. He has 14 years' experience in the development, launch, marketing and management of technology innovation. Since joining Butler Cox in 1989, he has carried out several consulting assignments in this area. He was joint author of Foundation Report 77, *Electronic Marketplaces*.

Object orientation is one of the hot topics in the IT industry today. Its proponents claim that it will revolutionise the way application systems are developed. Systems development managers have heard similar claims for other technologies, and are not yet convinced. In this paper, Richard Pawson explains what object orientation is in terms that are relevant to commercial systems developers and recommends a cautious but deliberate approach to adopting the technology.

BUTLER COX FOUNDATION

Published by Butler Cox plc
Butler Cox House
12 Bloomsbury Square
London WC1A 2LL
England

Availability of reports
Members of the Butler Cox Foundation receive copies of each report upon publication; additional copies and copies of earlier publications may be obtained by members from Butler Cox.

Photoset and printed in Great Britain by Flexiprint Ltd., Lancing, Sussex.

BUTLER COX
FOUNDATION

Object Orientation

A Paper by Richard Pawson
September 1991

Contents

BUTLER COX FOUNDATION

Object Orientation

Something radical is happening in systems development. It is called object orientation. Proponents of object orientation argue that it will be the Copernican Revolution of the systems world: Copernicus was the first to suggest that the world moved around the sun instead of *vice versa*, and his new model ultimately resulted in a new calendar, the dawn of the Age of Enlightenment, and the undoing of the authority of the Roman Church. Object orientation, they say, will result in a new age of information systems in which applications can be developed rapidly, and in which the systems department has a much reduced role. In at least one respect, the analogy is undeniable: the object-orientation issue is being addressed with almost religious fervour, both for and against.

Most systems professionals are underwhelmed. They have heard it all before. Relational database techniques, fourth-generation languages and CASE were all supposed to be revolutionary. Each has enjoyed success, but none has changed the fundamental way in which things are done. Some argue that object orientation is just a new name for techniques such as entity modelling and structured programming, which enlightened organisations have been using for years.

What is the truth about object orientation? Will it change the world of systems development, or will it fizzle out like a damp squib? Is it really new? Does it bring real benefits? At what cost? Most important, does the systems department need to do anything about it, or is the best strategy simply to wait and see?

Object orientation will eventually bring substantial benefits, but there are large hidden costs

We believe that object orientation is a real issue. We believe that within 10 years, systems development will look more like the object-oriented approach than like today's development approaches. We believe that object orientation could bring substantial benefits to most organisations, but that the costs of the changeover are large, and largely hidden. We believe that systems departments must start to address the issue now, and that there is a right and a wrong way to go about it.

In this paper, we attempt to clarify the issues surrounding object orientation. It would be nice to verify the claims by reference to user experience, but as yet, there is insufficient hard data available. That is changing, however. Within the next year, several large organisations will have completed medium- to large-scale object-oriented projects for commercial applications. It is therefore our intention to publish a full research report on this topic at a future date.

Object orientation is not new, but it is now gaining more attention

Object orientation is not new. Its roots lie in the programming language, Simula, developed in Norway in the 1960s for simulating complex systems such as manufacturing plants. In the early 1970s, researchers at Xerox Parc (Palo Alto Research Center) employed the same concepts to develop software that could be used by children. This work resulted in the programming language, SmallTalk, that is still one of the keystones of the object-oriented world.

For the next few years, object orientation remained little more than a curiosity, with just a handful of enthusiasts using it for real software development. In the mid-1980s, its popularity increased, particularly for computer-aided design (CAD), for developing graphical user interfaces, and for embedded applications such as control systems and avionics. Then, suddenly, in the last couple of years, the profile of object orientation has risen. Several respected software vendors are stating that it is the way forward, and some large user organisations are seriously assessing its suitability for their mainstream commercial systems development. What has caused this change?

Partly, the change has been driven by technology. Developments in high-performance workstations, in image and graphics technology, and in client-server architectures have all worked in favour of object orientation, for reasons that we shall examine later. A far more important driving force, however, is the growing dissatisfaction with conventional systems development.

Object orientation is being considered because traditional development is failing

The malaise has several symptoms. Software development accounts for an increasing proportion of total systems cost. Most systems departments have had to get used to a long applications backlog – one survey suggests that 60 per cent have a backlog of at least 12 months and 30 per cent of more than 24 months. Most worrying is the growing proportion of systems development effort that has to be devoted to software maintenance, reflecting the relatively poor productivity achieved in modifying existing systems. Peter Keen, director of the Washington-based International Center for Information Technology, has suggested that every dollar invested in software development commits four dollars to subsequent support and maintenance.

In addition, improvement in systems development has not kept pace with improvement in hardware. Processing power per dollar has been doubling every two years, and with the emergence of parallel architectures, that rate could increase. The same applies to memory and storage, and to a lesser extent, to display resolution and communications bandwidth. Although new languages and support tools and wider use of off-the-shelf packages have improved programmer productivity, this has not been at the rate needed to keep up with growing user demands and system complexity.

One cause of this is the fundamental mismatch between the internal structure of a piece of software and the 'natural' structure

