"Can You Have It All?": Managing The Time And Budget Against Quality Issue in A Dynamic Business Object Architecture Development

KSY HUNG[1] , AJH SIMONS[2] and **A ROSE[3]**
[1,2] Department of Computer Science, The University of Sheffield
Regent Court, 211 Portobello Street, Sheffield, S1 4DP, UK
e-mail: [1] k.hung@dcs.shef.ac.uk ; [2] a.simons@dcs.shef.ac.uk
[3] CAD Consultants Ltd.
797 London Road, Thornton Heath, Surrey, CR7 6XA, UK
e-mail: [3] tony.rose@btrinc.com

ABSTRACT

It has been widely observed in the information technology (IT) communities that IT developers are coming increasingly under more pressure than ever in juggling between software quality and timely delivery in a tight budget. Developers are torn between the dilemma of either delivering quality software at a price of longer development time and higher cost or delivering software in a timely fashion neglecting quality. Many attempts have been made to tackle the challenge of: "Can-We-Have-It-All?". This paper recommends an approach to manage time and budget against quality and aims at achieving this tripartite objective. The paper covers the development of a Dynamic Business Object Architecture (DBOA) and its implementation through an insurance project case study. The structure and approach of the DBOA are explained through the development process and the case study is presented to demonstrate the initial result of this approach. Some insights resulting from applying the above techniques are also discussed.

1. Introduction

In the competitive business environment, software technology now plays a crucial role in providing increased lead time (shorter time to market). Electronic Information Systems (EIS) have become an integral part in most organisations [18]. It is a common goal amongst business organisations to improve the quality of, and gain added value from, their information processing application for a minimum investment in time and cost. In respect to the software quality issue, Object-Orientation (OO) is currently regarded as a better alternative for developing quality software than conventional structured methods [5]. OO promises, with its focus on encapsulated components, better maintainability, extensibility, scalability, portability and reusability. Furthermore, Business object technology has been developed to capture and define a model of the user's business and its information processing requirements [9,12]. A Business object is a coarse-grained object abstraction that encapsulates a typical, generic business task, adapted for a particular

business domain [6,1]. Business objects are incorporated in a Business Object Architecture (BOA) [2], which is a re-configurable framework for handling the communications amongst business objects within a business domain. However, those techniques only swing the pendulum towards improving software quality regardless of the cost and time.

In regards to time and budget, Rapid Application Development (RAD) [11] is increasingly being adopted as the best way to deliver systems quickly and at low cost in many situations. The downside of RAD is that the pendulum swings towards shorter development time and reduced costs, at the expense of quality. As a result, software may be bug-ridden, or poorly structured which makes it difficult to maintain. The Dynamic Systems Development Method (DSDM) [3,15] has been adopted to impose a software life cycle on RAD and improve project management. However, this does not fully address the core issue of software quality, which is more affected by the development techniques used to capture and model business information.

This paper presents an approach that combines BOA and DSDM to provide a "Dynamic Business Object Architecture (DBOA)" [7] to develop quality software applications within practical time scales and for minimum costs. Projects using BOA techniques are implemented within the DSDM life-cycle environment. A credit insurance project case study was carried out using the DBOA approach to look at areas involving time and budget against quality. The structure of the rest of the paper is organised as follows. Section 2 outlines how Business Objects provide a mechanism to assist IT developers to understand the business domain better. Section 3 describes the adoption of a Business Object Architecture (BOA) to manage complexity encountered when reusing business-related software components. Section 4 describes the development of a Business Object Repository / Reuse Library to enhance the reusability of Business Objects. Section 5 addresses the necessity to involve business end-users to ensure the effectiveness and usefulness of the BOA to the business. Section 6 presents a DBOA framework applying the BOA coupled with DSDM life-cycle implemented through an insurance project case study. Section 7 evaluates the outcome from the case study. Section 8 justifies the balance between time and budget against quality by using a 'S.M.A.R.T.' evaluation criteria framework to evaluate the DBOA in terms of its 'Scalability', 'Measurability', 'Achitevability', 'Reusability' and 'Time-manageability'. And finally, section 9 concludes this paper and outlines further research work.

2. Business Objects And Business Object Architecture

2.1 What is a Business Object ?

Object Management Group (OMG)'s definition of Business Objects as: *"… a representation of a thing active in the business domain, including at least its business name and definition, attributes, behaviours, relationships, rules, policies, and constraints. A business object may represent, for example, a person, place, event, business process, or concept. Typical examples of business objects are: employee, product, invoice and payment …*" [16]. Business objects can be viewed as Modelling Objects, used in the design process and as objects in the information

system as illustrated in Figure 1. IT developers extract that 'modelling object' from the business and transform it into software components structured in Object-Oriented (OO) style, so that the modelled information can directly reflect the shape of the business model.

