

# DSDM Development Techniques

## Table of Contents

<b>1</b>	<b>INTRODUCTION.....</b>	<b>2</b>
1.1	AIM.....	2
1.2	AUDIENCE.....	2
1.3	CONTRIBUTORS.....	3
1.4	DEFINITIONS AND TERMINOLOGY.....	3
1.5	SECTION STRUCTURE.....	3
<b>2</b>	<b>SUMMARY OF ISSUES WHEN SELECTING TECHNIQUES FOR DSDM.....</b>	<b>4</b>
<b>3</b>	<b>A MODEL-DRIVEN APPROACH TO DSDM DEVELOPMENT.....</b>	<b>5</b>
3.1	INTRODUCTION.....	5
3.2	A MODEL-DRIVEN APPROACH.....	5
3.3	THE NEED FOR MODELLING.....	6
3.4	MODELLING VIEWPOINTS.....	6
3.5	LEVELS OF MODELLING.....	7
3.6	MODEL REFINEMENT.....	7
3.7	MODEL REPRESENTATION.....	8
3.8	USER-CENTRED DESIGN.....	8
<b>4</b>	<b>DSDM DEVELOPMENT TECHNIQUES.....</b>	<b>10</b>
4.1	CHOICE OF TECHNIQUES.....	10
4.2	STRUCTURED APPLICATION DEVELOPMENT TECHNIQUES.....	10
4.2.1	<i>Interface Modelling Techniques.....</i>	<i>10</i>
4.2.2	<i>Process Modelling Techniques.....</i>	<i>12</i>
4.2.3	<i>Data Modelling Techniques.....</i>	<i>15</i>
4.2.4	<i>Behavioural Modelling Techniques.....</i>	<i>16</i>
4.3	OBJECT-ORIENTED DEVELOPMENT TECHNIQUES.....	17
4.3.1	<i>Object Modelling Techniques.....</i>	<i>17</i>
4.3.2	<i>Behavioural Modelling Techniques.....</i>	<i>18</i>
4.3.3	<i>Functional Modelling Techniques.....</i>	<i>19</i>
4.3.4	<i>Interface Modelling Techniques.....</i>	<i>19</i>
4.4	ADDITIONAL DEVELOPMENT TECHNIQUES.....	19
<b>5</b>	<b>CONCLUDING REMARKS.....</b>	<b>21</b>
<b>6</b>	<b>BIBLIOGRAPHY.....</b>	<b>22</b>

## **1 Introduction**

When Version 3 of DSDM was produced in 1997, it was agreed that the main Manual should concentrate of DSDM specific material and that information not unique to DSDM should be removed. Much of the information on DSDM development techniques was removed. However it was felt, as the technique guidelines provided useful support to the DSDM Practitioner, the removed information should form part of a DSDM White Paper.

The DSDM Manual advocates the adoption of a model-driven approach to system development. This White Paper identifies and describes the various development models that could be produced during any DSDM development, together with example techniques that may be employed to specify and refine these models.

DSDM divides the development of supporting business information systems into a number of phases. Each phase is characterised by a change in emphasis, for example, from Feasibility Study to Business Study, from Business Study to Functional Modelling.

This reflects the commonly held view that any method provides:

- a structured framework to define the development process;
- a set of techniques to models of the system under development;
- a set of procedures to assist in the derivation of the models;
- a set of heuristics to help in the development process;
- a common language to communicate design ideas and decisions.

This White Paper addresses the techniques and common language issues. For using UML-techniques see the White-paper “DSDM and UML”.

### **1.1 Aim**

The objective of this White Paper is to provide help, advice and guidance about the identification and selection of DSDM development techniques.

This White Paper only considers the techniques that could be applied in the development of the DSDM products which are not management-based. Techniques used in the following areas are excluded from this White Paper: Facilitated Workshop, Project Management, Quality Management, Configuration Management, Business Process Prototyping and Change Management.

### **1.2 Audience**

The audience is DSDM Practitioners responsible for the identification and selection of DSDM development techniques.

### **1.3 Contributors**

This White Paper was written by Dr. Tony Mobbs of IBM Global Services, based on the work done for DSDM versions 1 and 2 by the large Tools and Techniques Task Group which Tony chaired. Our thanks to all those who contributed ideas in the past.

The update of this White Paper to DSDM version 4.1 was completed by Kees Peeters and Rik Jan van Hulst.

### **1.4 Definitions and terminology**

In order to maintain consistency of terminology between the DSDM Manual and this White Paper, the following terms are used throughout this White Paper:

- **Model:** an abstraction of some characteristic of the business or system, as seen from a particular viewpoint
- **Technique:** a means by which a model is developed
- **DSDM Product:** a collection of one or more models
- **DSDM Tool:** computer-assisted support for one or more techniques.

### **1.5 Section Structure**

Section 2 provides a brief summary of the techniques section in the DSDM Manual outlining some of the key issues that need to be addressed when identifying and selecting development techniques for DSDM.

Section 3 outlines the rationale behind the model-driven approach adopted within DSDM.

Section 4 overviews some of the common development techniques and notations which can be used within DSDM.

Section 5 summarizes the key points of the paper.