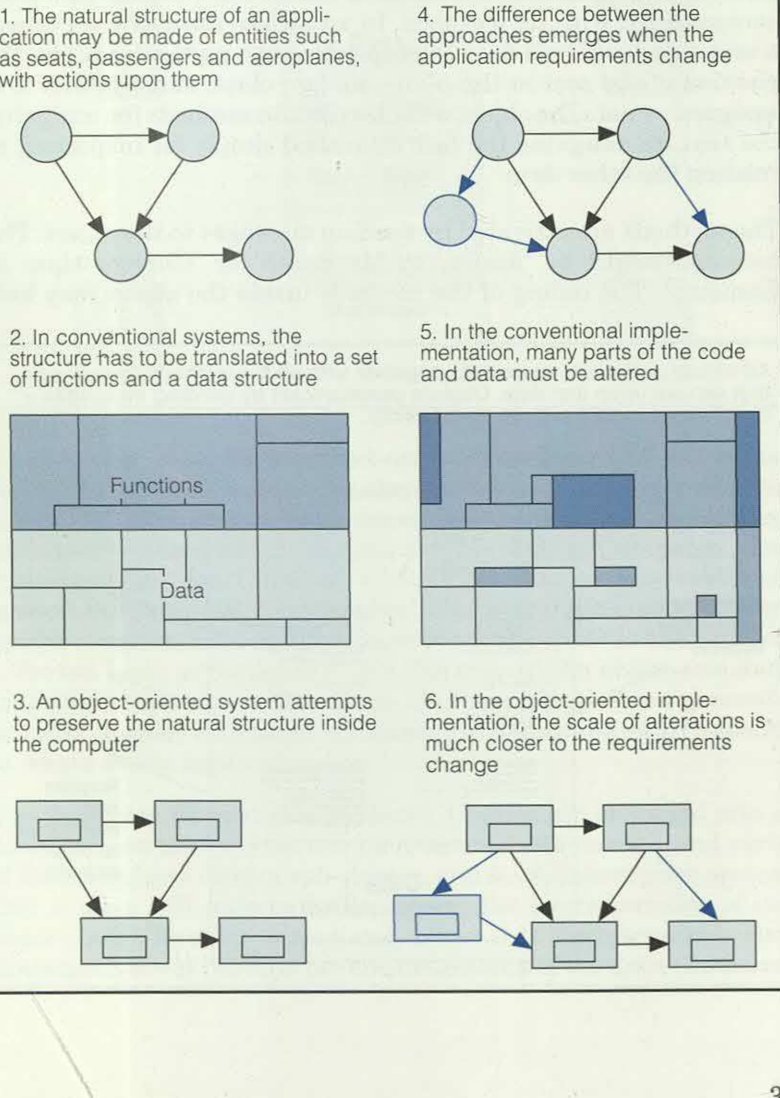
The problem arises because the internal structure of a system is different from the natural structure of an application

of the application that it is addressing. When processing power and memory were very restricted, it was important to represent software in the form most convenient for the computer. The process of converting the natural application structure into this form evolved into the processes that we now know as systems analysis, program design and coding. These processes require considerable time and effort. Moreover, a subsequent change to the application requirements – either because there has been a change to the business, or because there are new user requirements from the system – often means many small changes to the structure of the software.

For those organisations that are deploying networks of workstations and PCs, processing power and storage are in most cases no longer a critical resource – they represent a decreasing proportion of the total cost of any new system. It is no longer necessary to distort the natural structure of an application purely to comply with the constraints on processing power and storage. A radical approach to systems development is overdue.

Object orientation attempts to address these problems by aligning the internal structure of a piece of software more closely with the natural structure of the application. In technical terms, object

Object orientation attempts to align the internal structure of a piece of software more closely to the natural structure of the application.



1. The natural structure of an application may be made of entities such as seats, passengers and aeroplanes, with actions upon them

2. In conventional systems, the structure has to be translated into a set of functions and a data structure

3. An object-oriented system attempts to preserve the natural structure inside the computer

4. The difference between the approaches emerges when the application requirements change

5. In the conventional implementation, many parts of the code and data must be altered

6. In the object-oriented implementation, the scale of alterations is much closer to the requirements change

orientation achieves this by allowing a higher level of data abstraction. In plain English, if your business world consists of aeroplanes, suitcases, passengers, pilots and routes, you should be able to peer inside a developed application and see those same entities – not merely arrays, pointers, integers, strings and subroutines. Object orientation is not the first attempt at this idea, but it is the most thorough. We therefore now need to take a look at what object orientation actually is, and at what it is not.

Object orientation is defined by four core concepts

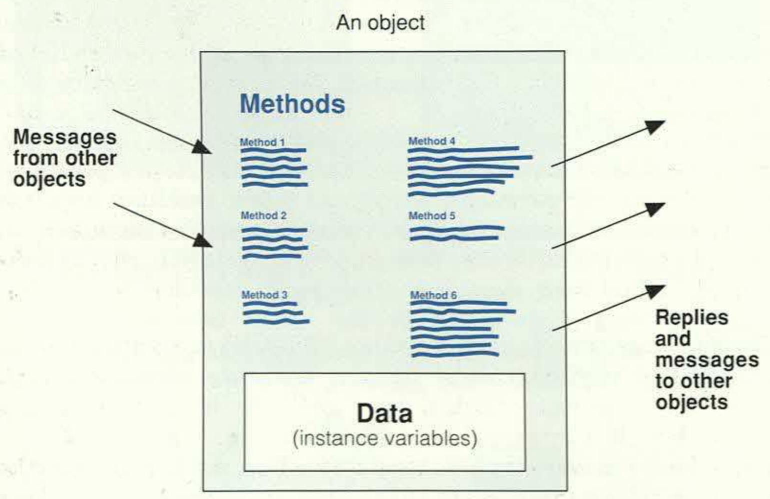
Object orientation is a distinct technology; it is not just an alternative name for ideas such as modular programming and entity modelling. Object orientation is defined by four core concepts: encapsulation, classification, inheritance and polymorphism. Around these four core concepts, several peripheral ideas have grown, such as multimedia data formats, but those peripheral ideas do not constitute object orientation. If a system does not explicitly support the four core concepts, it is not object-oriented.

An object-oriented piece of software is made up from objects. An object comprises some data in the form of simple variables, together with all the processes that can access that data. In object terminology, processes are called methods. Data is thus said to be *encapsulated* with its methods. In an airline reservation system, a seat object will hold data corresponding to a particular flight, the position of the seat in the plane, its fare class, and whether it is assigned or not. The object will also contain methods for assigning the seat, re-assigning the fare class, and simply for inspecting or reading the other data.

Data is encapsulated with its methods in an object

The methods are activated by sending messages to the object. The message might be 'Assign to Mr Smith' or 'Change class to Economy'. The coding of the methods inside the object may look

An object comprises some data, together with all the methods (processes) that can act upon that data. Objects communicate by sending messages – they cannot access remote data directly.