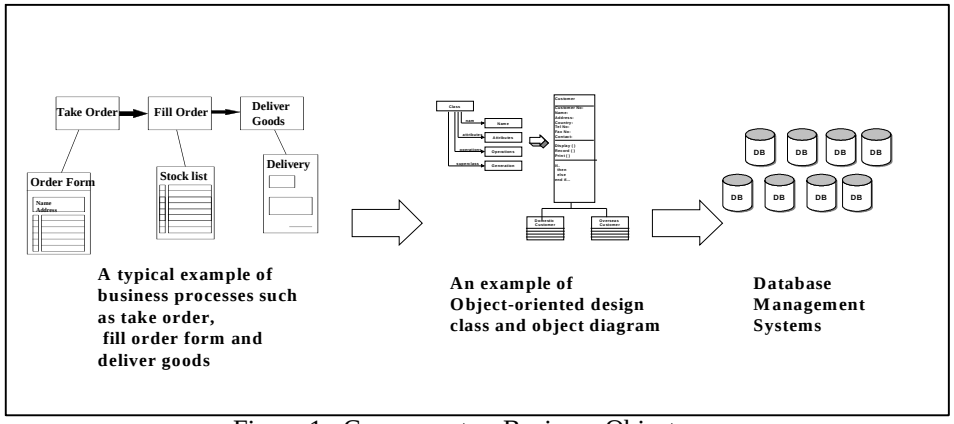

Figure 1 : Components a Business Object

2.2 Development of a Business Object

Standards are currently being developed by the OMG Business Object Domain Task Force through a request for proposals (RFP) from the software developers and academic researchers. The RFP is still under review pending for agreed common standard. At the moment there is no standard approach or method for creating business objects. To meet this need, we have combined Jacobson's Use Case Engineering (UCE) [9,10] and Ramackers' Domain Business Object modelling approach [13], as shown in Figure 2.

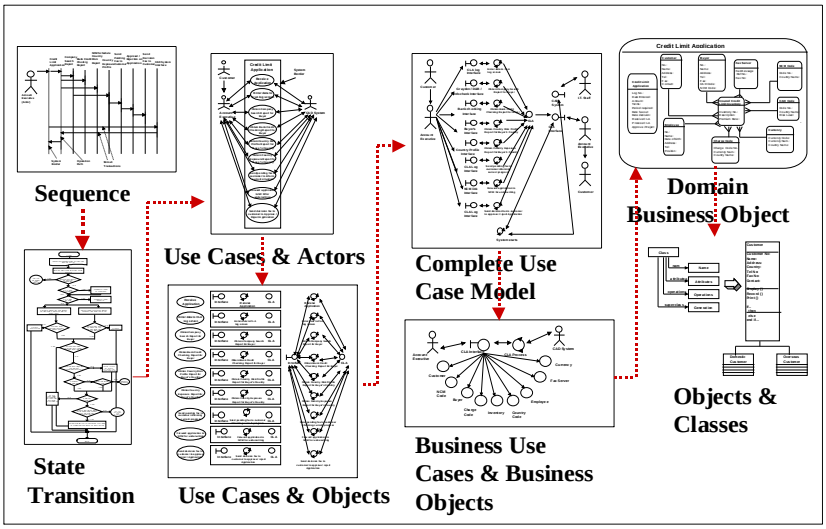

Figure 2 : Development of a Business Object

We use the sequence diagram, which gives an overview of the business processes; and the state transition diagram to describe the business processes step-by-step. The use cases and actors bring out the names of the tasks and how they are performed. These use case and object diagrams convert data into 'entity objects', processes into 'control objects' and actors into 'interface objects', these three different kinds of objects playing different roles. Since UCE does not provide any further techniques after the business use cases have been converted to business objects, we have adopted Ramackers' Domain Business Object to visualise all the components and relationships within a business object. Then we move on to object and class diagrams for implementation.