Section 6 provides a short bibliography for further reading on system development techniques.

## **2 Summary of Issues when Selecting Techniques for DSDM**

The on-line DSDM Manual introduces some of the concepts and thinking behind the development techniques for DSDM. For completeness some of the issues which need to be addressed when identifying and selecting development techniques for DSDM are summarised below. For their full description the reader is referred to the on-line DSDM Manual.

### **Rapid development**

DSDM is concerned with providing business solutions in a timely manner, hence any technique must not impose any undue burden.

### **Easily Understood by User and Developer alike**

DSDM requires user involvement throughout the development lifecycle, therefore the users must easily understand any technique used.

### **Communications**

Any technique used during system development should aid the communication between all parties, whether it is between users and developers or between developers and maintainers of the system.

### **Semantic Gap**

This represents the different viewpoints taken between the users, the developers and the technology. These differing viewpoints historically have led to deep misunderstandings.

### **Structured Development verses Object Oriented Development**

DSDM does not recommend any preferred development approach. However some methods are more productive than others.

### **Integration with existing techniques**

DSDM is not prescriptive about the techniques that should, or should not be used. If any organisation has a preferred set of techniques, then those techniques should be used, but maybe tailored or complemented with techniques outlined within this White Paper.

The decision as to which technique, or set of techniques, will be deployed on any given DSDM development will be the responsibility of the nominated technical authority for the project.

### **3 A Model-Driven Approach to DSDM Development**

#### **3.1 Introduction**

When we wish to buy something, what are the various stages we may go through? Often we may see an advertisement in the paper, on television or maybe through the post. This is most frequently good enough for us to make a decision to buy.

For more expensive purchases, such as a washing machine or a video recorder, we may need to see a more detailed specification of its functionality, any constraints it may have and so on. Again, often reading these specifications gives us enough information to make a decision as to which product to buy.

In some cases, we may wish to see a model of what the final product may look like. For example, if you are having a new building erected, then an architect will produce a scale model of the building, showing its key features, its aspect and so on. This representation of the new building is often all there is to decide whether to go ahead with the building programme or not.

For some luxury purchases, such as a house or a new car, often it is best to look at the real thing. So a visit to a show house or a test-drive in a demonstration car is required to make a final decision to purchase.

In each of the above examples, we are making a decision to purchase based on a representation of the final product you are about to purchase. The representation exists at different levels of abstraction, from the very abstract picture in the advertisement to the physical car. In each case they can be considered to be a representation, or model, of the final product. However, the representation was “good enough” to make a decision to move to the next stage – namely to purchase.

If this approach serves us well in our every day decision making for selecting which product to purchase, then can we not take a similar approach to the development of solutions to business problems in DSDM?

#### **3.2 A Model-Driven Approach**

Successful software development depends on the ability to apply a set of techniques to build models of the proposed system which represent differing views, or aspects, of the system. This approach to software development is referred to as “model-driven”.

The approach adopted in identifying suitable techniques for any DSDM development is to concentrate on those techniques that can be easily understood by users and developers alike and which provide a powerful means of communication between all interested parties.

The preferred approach is to consider the development process as a series of refinements of various models of differing aspects of the system being produced. This “model-building” approach is inherent in a number of formal development methods. The

techniques within DSDM describe how these various views of the systems can be modelled and how the information within the models should be collected and interrelated.

### **3.3 The Need for Modelling**

Real-world systems are very complex and highly interactive. The main problems facing the DSDM Practitioner are the management and control of this complexity. The generally agreed approach is to describe the system using a set of models that is sufficiently constrained as to describe the features of the system thought to be essential, but at the same time filtering out non-essential information. Building models of complex systems is a difficult and error-prone activity and must not be approached without careful consideration of the modelling language to be used.

### **3.4 Modelling Viewpoints**

Modelling helps the DSDM development team gain a good understanding of the business domain. In understanding the problems, accurate models can be produced which reflect the realities of the business world. This understanding can be gained by analysing the problem from different viewpoints. Common views taken of the system are:

- **The processing view** which models the system as a set of business processes, or activities, which transform input data items to output data items. Processes can be either combined to form higher level processes, which in turn can be combined again to form yet higher level processes, or decomposed into their constituent sub-processes. This corresponds to the traditional “Why, What and How” type of questioning used during requirements elicitation.
- **The data view** which models the business information as a set of objects, or entities, and the relationships that exist between these objects. Various classes of relationship exist with a system, for example, aggregation, association, and generalisation.
- **The behavioural view** which models the behavioural characteristics of the system in terms of a set of events and states, where events cause changes in the states of the system. Events may be generated within or external to the system.
- **The people view** which describes the various interaction between parts of the organisation, their jobs and tasks.
- **The network view** which models the locations at which the business operates, the distributed architecture of the system.
- **The business view** which models the motivation of the organisation in terms of business strategy, objectives and plans.
- **The user interface view**, which models the interactions and interfaces between the system user and the system itself.
- **The object-oriented view**, which models the system as a set of interacting objects. An object-oriented view combines the above three views into a common view.