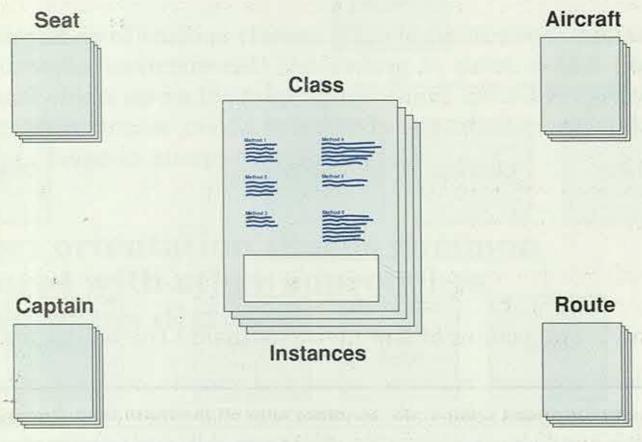
similar to the coding of a conventional programming language, although some object-oriented languages have quite different syntaxes and capabilities.

The benefit of encapsulation is that it offers total protection of the data. Because the only way that the data can be accessed is through the methods in the same object, it cannot be accessed or updated in a non-standard way. Provided that the message interface is preserved, internal alterations to an object cannot generate side effects.

Each object is an instance of a class

The second core concept is *classification*. This simply means that each object is an instance of a class. The reservation system will have a class called 'Aircraft', with one instance for every aircraft in the fleet, and a class called 'Captain', with instances for Captain Schmidt, Captain Jones and so forth. A class can therefore be thought of as a template that defines both the data (instance variables) held within objects of that class, and the methods that can operate on them. Like the record definition in a record-based database, it saves time and space by minimising unnecessary duplication.

Every object is an instance of a class, which can be thought of as a template that specifies the instance variables and the methods.



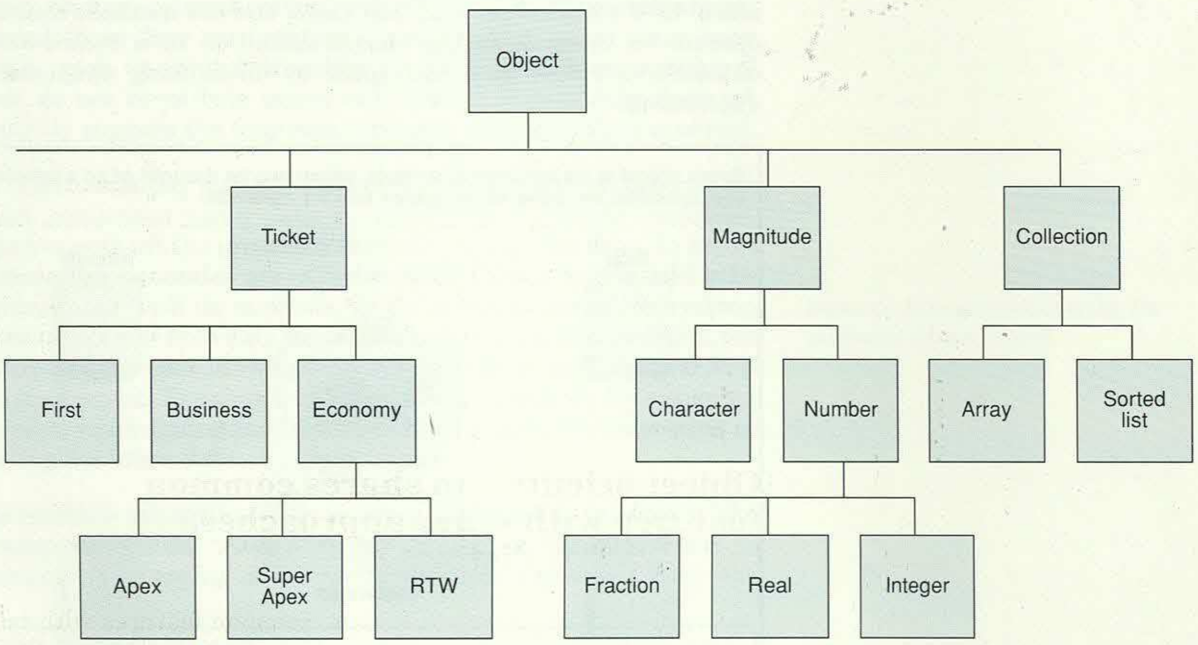
The starting point for an object-oriented development will be to define the classes needed for the application. In line with the concept of 'data abstraction', these classes can be as specific as necessary – there can be a class for 'Marketing Campaign', for 'Employee' and for 'Country'. Classes can also be low-level and general-purpose, such as 'Number', 'Date' and 'Character'. They can be complex, holding the names of other objects in the form of a 'Sorted List', a 'Dictionary' or a 'Document'. An object-oriented language may come with a large number of predefined general-purpose classes, which can substantially reduce the effort needed to create a new application.

Objects inherit the characteristics of their super-class

The third core concept is *inheritance*. Classes are arranged into a hierarchy, so that a class can have several sub-classes, and each of these can have its own sub-classes, and so on. Inheritance means that a class will automatically inherit the characteristics of its super-class (the class immediately above it in the hierarchy). The characteristics it inherits are the definitions of the data (instance variables) and all the methods. Having inherited them, a sub-class can modify them or add to them.

In the airline reservation system, there may be a class called 'Ticket', which will have instance variables corresponding to the route to be flown, and the total cost. It will have methods to specify the route segments, and to calculate the total fare. The 'Ticket' class may have sub-classes for First, Business and Economy, and the Economy class will have sub-classes for Apex, Super Apex and so on. In addition to the data and methods inherited from above, the Apex class will hold data and methods relating to minimum booking period, and restrictions on subsequent changes.

Classes are arranged into a hierarchy. Sub-classes inherit the instance variables and methods from above, but may modify or add to them. A sub-class is created when a specialised version of an existing object is needed. A typical system will comprise general-purpose classes (such as Number) and user-defined classes (such as Ticket).