3. Business Object Architecture

3.1 Problems of Business Objects

We discovered the need for a different approach to modelling business objects after experience with using the UCE approach alone. We found that Jacobson's method did not produce business objects that were flexible enough for reuse in different contexts: the use-case driven approach tended to fix the interfaces of business objects; and the objects tended to be too coarse-grained, encapsulating business data and business processes in ways that were hard to break apart. In reality, businesses always face a situation in which different business operations might share common processes, tasks and data. If we develop coarse-grained business objects one after the other and put them together within a business domain, we will end up with object redundancy / object overlapping. For example, when we want to perform some business transactions such as invoicing and insurance claims, we need to involve customers. Does it mean that we need to include the customer object in both the invoicing business object and insurance claims business object? If so, we are overlapping the customer object. If not, how do we share the same business processes and objects? Moreover, we also need workflow direction to describe the sequencing of different business transactions, and the way in which these sequences may be broken apart, adapted and reused.

3.2 What is an Architecture?

The problems of Business Objects have prompted the adoption of a business Object Architecture (BOA). An architecture is a set of rules to define the structure of a system and the interrelationships between its parts. The components within an architecture are the basic building blocks and tools. An architecture also contains patterns, which advise on how to combine basic components using the tools. The functions of a BOA are to represent the components that are used to 'model' the business problems and build the system [2].

3.3 BOA Framework

As with Business Object, there is no standard way to develop a BOA. Our approach is to adopt both top down and bottom up directions as shown in Figure 3. After defining the business process from a high level, we then identify all the necessary entity objects from the bottom up. Finally, we develop the business objects in the

middle layer by collecting the appropriate business processes, functionalities, attributes and operations together. Those business objects do not hold the business processes or the entity objects. Instead, they point to them when they want to use them. Different business objects thus share a single business process or entity object. The benefit of it is that when we change any business processes or entity objects, the updated versions will point to the relevant business objects. Another benefit of the BOA is that not only can we reuse the business processes and the entity objects but also we can reuse the business objects as a package. The reuse of business object will substantially improve software quality as developers can reuse the pre-defined and pre-tested object components.

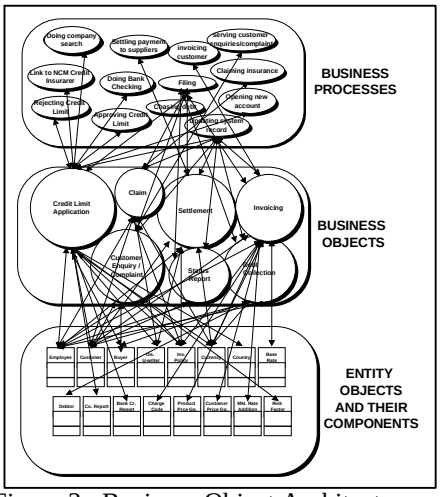

Figure 3 : Business Object Architecture

4. Business Object Repository / Reuse Library

4.1 Business Object Repository

The BOA framework emphasises the reuse of business processes and object components, the aim of the business object repository is to materialise it. Figure 4 shows the infrastructure of a business object repository in which the entity objects are at the central layer representing the common database in an organisation. The business processes, which normally form the common functionality in the business domain, are situated in the second layer. The outside layer contains different business objects. Both the entity objects and business processes are shared by the business objects.

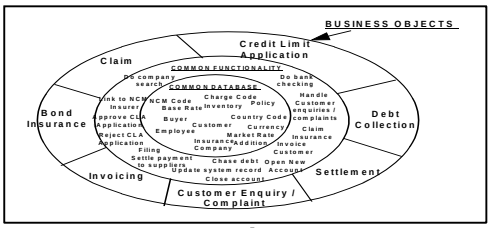

Figure 4 : Business Object Repository

4.2 Reuse Library

Ideally a reuse library is to be run in a CASE tool environment. As illustrated in Figure 5, the business process and entity object directories are the sub-directories of the business object directory. For example, if we click open the credit limit application (CLA) business object folder, the sub-folders of business process and entity object will prompt out on the screen. If we click open these sub-folders, all the relevant business processes and entity objects files will display on the screen. In this business object directory, those business processes and entity objects are "read-only" files. If we want to edit any of these files, we need to go to the business processes and entity objects main directories to do the changes. The updated versions of the business processes and entity objects will point back to the sub-directories of business processes and entity objects in business object main directory. The benefit of which is that we only have to change once no matter how many times the business processes and entity objects are reused by different business objects.