The above views of the system can be used to generate differing levels of models. For some complex systems, where the non-functional requirements of the system are considered to be a prime risk, additional models may have to be developed to model such system characteristics as security or performance.

Within DSDM some of the above models can be animated, in the form of various prototypes.

In summary, models are:

- an effective means of communication between all parties
- a way of capturing the essence of a problem, or design, such that it can be mapped into another form without loss of detail
- a means of providing abstraction by hiding unwanted detail
- a representation that provides insight into and understanding of the business
- an effective mechanism for scoping the business domain
- a record of decisions which will aid the maintenance of the system
- a means of providing traceability between different DSDM stages, iterations, increments and prototypes.

### **3.5 Levels of Modelling**

The above views of the system can be used to generate differing levels of models. Three levels of model are usually developed:

- 1) **Physical models** which represent a current implemented and operational system. The model describes physical file stores and processes introduced to meet particular implementation constraints.
- 2) **Logical models** which are descriptions of the system that are technology-independent. The development of logical models involves the abstraction and logicalisation of the physical model.
- 3) **Conceptual models** which are high-level descriptions of the universe of discourse for the system from a particular viewpoint.

The advent of formal methods has introduced additional types of models based on algebra and set theory. In formal methods the conceptual model has its foundation firmly based in mathematics. It is unlikely that these models would be developed during a normal DSDM development.

### **3.6 Model Refinement**

The models developed within one set of DSDM iterations or perhaps a Timebox are refined later. Initially, models specify abstract concerns of the system. As the

development process continues, these abstract concerns become more concrete and move closer to implementation. For example:

- The high-level requirements specified during the Business Study stage describe the behavioural aspects of the system without resorting to the detail of how those behaviours are achieved.
- A functional model hides the detailed information of the internal communications and data stores from the potential users of the system.
- A design prototype hides the detailed control and calling mechanism between programming modules.
- A module specification hides the detail of the chosen programming language.
- A high-level programming language hides the complexity of the underlying assembler code which itself is a representation of the contents of the store in the computer.

### **3.7 Model Representation**

Traditionally, various symbols are used to represent differing abstractions within a system. For instance:

- rectangular boxes or circles represent processing abstractions
- straight or curved lines denote data flow abstractions
- open-ended rectangular boxes denote data abstractions.

From these basic building blocks, together with their supporting text, large complex systems can be modelled.

### **3.8 User-centred design**

The techniques selected to support DSDM initially concentrate on modelling the aspects of the system seen from a user's perspective. This approach has been called "User-Centred Design" (UCD), since the techniques are not only easily understood by the user but they also model concepts familiar to the user. UCD contains not only the techniques to describe the internal processing and data aspects of the development, but also a set of techniques for the development of user interfaces, in respect of both requirements and constraints. User interface techniques are oriented both to the user and to more formal models, and include the following:

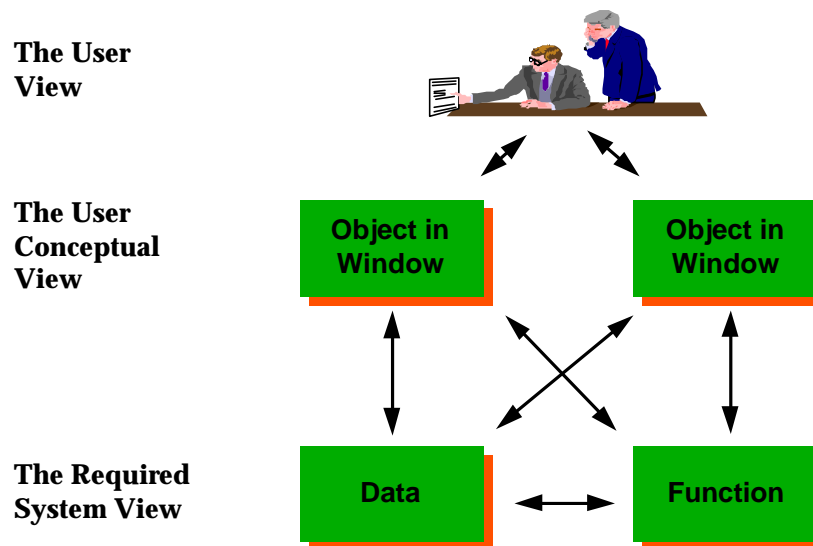
- **User Analysis** which provides insight into the range and responsibilities of the end-users of the proposed system and produces a catalogue of the various classes of users, their jobs, skills, access requirements, etc.;
- **Usability Analysis** which determines the characteristics of the proposed interface design which will satisfy the users' non-functional requirements



- **Task Modelling** which models the various activities to be performed by the users of the system
- **Task Scenario Definition** which identifies particular instances of task execution for any given system user
- **User Conceptual Modelling** which produces a model of the system that is simply understood by the users of the system
- **Graphical User Interface Design** which produces the interface for the user which provides support for the identified tasks, within the project constraints
- **User Interface Prototyping** that provides an animated view of the proposed design of the user interface.

Finally, these initial user-centred models are refined further by the application of more formal developer-centred design techniques. Further refinements are performed on the formal models of the system, until the most detailed model of the system is developed, namely code itself.