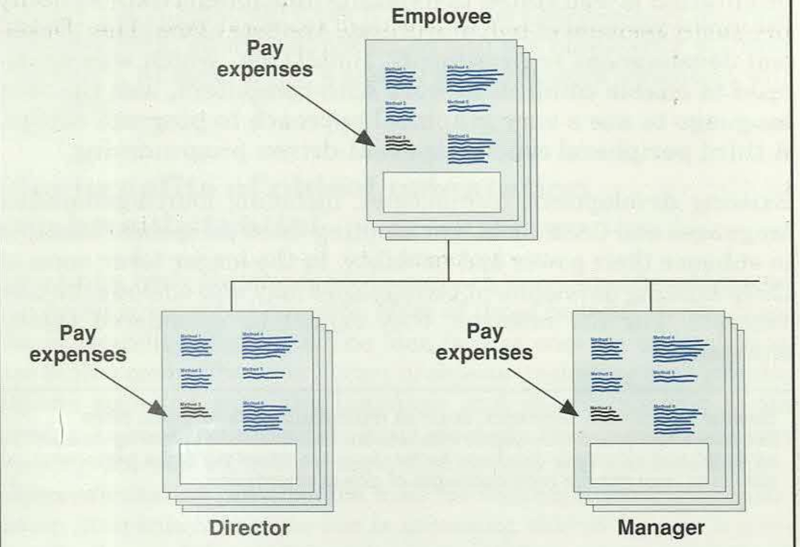
A sub-class is therefore a specialised case of its super-class. This is the most powerful feature of object orientation, because it mimics the way that humans tackle problems – starting with a general-purpose solution and adding specialised cases and exceptions. If a situation arises in an object-oriented system that does not quite fit the object specification, it is very simple to create a sub-class. The sub-classes are tightly managed in a hierarchy, and any subsequent changes to the super-class will still be inherited.

The fourth core concept is *polymorphism*. This simply means that different sub-classes can interpret the same message in different ways. The class 'Employee' may have sub-classes for 'Director', 'Manager' and 'Cabin Staff'. Each can accept the message 'Pay Expenses'. The coding of the method to implement that message will be different for each sub-class, however, reflecting the different auditing and tax requirements for each category.

Polymorphism means that different sub-classes interpret the same message in different ways

Polymorphism is attractive because it reduces the amount that a programmer needs to know and remember about the different

Polymorphism means that different sub-classes can implement the same message in different ways. The method that acts upon the message 'Pay Expenses' will be different for 'Director' and 'Manager', reflecting the different auditing and tax requirements.



characteristics of various classes. This is particularly apparent in a multimedia environment: the coding to print a text object, a graphical object and a bit-map-image object are all very different. All the programmer needs to know is that each responds to the 'Print' message in an appropriate way.

Object orientation shares common features with other approaches, but remains distinct

Object orientation clearly has some common features with more established software development tools and approaches. Structured programming, for example, attempts to achieve encapsulation through the concept of 'information hiding'. Most structured programming languages, however, are not able to enforce this discipline rigidly. (One exception is the military language, Ada.) Where it is not enforced, programmers will always be tempted to bypass the message interface and access the underlying data directly, often in non-standard ways.

Programming languages that allow the programmer to specify new data types may be said to have some form of classification, but these new data types cannot usually be used with the same flexibility as the standard data types that come with the language. In an object-oriented language, by contrast, there is no distinction between predefined general-purpose classes and user-generated classes.

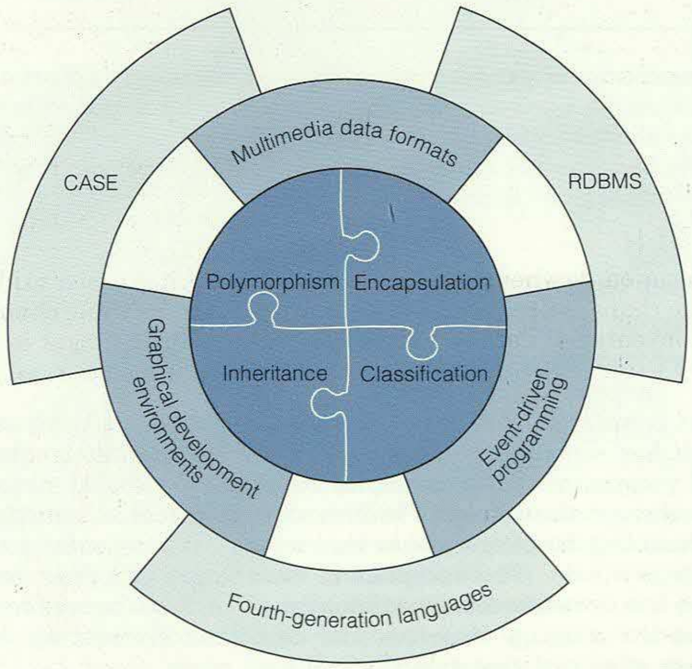
Inheritance has no direct equivalent in other development approaches

Inheritance has no direct equivalent in other modular programming languages and methods. In modular programming, the intention is to write general-purpose subroutines with encapsulated data structures. The problem is that when a situation does not quite fit, a new version of the subroutine has to be created, and these multiple versions rapidly become unmanageable.

Several concepts have become closely associated with object orientation. We call these 'peripheral concepts'. For example, object orientation is well suited to handling multimedia data formats, primarily because of polymorphism. Another example is graphical development environments. SmallTalk, which was developed to enable children to work with computers, was the first language to use a very graphical approach to program design. A third peripheral concept is event-driven programming.

Existing development technologies, including fourth-generation languages and CASE tools, are adopting these peripheral concepts to enhance their power and usability. In the longer term, some of these existing development technologies may also embody the core concepts. For the moment, they cannot be considered object-oriented.

Several 'peripheral' concepts, such as multimedia data formats, have become associated with object orientation. Established technologies such as CASE and relational database technology are adopting these peripheral concepts, but not the core concepts of object orientation.



Confusion is also being fostered by vendors whose products are threatened by object orientation – one example being relational database systems. Their arguments are beguiling. "Mathematically, there is a great deal of similarity between objects and relational structures." "We are in the process of migrating towards object orientation. We've already built in the multimedia capability." "If you stay with us, you don't need to worry about object orientation, because we'll get you there slowly and painlessly." These claims are reminiscent of the reaction of the hierarchical- and network-database vendors to the emergence of relational technology.