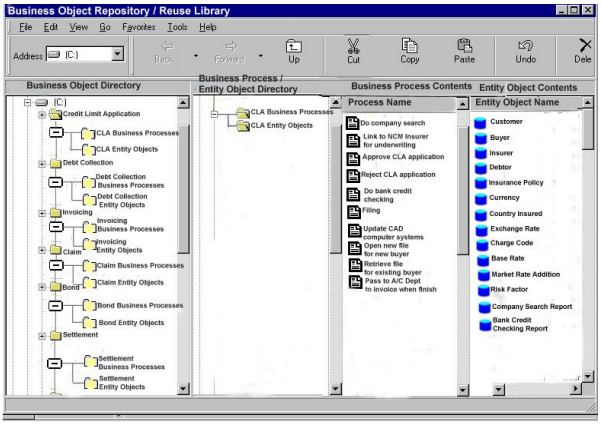

Figure 5: Reuse Library

5. Dynamic Systems Development Method

The Dynamic Systems Development Method (DSDM) is a formalisation of Rapid Application Development. Other than rapid development, DSDM also forms a vehicle to drive the IT developers and the business end-users together through its holistic approach such as emphasis on substantial end-user involvement, joint application development and joint requirement planning, function points, time-boxing, clean room technique, project estimation, usability testing, configuration management, change control, quality assurance and software procurement.

Traditionally, developers tend to put a subjective view on their work presuming that is what the real world needs. DSDM's fundamental assumption is that nothing is built perfectly first time. As a result all steps can be revisited as part of its iterative prototyping life-cycle. Therefore the current step needs be completed only enough to move to the next step. DSDM not only provides a life-cycle but also the necessary controls to ensure its success.

6. Dynamic Business Object Architecture

6.1 The Framework

The DBOA framework showing in Figure 6 is aimed at throwing one stone at two birds namely the 'quality' bird and the 'time & budget' bird. It is an integration between the BOA and DSDM. Amongst each life-cycle there is an incremental prototyping approach through these phases moving anti-clockwise from the top with feasibility studies, functional prototype, design prototype and implementation. The black arrows show the transfer points from one phase to the next and the grey arrows show where the development can easily return to an earlier phase. The white arrows indicate that the BOA model can always be re-architectured at different stages of the project phases.

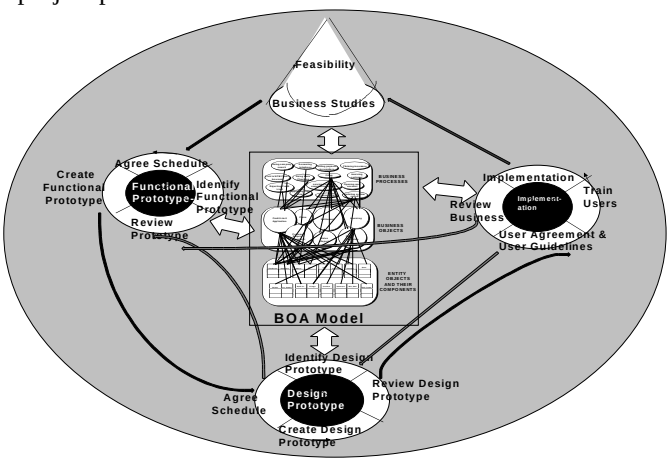

Figure 6 : Dynamic Business Object Architecture

6.2 Case Study Implementation : A Major Debtor Profile System for a Credit Insurance Agent

6.2.1 Background of the Project

The case study was carried out at a credit insurance agent, CAD Consultants Ltd.(CAD). When there is a transaction between seller and buyer, the buyer is given a certain length of credit period after the receipt of goods. The seller then insures the value of the products. Credit insurance is to protect the sellers (i.e. the insurance policy holders) from insolvency if their buyers fail to pay after the credit period. It is a commercial coverage by a contract binding a party to indemnify another against specified trading loss in return for premiums paid. CAD not only manages insurance policies on behalf of its customers but also has to determine the 'Credit Risk' of each buyer as well as the global risk exposure (political, economic, geographical) of the buyers' countries. This case study is to develop a Major Debtor Profile System to monitor the debt exposure. Under the credit insurance terms and conditions, any buyers who have credit are referred as debtors. The purpose of this project is to provide decision support to the business end-users on the approval/rejection of any future credit insurance applications.

6.2.2 Joint Requirements Planning (JRP) and Joint Application Development (JAD)

A 'Project Development Team' was formed consisting of three IT developers and four business end-users. After the initial JRP meeting of feasibility and business studies, JAD meetings took place at the end of each of the time-boxes. These took the form of a review of that time-box phase, and identifications of requirements for the next phase. During these 'End-of-project-phase' meetings, end-users were invited to test the prototype on-screen. The IT department staff would collect the end-users' comments and feedback to modify the prototype.