Figure 3.1 shows how the user view of functionality and data from outside the computer is mapped onto system components through objects in the user interface. It is important to keep the mapping from the user view of tasks to actual system tasks as simple as possible. This is achieved by good design of the Users' Conceptual Model (the objects in the window and their inter-relationships).



*Figure 3.1. The role of the User Conceptual Model*

## 4 DSDM Development Techniques

### 4.1 *Choice of Techniques*

The choice of possible techniques for use on a DSDM development is potentially quite large. Differing techniques will be applicable to differing phases of the project. In addition, some techniques will be more appropriate for use on differing developments. This will, in part, be dependent on the type of business system development. Finally, there is currently a dramatic move towards the use of object-oriented techniques, in preference to the more traditional development techniques. DSDM must be flexible enough to accommodate such diversity and range of interests.

The proposed set of techniques that are applicable to a DSDM development is reviewed below. The techniques are considered within two classes of system development:

- Structured Application Development, using structured development techniques;
- Object-Oriented Application Development, using object-oriented development techniques, including Use Cases from UML

The set of techniques proposed is for guidance only. Given any particular DSDM development, only a subset of the techniques may be applicable. It is worth emphasising that the various modelling activities are conducted in order to reduce the level of uncertainty on the project. For some projects, using certain development environments, such as 4GLs, some functions are simply implemented from a high-level description. In such cases, the intermediate levels of refinement are not necessary.

It is not the intention within DSDM to mandate a set of modelling notations, as it is envisaged that when DSDM is introduced into a user organisation, existing development techniques will continue to be employed. However, for the examples illustrated within this White Paper, the following notations will be used:

- Yourdon notation for process modelling
- SSADM-like notation for data modelling;
- UML for the object-oriented approach.

### 4.2 *Structured application development techniques*

For DSDM developments where structured development techniques are to be employed, the set of techniques discussed below are applicable. Within each group of modelling techniques, the techniques are listed in increasing order of refinement. In addition, the techniques are discussed with no particular structured method in mind. The underlying modelling concepts are constant across many development methods. The only feature that will differ is the particular notation used within the method.

#### 4.2.1 Interface Modelling Techniques

##### **User Analysis**

User analysis determines the variety and numbers of the user population for the proposed system. In addition, user analysis identifies the likely skill levels of the user population.

User analysis involves producing a description of the differing users of the system, which will have a view or a stake in the new system. The range of users includes trainers, maintainers and, for some systems, security officers.

#### *Use in DSDM*

This should be done very early on in the DSDM lifecycle. By understanding all the classes of end users and their potential use of the proposed system, the project can ensure that the views of all classes of end user are adequately represented.

All user analysis products are discarded or stored centrally for the use of later projects. They are not required for maintenance purposes, but they can reduce effort in user analysis for future projects in the same business area.

### **Task Analysis**

Task analysis is concerned with modelling the various tasks to be performed by the users of the system. Task analysis is an important activity as it ensures that the proposed user interface supports the users' tasks.

#### *Use in DSDM*

The products from task analysis can guide the flow of early prototypes for a particular user task. Task models should not go down to the detail level of atomic actions. They should identify the users' task goals and decision points. The detail will be elicited during prototyping.

### **Scenario Modelling**

Scenario modelling is an important technique that provides context to the task analysis exercise. It helps to provide realistic examples of the environment in which the users may find themselves in the proposed system. In addition, scenario modelling helps to identify the main system events, to which the system must respond. These are usually recorded as a set of "Use Cases" for each user role within the system.

#### *Use in DSDM*

Using a scenario to take a user through a prototype allows the user to think through the business issues of using the system. Without a scenario based on what the developer thinks is actuality, the developer and user can believe they have common understanding, when in fact they don't.

When prototyping, it is useful to provide a user with a scenario to work through and to ask the user to talk about what they are doing and thinking. In this way, the developer can capture not only functionality problems but usability as well.

Scenario modelling can be used to help generate test cases for system acceptance. It also assists in defining candidate prototypes to be developed.

## Conceptual Modelling

Conceptual modelling is a key technique in identifying what information should be displayed to the users, together with any rules and relationships that must be preserved by the interface.

A conceptual model is the end users' mental image of the structure and contents of the system. An excellent example of a conceptual model is the map of the London Underground. The diagram showing the different routes in differing colour schemes is an adequate model to allow the planning of routes between stations. In reality of course, the London Underground is a very complicated system.