The transition from relational to object-oriented technology will be painful and expensive

We do not believe that relational database technology is dead – 10 years from now, systems development will still owe a good deal to the relational model. We do not believe, however, that you can migrate slowly and painlessly from relational database technology towards object orientation, as some of the relational vendors are suggesting. Even if the vendors successfully migrate their products, existing applications will not necessarily migrate with them. At some point, a painful and expensive transition will be needed.

The benefits of object orientation can be substantial

Adopting a truly object-oriented approach to systems development can bring substantial benefits. Most of these benefits derive from the re-usability of objects. The idea is that once an organisation has built a comprehensive library of objects (including both generic objects supplied with the language and objects specific to its needs), it should be possible to build most of a new application from existing objects. Modular programming had the same objective; object orientation provides the tools for making it work. Encapsulation guarantees that no-one is accessing shared data in a non-standard way. Inheritance makes it possible to create specialised versions of existing objects and to keep them within a tightly managed framework.

The re-usable nature of objects can reduce systems development time, and subsequent maintenance

Thus, once an object library has been established, object orientation can dramatically reduce both the cost and the timescale of new systems development. The effect on software maintenance is likely to be even more marked. Objects and the messages that pass between them more closely reflect the natural structure of the organisation, so when the requirements change, it is easier to identify the changes needed to the software. Moreover, while changes to a conventional data structure can imply many changes to the routines that access it (or *vice versa*), changes inside an object cannot generate side effects.

A third benefit is portability. Using an object-oriented language and development environment, it is genuinely possible to write applications with advanced graphical user interfaces and run them on other platforms. This means that an application can be developed on, say, a PC running Windows, and be run on a Macintosh, or on a Unix workstation running Motif. The disciplines of object orientation can also improve software quality, especially for complex systems. This is one reason that object orientation has proved popular for developing embedded systems, such as control systems for machinery.

The costs of object orientation are large and often hidden

Weighed against these substantial benefits are some substantial costs and disadvantages. The external investment required is actually minimal – object-oriented languages and development environments start from a few hundred dollars and will run on any reasonably high-performance PC. The real costs of moving over to object orientation are not initially apparent.

The first hidden cost is re-education. Learning object-oriented development is not simply like learning a new programming language. It requires the analysts and programmers to learn to look at an application and 'see' objects, rather than functions and data structures. Experience to date suggests that not all development staff will be able to make the transition effectively.

Designing the initial class hierarchy is a major hidden cost

Second, there is the cost of establishing an initial class hierarchy. Getting the higher levels of the class hierarchy right is crucial to achieving the full set of benefits, yet the process of identifying these classes is far from intuitive, and so far, there are few well tested tools and formal analysis methods available.

Third, object-oriented systems may not run as efficiently as conventional code, but this is increasingly offset by the falling cost of processing power. Some applications, however, especially those with a complex structure, can actually run faster with object orientation.

Debugging object-oriented systems is currently more difficult than debugging conventional systems

Although object-oriented code is easy to write, initial debugging can be hard work, because there are no real barriers between the application being developed, the language and the development environment. Initiating a trace will reveal all the messages passing between internal objects as well as those that the programmer has actively invoked. This situation will doubtless improve as better tools and techniques evolve. This is therefore an appropriate point at which to examine the current state of object-oriented technology.

Object orientation is still immature, but the range of tools is growing

Although object orientation has been practised since the early 1970s, the range of software tools to support it is still limited. The range of formal analysis and design methods is more limited still. Renewed interest in object orientation means that the range of tools and techniques will grow very rapidly in the next few years. Quality will inevitably be mixed.

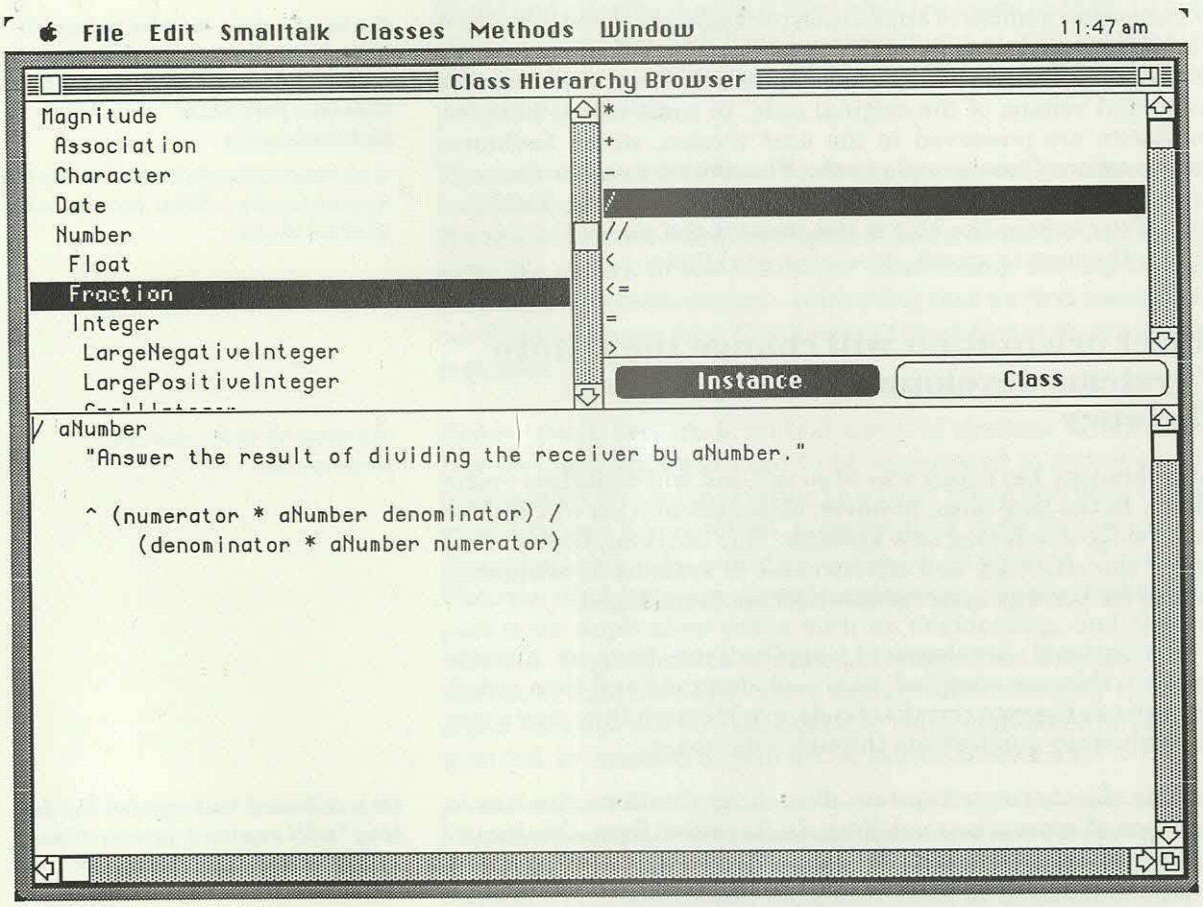
Object-oriented programming languages allow instance variables and methods for objects to be specified