6.2.3 Project estimation by using function points

The project started with an estimation of the project size by using function points which are a method of estimating the "amount of functionality" required from an application and is also used to estimate project completion time and resources (human and finance) required. The basic idea involves counting screen inputs and other features of a description of functionality [8]. Figure 7 shows the essentials of this estimation.

The estimation works by identifying each function as easy, medium or difficult in terms of expected development 'complexity'. Function points were set after the Joint Requirements Planning (JRP) meeting with the end-users where requirements were obtained from them. As function points are the units used to measure the project, should there be changes in the user requirements during the project phases, the function points table will need to be re-scheduled accordingly. It is important to note that within a fixed timescale, it would be impossible to accommodate extra functionality without changes to the function point estimation. Therefore there are significant implications for the cost and/or duration of the project if such changes are required.

Functions	**Points Allocated (1=easy; 2=medium; 3=difficult)**	**Estimated developer-hours (easy=6 hours; medium=12hours; difficult=18hours)**
Declared Month / Year (user entry)	1	6
Customer No. (user entry)	1	6
Customer Name (auto display)	1	6
Policy No. (auto display)	1	6
Policy Name (auto display)	1	6
Client Reference No (user entry)	1	6
CAD Buyer No. (auto display)	1	6
Buyer Name (auto display)	1	6
Country Code (auto display)	1	6
Amount (user entry)	1	6
Currency Code (auto display)	1	6
GBP Equivalent (auto display)	1	6
Insured Limit (auto display)	1	6
Total amount of debt (in GBP equivalent) for a particular debtor should be added up and shown at the bottom of the screen (auto display)	3	18
The entry of Major Debtor Profile record is done monthly. Transfer previous month to history	2	12
The new screen for entry of current month's record will be cloned from previous month and the end-users overwrite it.	1	6
Report by Debtors	1	6
Report by Currencies	1	6
Report by Countries	1	6
Report by Customers	1	6
Report by Period (i.e. monthly record)	1	6
Report by Amount (in GBP)	1	6
Report by Teams	1	6
Update Main Menu	1	6
Total :	27 points	162 hours

Figure 7 : Function Point for Major Debtor Profile System

6.2.4 Time-Boxing

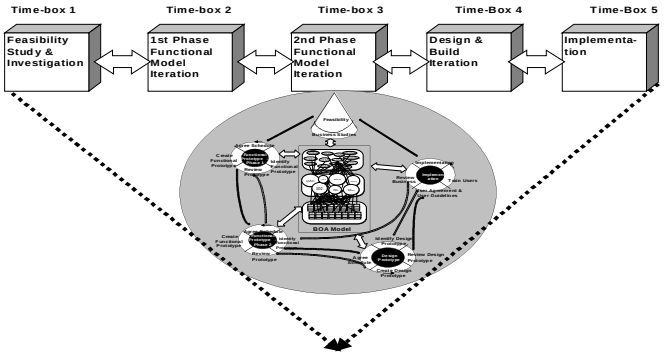

Figure 8 : Time-boxing for Major Debtor Profile System

As suggested by the DSDM manual, "Requirements can change, time can never slip" so heavy emphasis is placed on the importance of time-boxing technique as shown in Figure 8 to ensure project will be delivered on time thus within budget. DSDM defines time-boxing as 'setting a deadline by which a business objective must be met', and suggests that the boxes are set for the clearly defined delivery objects. This project consists of five time boxes namely:-

- ***Time-box 1*** **: Feasibility / Business Studies :** To get to know the user requirements. The project provided quite clear-cut requirements, in the technical context of a general need to provide information on all the current debtors for a particular customer. The Major Debtor Profile Interface is in fact a consolidation of different database files such as Customer, buyer, Credit Insurance Policy, Country, Currency and Exchange Rates. The Major Debtor Profile system correlates relevant data fields from different database files to the Major Debtor Profile Interface. When the users enter the customer reference number, this customer reference number data field will trigger other correlative data fields to display all the existing debtors' details for that customer as well as the insured limit where CAD's customer is allocated to each buyer. Over and above these features, the end-users can also view a profile by currencies, countries and period etc. This Major Debtor Profile system is to assist the end-users in making decision whether to approve or reject any future credit insurance application on a particular buyer.
- ***Time-box 2*** **: 1st Phase Functional Model Iteration :** To produce a version of the working system, from analysis and design model, notation to implementation, that could demonstrate to the user the essential features required to enable a user view the Major Debtor Profile and to test the prototype.
- ***Time-box 3*** **: 2nd Phase Functional Model Iteration :** To provide essential functionality to the model. The first phase had been mainly concerned with user interface design, with little in the way of 'business functionality'. An important issue of this early work in Phase 2 was to revisit the BOA framework

and its break-down to find out whether the conceptual model was the right shape to drive through to deliver systems / applications to the business.