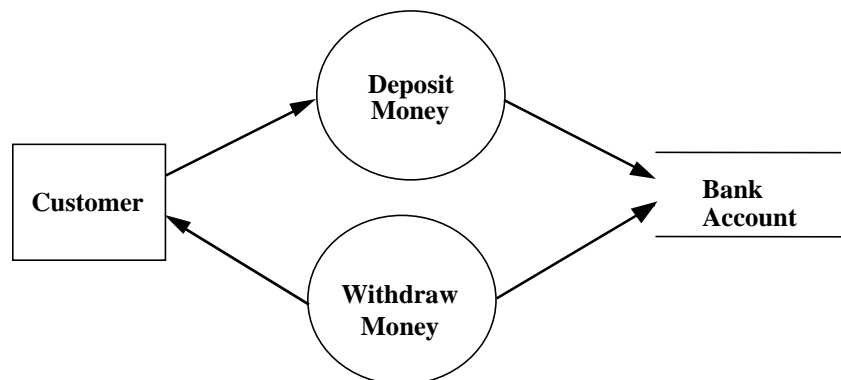
### *Use in DSDM*

Conceptual modelling is a powerful technique for ensuring that user tasks match system tasks as closely as possible. Therefore it should be developed alongside the prototypes and maintained during development of the user interface to ensure consistency between the system and the user. In DSDM development projects, the conceptual model can be produced within a facilitated workshop and can provide a common working understanding of the terminology used within the business. On multi-team DSDM projects, it provides a common working document. The conceptual model provides important input to the training material.

## 4.2.2 Process Modelling Techniques

### Data Flow Diagrams

Data flow diagrams (DFDs) have been used for many years to help model many different situations. The main advantage of DFDs is their ability to communicate ideas of what the system is doing, without going into too much detail.



The main components within a data flow diagram are:

- the **external entities** which represent the users or other systems external to the system being developed
- the **processes** which represent the transformations or manipulations of data

- the **data flows** which represent the pathways along which data may flow
- the **data stores** which represent data which must be held within the system over a period of time.

#### *Use in DSDM*

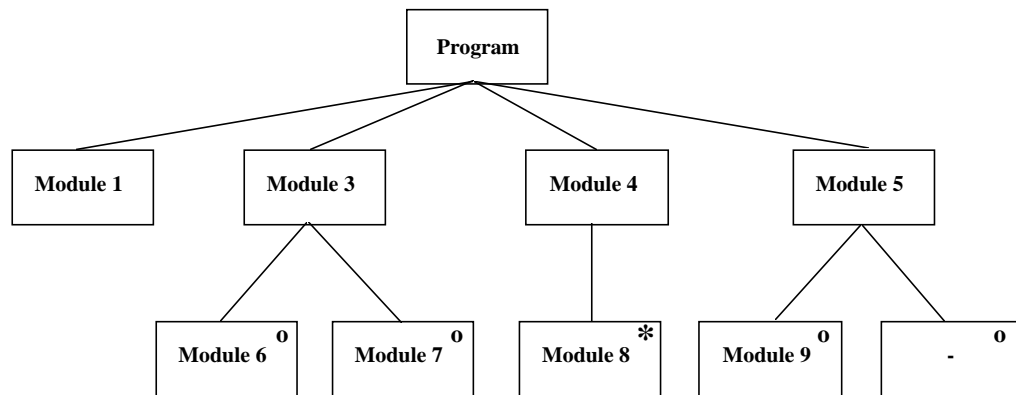
It is recommended that DFDs be used at a high level only to scope the business functionality to be supplied by the system. The DFDs can then be used to partition the functional prototyping work into suitable chunks or, on larger projects, to separate the work effectively between teams.

The details of data flows, which have been assigned to different developers or teams, need to be kept up to date to ensure the system will integrate successfully. However it is not necessary to detail any process down to the atomic level unless there is some degree of uncertainty which needs to be resolved before prototyping can begin. Indeed detailed work on describing processes is to be discouraged since the detailed functionality will be recorded in the dynamic functional model.

Data flow diagrams should be kept as part of the system documentation.

### **Jackson Structure Charts**

The Jackson structure chart (JSC) is a powerful modelling technique used traditionally for determining the structure of the programs that have to be developed. JSCs can be used to model both the logical and the physical structure of the program and of the data.



The main components of a Jackson Structure Chart are:

- **sequence boxes** which represent the time ordering control of the modules within the program
- **selection boxes** which represent the choice of process to be performed, given a certain condition
- **iteration boxes** that represent repeat processing given certain looping conditions.

## Use in DSDM

Jackson structure charts should not be used to perform detailed program design in DSDM. It is expected that the tool support should make this unnecessary. Also since the programs are evolving through prototyping, the effort in keeping Jackson structure charts in step with the system under development would be too great.

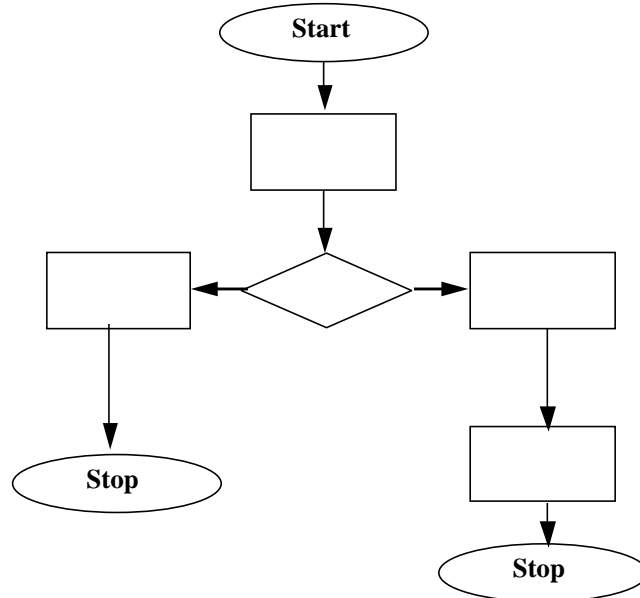
They are useful, in the absence of any other user interface modelling technique within an organisation, for clarifying menu structures and interface navigation in terms of sequence, iteration and selection. They should be viewed as working documents to be produced as required by the developers and should be discarded once they have served their purpose.