The most important software tool in object orientation is the language. Strictly speaking, the function of the language is to specify the instance variables and methods of each object class, and then, at run-time, to manage the flow of messages between them. In practice, most object-oriented languages also incorporate a sophisticated development environment that allows the programmer, among other things, to browse and inspect the class hierarchies.

SmallTalk was the first comprehensive object-oriented language, and in many respects, is still the purest. Several commercial versions are available. SmallTalk is a powerful language, and notwithstanding the difficulties of learning the object-oriented approach, is relatively easy to use. Although conceived for research purposes, SmallTalk is very suitable for developing technical applications, and is now being promoted as a viable language for commercial systems development.

The most popular object-oriented language is C++, which is a set of object-oriented extensions to the C language. C++ is a hybrid

SmallTalk – still the purest of the object-oriented languages – includes a graphical development environment for browsing through object classes. This screen display shows a few of the standard classes that come with the SmallTalk/V system from Digitalk.



```
/ aNumber
    "Answer the result of dividing the receiver by aNumber."

    ^ (numerator * aNumber denominator) /
        (denominator * aNumber numerator)
```

language that allows conventional programming to be mixed with objects. This suggests the possibility of a gradual migration from conventional coding (assuming the conventional code is written in C) to object orientation. With hybrid systems like this, there is a greater risk of incurring the costs of object orientation without the benefits. However, many large software vendors have chosen the C++ route, so we can expect to see the range of tools supporting this language to increase.

Object-oriented databases provide better support for multi-user development and operation

Object-oriented languages provide little support for the sharing of objects between several users, either during development or at run-time. This is where object-oriented databases come in. (Delegates visited two object-oriented database vendors on the 1990 Foundation Study Tour and one on the 1991 tour.) Object-oriented databases provide the same kind of locking, mirroring and security features for objects that conventional databases do for data. It is not necessary to purchase such a database to write object-oriented applications, but it is desirable if the application is large, or if it services several users.

Several vendors now market graphical development environments that use the object-oriented approach to build applications starting from the user interface. The most sophisticated of these is NeXTStep, which comes as standard with the NeXT workstation, and has been licensed to IBM to run on its RS 6000 series.

An increasing number of application packages are being developed in an object-oriented fashion. This may have benefited the vendor, but in most cases, it makes no difference to the user, who sees only a compiled version of the original code. In some cases, however, the objects are preserved in the user version, which facilitates customisation. One example is the ViewStar document management system presented during the Foundation's 1991 Technical Study Tour, where the object structure of the package makes it easy for the user to specify document workflows.

An increasing number of application packages are developed using object orientation, and this can facilitate customisation

Object orientation will change the nature of systems development, not just its efficiency

The technology has a long way to go yet, and will doubtless evolve in form. In the long term, however, object orientation will become the norm for developing new systems. This holds implications not just for the efficiency and effectiveness of systems development, but also for the way systems development is managed.

In conventional development, applications have a discrete identity – they are specified, analysed, designed and then coded. They exist as discrete chunks of code, even though they may share data with other applications through a database.

In a fully object-oriented system, discrete applications give way to the notion of 'enterprise modelling'. In its purest form, this means building a complete simulation of the way that the business enterprise functions, using objects. (Remember that object orientation has its roots in simulation.) Monolithic applications are replaced by user-interface objects that pass messages to this simulation, to read and write information, and where appropriate, to change the operation of the simulation. (Object orientation is well suited to providing the IT support for redesigned business processes, which we discussed in Report 79, *The Role of Information Technology in Transforming the Business*.) The investment in building and maintaining this enterprise model may be very large, but the cost and time needed to build 'applications' from this model are very small.

Object-based 'enterprise modelling' will replace conventional applications development

Many organisations will not build a complete and comprehensive enterprise model, but their systems development will polarise into building core objects and application interfaces. Core-object programmers will need to be both first-rate conceptual thinkers and creative coders, which implies a high level of investment in education and training. Some organisations will buy in this function.

Application interfaces, however, will increasingly be built either by users or by a new breed of analyst/programmers working very closely with users, and more often working individually than in teams. The primary skill needed by the analyst/programmer will be the ability to understand and articulate the needs of the user.

Unlike today's arrangement, though, where the systems analyst passes on the requirement to another specialist for systems design, in the object-oriented world, he will build the system. Rapid prototyping will become the norm – one analyst/programmer working on-site with users for a few days, building a user interface to the model and altering it until it does what the users require. In other words, the big payback from the investment in building the right core objects is a substantial reduction in the cost of building individual applications.