- ***Time-box 4*** **: Design and Build Iteration :** To design and actually build the system. Hence by the end of this phase the system must contain absolutely all functionality, in a form which is suitable for testing.
- ***Time-box 5*** **: Implementation** : To test the system with the end-users which was purely to test the reliability and to debug the system. No extra requirements from the users were accepted within this Time-box. If the user had wanted to make further changes, we would have had to reschedule the project development life-cycle. During this project, end-users' approval was obtains and User Guidelines were prepared. Staff training programmes were conducted before system configuration, data take-on and system went live.

6.2.5 Prototyping Strategy

As time-boxing controls the pace of design and development that is so essential to the project, computer based techniques make this feasible. It would indeed be quite impossible to entertain the idea of tight schedule time-boxes without a means of maintaining both design documentation and implementation in a flexible and responsive way. The implication for the quality of the delivered product is quite clear. Therefore, the criteria of both the Business Object design model and interface prototype must clearly reflect the business, be flexible to change, quick to build / assemble and support reuse. In this project, a prototype strategy was to produce a version of the working system. During the feasibility and business study phase, a major debtor profile business object was created as shown in Figure 9.

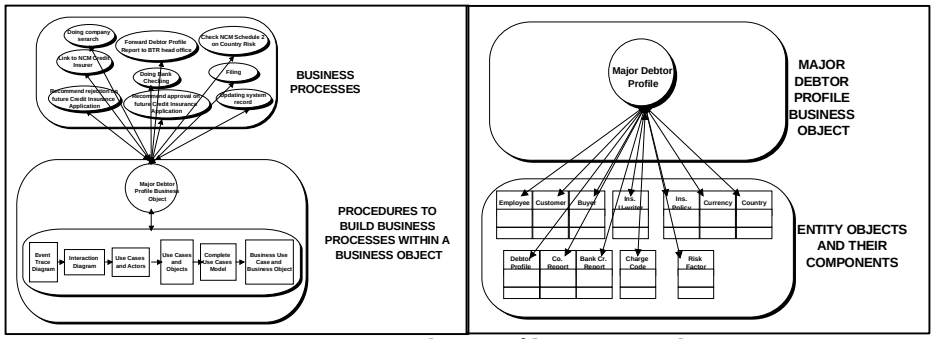

Figure 9 : Major Debtor Profile Business Object

Figure 9 is the breakdown of Figure 3 (Business Object Architecture) specifying one single aspect of the Major Debtor Profile. On the left hand side of Figure 9, a business process model starts with an Event diagram followed by an Interaction diagram, Use Cases and Actors, Use Cases and Objects, Complete Use Cases Model and Business Use Cases and a Business Object diagram. On the right hand side of Figure 9, relevant Entity Objects and their other components are identified to be used for the Major Debtor Profile Business Object.

The Unified Modelling Language (UML) notation [17,4] was chosen as the design notation for this project, as shown in Figure 10.

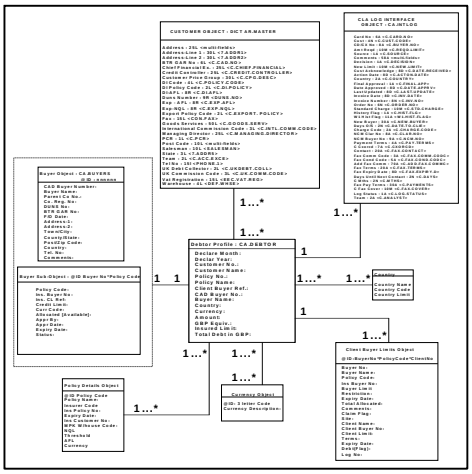

Figure 10 : UML notation

The interface was constructed using System Builder Plus [4] 4GL GUI tools to develop the interfaces that could be demonstrated to the users the essential features required to enable the application to be developed as shown in Figure 11.