## Flow Charts

Flow charts date back to the earliest days of computing. The flow chart is a vehicle for specifying the detailed logic flow within a computer program. Flow charts represent the physical structure of the program.

The main components within flow charts are:

- **starting/stopping nodes** which represent the beginning and termination of the processing
- **sequence boxes** which represent the time ordering of the processing
- **decision nodes** that represent a possible change in logic flow within the program, given a particular condition.



## Use in DSDM

Flow charts are difficult to maintain and should not be used for any investigative work. However they can be useful for recording the tasks that a user will perform once the

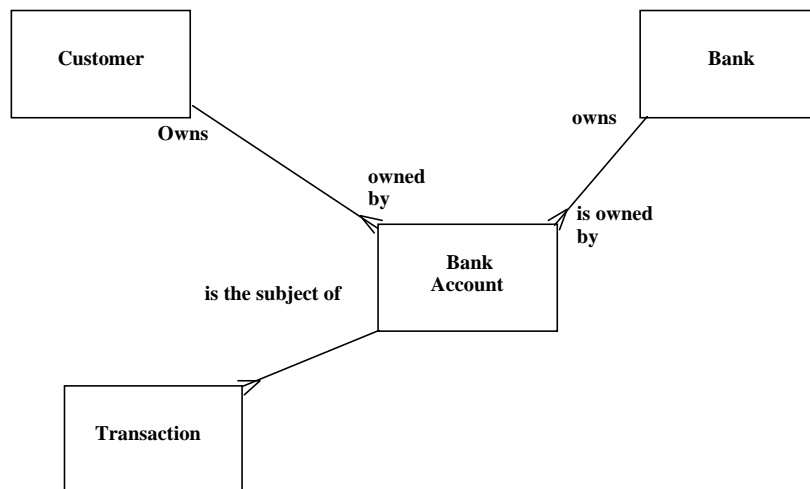
system has been built. The tasks can be both manual and system related. If the users are happy for them to be included in the User Manual then they should be, otherwise they should be thrown away once the user tasks have been formally documented.

#### 4.2.3 Data Modelling Techniques

##### Entity-Relationship Diagrams

Entity-relationship diagrams (ERDs) are a powerful aid to identifying the structure of the data within the system. Many variations of the ERD are in common usage. However the main components of the entity-relationship diagram are:

- the **entity** which is a conceptual grouping of data items to be stored within the system
- the **relationship** that represents an association between two or more occurrences of different entity types.



In addition to the entities and the relationships, the attributes of the entities must also be described.

##### *Use in DSDM*

For communication purposes the ERD should stay unnormalised for as long as possible. Users do not understand the need for some of the entities in a normalised data model and can get bogged down in the detail of these.

A high-level ERD should be produced before any prototyping work is undertaken showing unnormalised entities and their relationships. Important attributes are expected to have been identified but should only be recorded as names plus logical data type: lengths, etc. will evolve during prototyping activity.

The ERD must be fully worked through before going into design work, including all attributes, relationships, volumes, access rights, etc. Indeed with some tools, this level of

detail may be pushed higher up the lifecycle, since they assume a fully defined database before prototyping can begin.

The ERD should be kept as part of the system documentation together with all related information, such as logical attribute descriptions.

### **Relational Data Analysis**

Relational data analysis (RDA) is a widely used technique for developing normalised relations from sets of unstructured data. RDA is a technique used to complement the ERDs and it is used to check that:

- all attributes required by the system processing are within the ERD
- the ERD represents a normalised view of the data, thereby eliminating unnecessary processing when data needs to be updated
- the ERD is sufficiently specified to reflect the processing requirements.

ERDs and RDA are two complementary techniques. ERDs develop the system 'top-down' and are usually constructed from the user requirements. RDA develops the data model from the 'bottom-up' from the collections of attributes.

#### *Use in DSDM*

This is a backstop technique only to be used when some of the business rules are unclear. All working papers, etc. can be discarded once the ERD has been normalised. RDA is a technique many users find difficult to grasp - use the conceptual model instead.