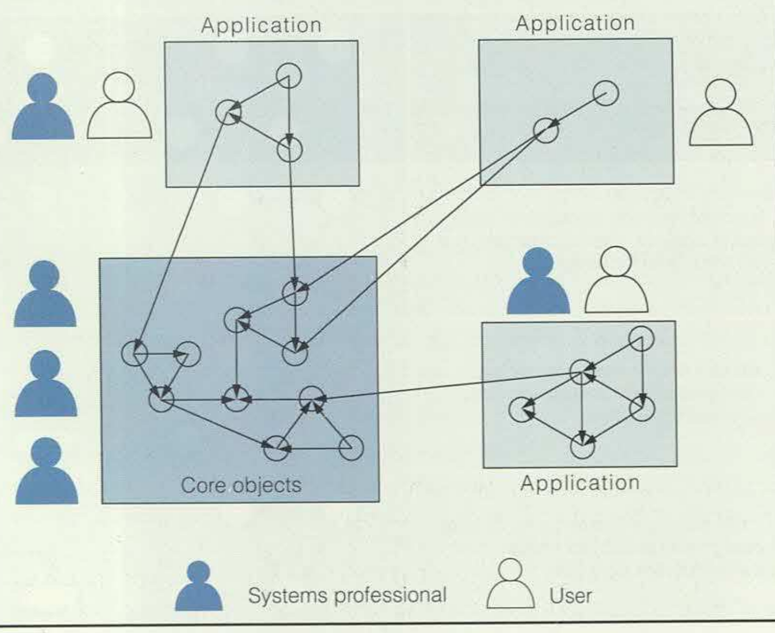
Object-oriented development is a good match with a client-server architecture

This polarisation between the enterprise model, or core objects, and the applications that interface to them suggests a good match between object-oriented development and a client-server architecture. We expect to see a stronger relationship develop between these two new technologies – something that we first identified in our Position Paper, *New Directions in Client-Server Systems: Findings from the 1991 Study Tour*.

Re-use must be actively encouraged

Re-use must become a central tenet of systems development. Development staff will need to be encouraged to re-use existing objects in preference to creating new ones. When new object classes or sub-classes are required, programmers must be rewarded for designing them to be as widely applicable as possible. Commercial libraries of objects are already appearing, usually dedicated to particular application areas such as engineering, and this will increase. Organisations will need to ensure that they are making full use of such libraries, where appropriate. Internally developed object libraries will hold considerable value and should be carefully guarded, or licensed to gain a new source of revenue.

Systems development will be polarised. A dedicated team will develop and maintain the core objects, possibly in the form of a complete enterprise model. Individual applications will tap into this model and require little additional code development. Those applications will be developed either by end users, or by users working in partnership with an analyst/programmer.



There is one right way and several wrong ways to approach object orientation

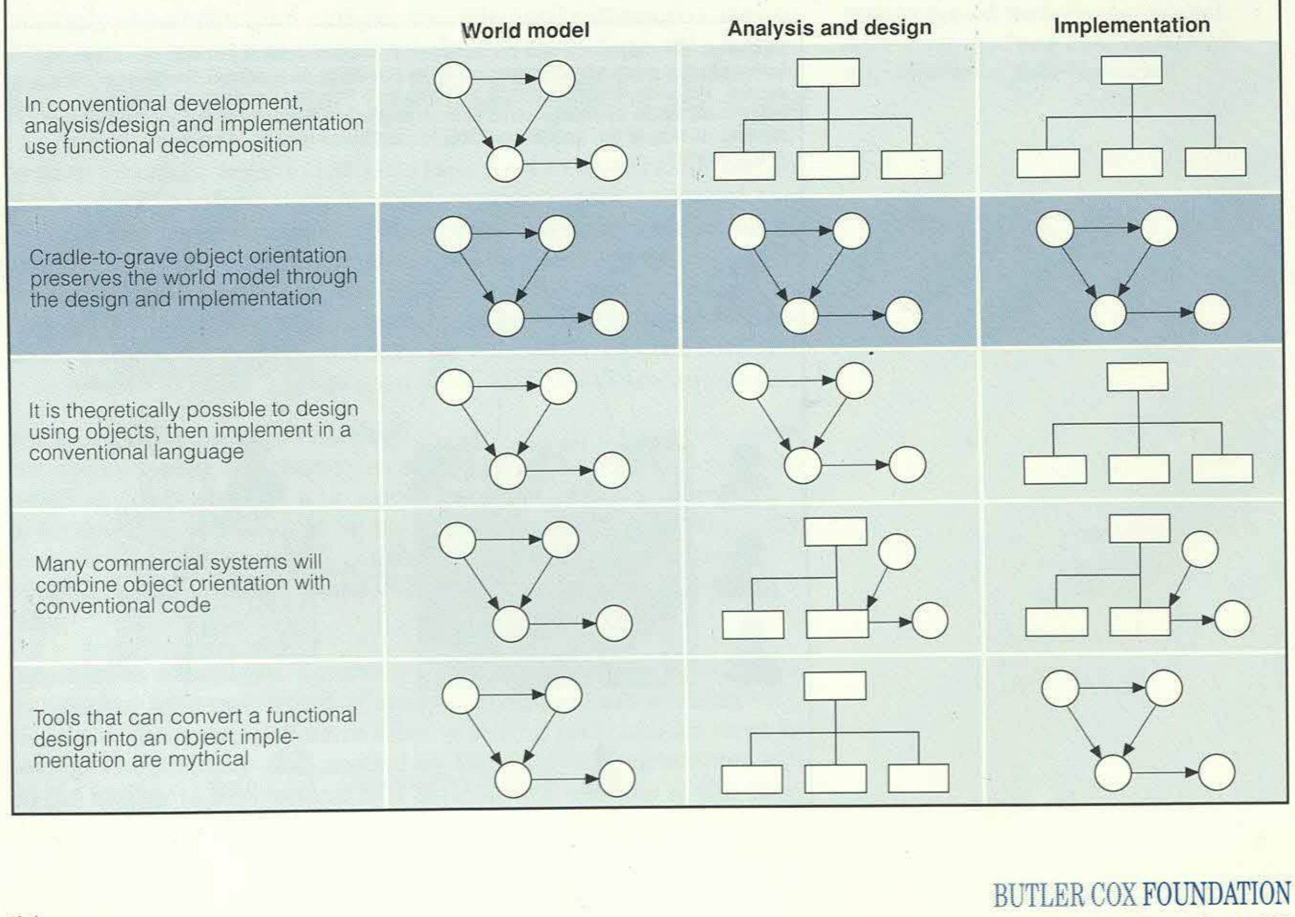
In theory, there are several strategies with regard to object orientation. The first is to do nothing – continue with the traditional approach of functional decomposition at the analysis/design stage, and a translation of this design into program code. The second option is cradle-to-grave object orientation – analyse the application into an object structure and code it with an object-oriented language. This is the purist approach.