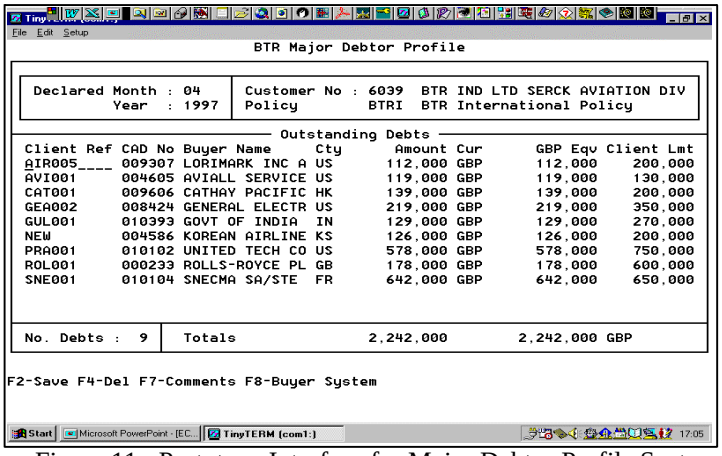

Figure 11 : Prototype Interface for Major Debtor Profile System

7. Project Evaluation

The 'Major Debtor Profile System' project had been delivered on time and was in operation. The BOA model was considered to be satisfactory as a vehicle to communicate with the end-users and interpret the business requirements for the new system and how it linked with other business objects. At the end of the project, the user community was satisfied, and had clearly felt very much involved in the whole process. The final success of the project was felt by the complete 'team'; not only the developers, but also the equally essential end-users who had been so actively involved in the JRP and JAD sessions and the user acceptance testing. The result of this case study has highlighted a few areas in which we feel that the DBOA showed

particular strength. These areas have also painted a picture of a successful interpretation of the method for a DBOA development in that:

- The reuse of existing business processes and entity objects within the BOA has reduced the development time and effort.
- The gap between the conceptual model and software implementation had obviously been narrowed. Had it not been the DSDM approach, we would not be able to check whether our conceptual BOA model is the right model for the business. The definition of good quality is largely the suitability to the business.
- Communication between developers and end-users was much better. The users were very much involved, to the point where 'the team' was quite definitely a description applicable to the mix of people, developer and user, involved in the project. There was an integration of the two roles; a change of relationship from supplier/consumer to partnership. The final system was our system, not their system. Equally significantly, if not more so, the users enjoyed the experience of taking responsibility for their own system. It is also worth mentioning that the experience was (most of the time!) enjoyable for the developers.
- The holistic approach, as a result of this partnership, has enabled the developers to obtain a better understanding of the business and its requirements. The intensity and effectiveness of the JRP and JAD sessions was beyond any doubt. The concept of getting the right people to concentrate exclusively on the problem, and of empowering them to make the right decisions, paid off. And because of the heavy involvement of the business end-users, an IT project has become more of a business project. This is consistent with the prototyping that the function of IT support is to solve business problems.
- The iterative approach to design worked. It enabled us to revisit the BOA conceptual model and modify it in response to the changes in circumstances. The first functional prototype was very much imperfect. But at least it was something for the user to work with. The process of refinement which went on through Time-boxes 2, 3 and 4 resulted in numerous opportunities to fix the imperfections.
- We met, with comparative ease, what would have been an impossible deadline using the conventional life-cycle.

The whole rationale of this paper is "To achieve the objective of delivering quality software on time and within budget". With this two-way echo between the developers and the end-users, we consider we have successfully brought these acronyms (BOA and DSDM) together through our experience obtained from the above case study. Such synergy quickly and effectively reacts to the business changes.

8. "S.M.A.R.T." Evaluation Criteria

An evaluation criteria framework called "S.M.A.R.T.", based on the characteristics of both the BOA and DSDM technique, has been developed to evaluate the DBOA schema in terms of:

'*S*'calable: As each business object component is individual, we can always increase the number of the business object without affecting the integrity of the existing one.
'*M*'easurable: Function Points were used to measure the size and complexity of the system. User Acceptance Testing was also used to measure the satisfactory level of the end-users on the system.
'*A*'chiveable: The holistic approach of DSDM life-cycle environment has increased the interactions between the end-users and the developers. Communication between them has thus been improved to enable the IT developers to deliver a software more achievable to the business requirements.
'*R*'eusable: The sharing of entity objects and process objects amongst business objects is the classic way of object reuse. Business objects themselves can also be reused as a package as well.
'*T*'ime-manageable: The time-boxing technique has provided a good control of time management to run project in order to deliver the system on time and within budget.