## **4.2.4 Behavioural Modelling Techniques**

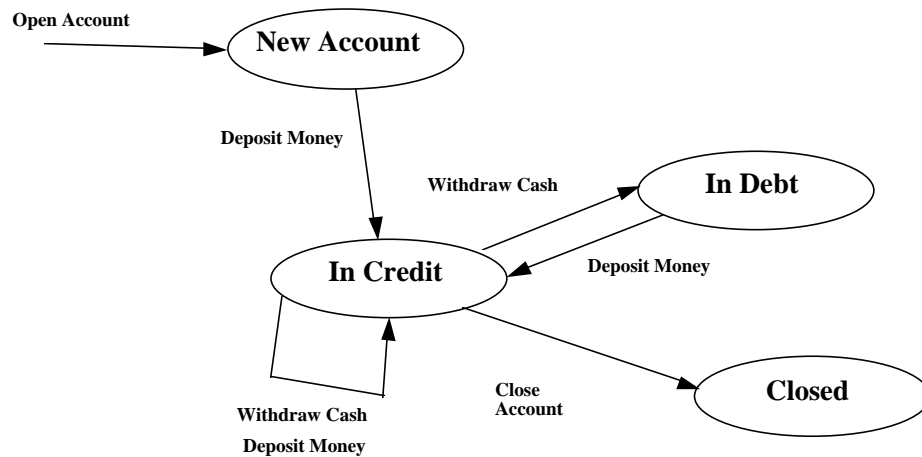
### **State Transition Diagrams**

The State transition diagram (STD) is a means of modelling time-dependent characteristics of the system behaviour. The STD is used to model what happens when. It provides a mechanism for modelling the essentials of control within the system.

The main components within STDs are:

- **states** which represent constant system behaviour which persists over a period of time
- **transitions** which represent a change from one state to another, which is usually the result of the arrival of some system event
- **conditions** which describe what must happen to cause the system to change state
- **actions** that describe what happen to effect a change of state.





#### *Use in DSDM*

Amongst other things, STDs are an excellent tool for clarifying navigation around GUIs. They can also be used to sort out the various possible states in complex decision structures.

In all but real-time systems (for which DSDM is not designed), they are usually no more than a way of sorting out options and clarifying complex system issues and as such should be thrown away once those issues have been understood and implemented successfully.

### **4.3 Object-oriented development techniques**

#### **4.3.1 Object Modelling Techniques**

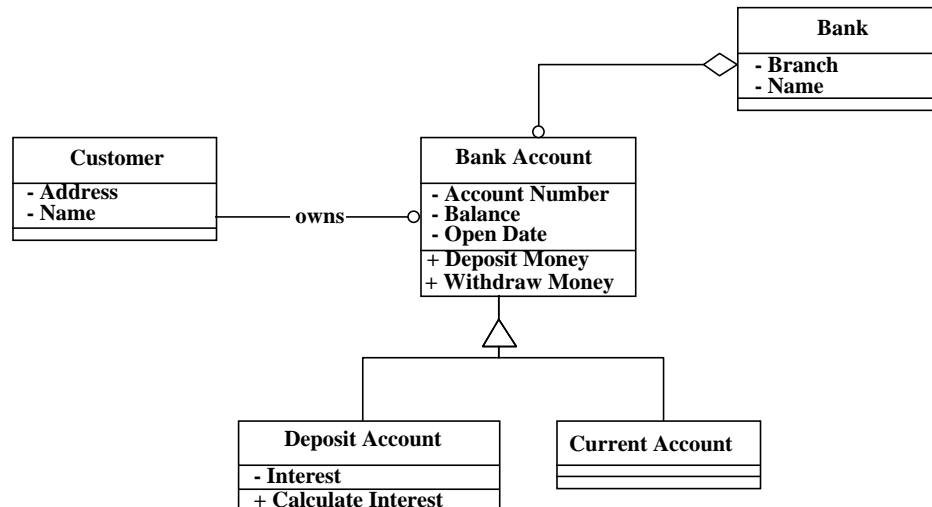
##### **Class Modelling**

An object model (often called a class model) captures the static structure of the system being developed. The model shows the classes in the system, the relationships between the objects, together with the attributes of the objects and the operations on the objects.

An object is a concept, abstraction, or something in the system with a clear boundary and which has meaning within the system. Objects represent real-world concepts; each has a unique identifier and must be distinguishable from each other. A Class describes a group, or collection, of objects that share common properties.

Relationships between classes are represented as connections between the classes. Relationships can be of three main types:

- association, which describes a simple linkage between two classes
- aggregation, which describes the “consists of” or “has parts” relationship
- generalisation, which describes the inheritance, or “isa”, relationship between classes.



Generalisation and inheritance are fundamental concepts within an object-oriented approach that are not normally supported by standard structured development approaches. The inheritance of attributes and operations of a class from its parent class in the generalisation structure provides a very powerful framework for reuse on a project and between differing projects.

An object model provides an important focus for identifying components of the system that may be candidates for prototyping.

#### *Use in DSDM*

A high-level object model identifying the major business-based classes should be produced before prototyping begins, including their key attributes and operations. It should be refined at the class level during prototyping to ensure that all classes are represented. As additional attributes and operations are identified they should be added to the object model.

The object model should be kept as part of the system documentation for maintenance purposes.

Note that the correlation between the object model and the user's conceptual model should be very high, so it is unlikely that the conceptual model needs to be kept in an object-oriented development project.

### 4.3.2 Behavioural Modelling Techniques

#### **Object Interaction Diagrams**

Object Interaction Diagrams show the various time-ordered interactions between the objects within the system following the arrival of some external event sequence. The scenarios, identified during the Interface Modelling activities, are a sequence of events, therefore the scenarios provide useful vehicles for identifying system-level events. Each event will transmit information from one object to another.