The other options are all hybrid. In theory, it is possible to undertake the analysis/design using an object approach, and to implement the objects using any conventional language such as Cobol. It is also possible to build an application that is partly conventional and partly object-based – perhaps with one of the hybrid languages such as C++, and others that are imminent.

Some vendors are even suggesting that organisations can continue to undertake analysis/design in a conventional manner, and that their software packages will somehow transform this into object-oriented form. We do not believe it.

In its current state of maturity, it is not realistic to suggest that systems departments should drop what they are doing now and move across to object orientation, lock, stock and barrel. Even if an organisation wanted to build a complete object-oriented enterprise model, it would probably not be able to find appropriate tools, methods, or even professional guidance.

The cradle-to-grave approach to object orientation is the best.



In conventional development, analysis/design and implementation use functional decomposition

Cradle-to-grave object orientation preserves the world model through the design and implementation

It is theoretically possible to design using objects, then implement in a conventional language

Many commercial systems will combine object orientation with conventional code

Tools that can convert a functional design into an object implementation are mythical

The right approach is to start using object orientation, preferably on a cradle-to-grave basis, for a few carefully selected applications. Choose new applications that are small and relatively self-contained, but that can interface to conventional applications and databases. The more complex the structure of the application, the more the benefits of object orientation will show. Above all, pick applications where there is a strong likelihood that user requirements will change.

By testing object orientation on a small-scale, but real, application, you will be able to evaluate the costs, benefits and implications in the context of your own organisation. You will be better placed for a full-scale move into object orientation when the tools and approaches are sufficiently mature. You will not get to this position if you confine object orientation to the IT research department.

We recommend this purist approach for initial forays into object orientation, because in our view, that is the only way that the implications for the organisation will become clear. It is not realistic, however, to suggest rewriting existing large-scale commercial systems using pure object-oriented techniques. Most organisations will have to compromise, and compromise is a difficult strategy to get right.

Accordingly, the Butler Cox Foundation has commissioned a research project into the practical aspects of object orientation for commercial systems development. We shall be studying organisations that have undertaken substantial object-oriented projects to ascertain the degree to which the technology has and has not lived up to its promises. We shall be examining some of the tools, together with the design and analysis methods. From that information, we shall be drawing up some practical guidelines for adopting object orientation, and for combining it with existing systems. These findings will be published in a future Foundation Research Report.

In the meantime, senior managers must gain an understanding of what the real issues are. Object orientation is here to stay.

BUTLER COX
FOUNDATION

The Butler Cox Foundation

The Butler Cox Foundation is a service for senior managers responsible for information management in major enterprises. It provides insight and guidance to help them to manage information systems and technology more effectively for the benefit of their organisations.

The Foundation carries out a programme of syndicated research that focuses on the business implications of information systems, and on the management of the information systems function, rather than on the technology itself. It distributes a range of publications to its members that includes research reports, management summaries, directors' briefings and position papers. It also arranges events at which members can meet and exchange views, such as conferences, management briefings, research reviews, study tours and specialist forums.

Membership of the Foundation
The Foundation is the world's leading programme of its type. The majority of subscribers are large organisations seeking to exploit to the full the most recent developments in information technology. The membership is international, with more than 450 organisations from over 20 countries, drawn from all sectors of commerce, industry and government. This gives the Foundation a unique capability to identify and communicate 'best practice' between industry sectors, between countries, and between information technology suppliers and users.

Benefits of membership
The list of members establishes the Foundation as the largest and most prestigious 'club' for systems managers anywhere in the world. Members have commented on the following benefits:

— The publications are terse, thought-provoking, informative and easy to read. They deliver a lot of messages in a minimum of precious reading time.

— The events combine access to the world's leading thinkers and practitioners with the opportunity to meet and exchange views with professional counterparts from different industries and countries.

— The Foundation represents a network of systems practitioners, with the power to connect individuals with common concerns.

Combined with the manager's own creativity and business knowledge, membership of the Foundation contributes to managerial success.

Butler Cox

The Butler Cox Foundation is one of the services provided by CSC Index. CSC Index is an international consulting group specialising in information technology, organisational development and business reengineering. Its services include management consulting, applied research and education.

Butler Cox plc
Butler Cox House, 12 Bloomsbury Square,
London WC1A 2LL, England
☎ (071) 831 0101, Telex 8813717 BUTCOX G
Fax (071) 831 6250

Belgium and the Netherlands
Butler Cox Benelux bv
Prins Hendriklaan 52
1075 BE Amsterdam, The Netherlands
☎ (020) 6 75 51 11, Fax (020) 6 75 53 31

France
Butler Cox SARL
Tour Akzo, 164 Rue Ambroise Croizat,
93204 St Denis-Cédex 1, France
☎ (1) 48.20.61.64, Télécopieur (1) 48.20.72.58

Germany, Austria and Switzerland
Butler Cox GmbH
Richard-Wagner-Str. 13, 8000 München 2, Germany
☎ (089) 5 23 40 01, Fax (089) 5 23 35 15

Australia, New Zealand and South-east Asia
Mr J Cooper
Butler Cox Foundation
Level 10, 70 Pitt Street, Sydney, NSW 2000, Australia
☎ (02) 223 6922, Fax (02) 223 6997

Finland
TT-Innovation Oy
Sinikalliontie 5, 02630 Espoo, Finland
☎ (90) 358 0502 731, Fax (90) 358 05022 682

Ireland
SD Consulting
8 Clanwilliam Square, Dublin 2, Ireland
☎ (01) 764701, Fax (01) 767945

Italy
RSO SpA
Via Leopardi 1, 20123 Milano, Italy
☎ (02) 720 00 583, Fax (02) 86 45 07 20

Scandinavia
Butler Cox Foundation Scandinavia AB
Jungfrudansen 21, Box 4040, 171 04 Solna, Sweden
☎ (08) 705 83 60, Fax (08) 730 15 67

Spain and Portugal
T Network SA
Núñez Morgado 3-6°b, 28036 Madrid, Spain
☎ (91) 733 9866, Fax (91) 733 9910