9 Conclusion And Further Research

9.1 Conclusion

In this paper, we present a DBOA to recommend a strategy for managing the time and budget against quality issue. An implementation of this is also presented through an insurance project case study. The DBOA techniques used in this paper should also be applicable to projects in any other business sectors. Although the result of the above case study is considered to be successful, DSDM is still not a mature technology. There are several 'challenging' areas where we would have to warn the developers when using the DBOA approach:-

- ***Friction between developers and end-users*** : there is always a situation where the developers and the end-users do not get along well.
- ***How to select the "right" people and to empower them to make "right" decisions?***: this is more to do with business issues and it can only be improved through experience.
- ***Time-boxing Syndrome***: everything is set inside a time-scale agreed with the business end-users. If planning is insufficient, developers would juggle between time-boxes. They will be forced to omit some unfinished tasks if they overrun the time-boxes or get panic to catch up at later time-boxes or they might have to abandon project if under pressure.
- ***Work Pattern / Paradigm shift for developers*** : the boundary between IT and business world is taken away. Developers have to cross the border to communicate with the business end-users and to experience business environment rather than developing the system in their own environment.

9.2 Further Research

Currently, we are investigating how to tackle the problems arisen from the conflict between developers and the end-users as well as the time-boxing syndrome. In the meantime, a business object repository is under construction using the Rational Rose

Version 4.0 UML CASE tools. Furthermore, a multiple projects case study will be carried out using the DBOA model to deal with complexity management.

10 References

1 Casanave C, Standardised Business Objects, Conference of Building & Using Financial Business Objects, London, UK, 22-23 October 1997 (http://www.omg.org).

2 Casanave, C, Business Object Architectures And Standards, Proceedings of Object-Oriented Programming, Systems, Languages & Applications (OOPSLA)'95 Conference Business Object Design And Implementation Workshop, Austin, Texas, USA, October 1995 (Eds. Sutherland et al) (Springer-Verlag, London) pp 7-28.

3 DSDM CONSORTIUM, Dynamic Systems Development Method Manual Version 3.0, 1997 (Tesseract Publishing, Surrey, UK).

4 Fowler M, UML Distilled - Applying The Standard Object Modelling Language, 1997 (Addison-Wesley Longman, Massachusetts).

5 Graham I, Migration to Object Technology, 1994 (Addison-Wesley, N.Y).

6 Hartha W et al, An Architecture Framework: From Business Strategies To Implementation, Proceedings of Object-Oriented Programming, Systems, Languages & Applications (OOPSLA)'95 Conference Business Object Design And Implementation Workshop, Austin, Texas, USA, October 1995 (eds. Sutherland et al) (Springer-Verlag, London) pp 47-60.

7 Hung K et al, A Dynamic Business Object Architecture For An Insurance Industry, Proceedings of Object-Oriented Information Systems (OOIS)'97 Conference, Brisbane, Queensland, Australia, 10-12 November 1997 (Eds. Orlowska et al) (Springer-Verlag, London) pp 145-156.

8 IFPUG, International Function Point Users Group, Ohio, USA (http://cuiwww.unige.ch/OSG/FAQ/SE/se-faq-S-2.html.#S-2).

9 Jacobson I et al, The Object Advantages: Business Process Reengineering With Object Technology, 1994 (Addison-Wesley, New York).

10 Jacobson I, Use Case Engineering Tutorial, Object-Oriented Programming, Systems, Languages & Applications (OOPSLA)'96 Conference, San Jose, California, USA, October 1996.

11 Martin J, Rapid Application Development, 1991 (Macmillan, New York).

12 OMG, Object Management Group - Object Management Architecture Guide, 1995 (John Wiley & Sons, Inc., New York).

13 Ramackers G et al, Object Business Modelling Request & Approach, Proceedings of Object-Oriented Programming, Systems, Languages & Applications (OOPSLA)'95 Conference, Austin, Texas, USA, October 1995 (Eds. Sutherland et al) (Springer-Verlag, London) pp 77-86.

14 SB+, System Builder Plus Version 2.3 Developer and Administrator Guide, 1995 (System Builder Technology UK Ltd., Prestbury, UK).

15 Stapleton J, DSDM: The Method In Practice, 1997 (Addison-Wesley, Essex, UK).

16 Sutherland J, The Object Technology Architecture: Business Objects For Corporate Information Systems, The 1995 Symposium for VMARK Users, Albuquerque, USA, 1995 (http://www.tiac.net/users/jsuth/).

17 UML97, Unified Modelling Language Version 1.0, January 1997 (http://www.rational.com).

18 Young J et al, Time For IS Professional To Come Out Of The Closet And Join The Party - Using A Strategic Change Framework To Understand The Influences on IS, 7th Annual Business Information Technology (BIT)'97 Conference, Manchester, UK, 5-6 November 1997 (Manchester Metropolitan University CD-ROM).