#### *Use in DSDM*

Object Interaction Diagrams provide an essential link between the static object model and the dynamics of the working system. The Object Interaction Diagrams help to animate the designs for the system and help to show where abnormal conditions can arise. Many designers now use the Object Interaction Diagram as a replacement for the more ambiguous data flow diagram.

#### **State Transition Diagrams**

State Transition Diagrams within an object-oriented approach, exhibit the same characteristics as State Transition Diagrams used in structured development approaches. However, within an object-oriented approach, the STDs are produced at the object level.

#### *Use in DSDM*

These are essential for “interesting” classes in identifying the transfer between key states. They are also useful in identifying pre- and post-conditions.

They need not necessarily be maintained during the life of the project, unless it is intended to pass them on to maintenance.

### **4.3.3 Functional Modelling Techniques**

#### **Data Flow Diagrams**

The functional model specifies the results of a computation without specifying how or when they are computed. The functional model specifies the meaning of the operations in the object model and the actions in the dynamic (behavioural) model.

The most commonly used technique for functional modelling within an object-oriented approach is the Data Flow Diagram.

#### *Use in DSDM*

Many OMT developers never produce the Functional Model. So the advice is this: if it is useful as a development tool then produce it but do not necessarily keep it for maintenance purposes.

### **4.3.4 Interface Modelling Techniques**

The Interface Modelling Techniques used within an object-oriented development are the same as those used in a structured application development.

## **4.4 Additional development techniques**

The above list of techniques is not meant to be exhaustive, but it provides a carefully selected range of techniques that could be applied to a DSDM development. Many other techniques can be used on DSDM projects. Indeed the approach proposed within DSDM is to capitalise on existing knowledge and experiences within any user organisation. However, care should be taken to document only what is absolutely essential and only to

the level of detail which enables understanding. The system should be documented in the code (or at the level at which it is generated). The interim documentation needs of slower projects are not those of DSDM.

Techniques for business modelling can be considered for inclusion in a DSDM project. Such techniques can be found in Checkland's Soft Systems Methodology and Maclean et al.'s Object-Oriented Requirements Capture and Analysis (ORCA).

## **5 Concluding Remarks**

This paper has provided the rationale behind the model-driven approach to system development recommended within DSDM and provided some of the detailed thinking behind the concepts introduced within the on-line DSDM Manual Version 4.1. Guidelines for selecting techniques for DSDM development are provided.

This paper does not recommend any preferred notation or approach, as this will vary from organisation to organisation. However, both structured and object-oriented application development approaches can be accommodated within the framework.

## 6 Bibliography

The following provide useful further reading development techniques.

Allen, P. and Frost, S., "Component-Based Development for Enterprise Systems", Cambridge University Press, 1998, ISBN 0-521-64999-4.

Ashworth, C. and Slater, L., "An Introduction To SSADM Version 4", McGraw-Hill, 1992, ISBN 0-07-707725-3.

Date, C., "An Introduction To Database Systems", Addison-Wesley, 1986, ISBN 0-201-19215-2.

Hix, D. and Hartson, H.R., "Developing User Interfaces: Ensuring Usability through Product and Process", Wiley, 1993, ISBN 0-471-53846-9.

Jacobson, I., Christerson, M., Jonsson, P. and Overgaard, G., "Object Oriented Software Engineering", Addison-Wesley, 1992, ISBN 0-201-54435-0.

Jacobson, I., Ericsson, M., Jacobson, A., "The Object Advantage - Business Process Re-engineering with Object Technology", Addison-Wesley, 1995, ISBN 0-201-42289-1

Maclean, R., Stepney, S., Smith, S., Tordoff, N., Gradwell, D., Hoverd, T. and Katz, S. "Analysing Systems: Determining Requirements for Object-Oriented Development", Prentice-Hall, 1994, ISBN 0-13-301433-9.

Redmond-Pyle D. and Moore A., "Graphical User Interface Design and Evaluation - A Practical Process", Prentice Hall, ISBN 0-13-315193-X

Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. and Lorensen, W., "Object-Oriented Modelling and Design", Prentice-Hall, 1991, ISBN 0-13-630054-5.

Shlaer, S. and Mellor, S.J., "Object-Oriented Systems Analysis", Yourdon Press, 1988, ISBN 0-13-629023-X.

Sowa, J.F. and Zachman, J A., "Extending and Formalising the Framework for Information Systems Architecture" IBM Systems Journal, Volume 31, Number 3, Pages 590 - 616, 1992

Tudor, D. and Tudor, I., "Systems Analysis and Design - A Comparison of Structured Methods", Macmillan, 1997, ISBN 0-333-72139-X

UML (1997), Unified Modelling Language version 1.0, Santa Clara, CA: UML Partners. (This is now an OMG Standard UML)

Yourdon, E. and Constantine, L., "Structured Design", Yourdon Press, 1976, ISBN 0-917072-11-1.

Yourdon, E., "Modern Structured Analysis", Prentice-Hall, 1989.

Zachman, J. A., "A Framework for Information Systems Architecture", IBM Systems Journal, Volume 26, Number 3, Pages 276 - 292, 1987