Software Engineering Methods

Session 6

SSADM4+ – Functions

© The NCC Education Limited, 2001

Session 6 – Modification History

Revision	Date	Revision Description
V1.0	19 August, 1997	For issue
V2.0	17 September 2001	Updated for new syllabus V2/2001.

© NCC Education Limited, 2001

All Rights Reserved

The copyright in this document is vested in NCC Education Limited. The document must not be reproduced by any means, in whole or in part, or used for manufacturing purposes, except with the prior written permission of NCC Education Limited and then only on condition that this notice is included in any such reproduction.

Information contained in this document is believed to be accurate at the time of publication, but no liability whatsoever can be accepted by NCC Education Limited arising out of any use made of this information.

Trademarks

NCC Education acknowledge that the trademarks and registered trademarks of products mentioned in this material are held by the companies producing them. Use of a term in this material should not be regarded as affecting the validity of any trademark or service mark.

Copyright of any screen captures in this material are the property of the software's manufacturer.

This material may contain some clipart, which is copyright to the Corel Corporation.

Session 6

SSADM4+ – Functions

Lecturer Guide

Session Specification

Duration

Allow approximately 1 hour 30 minutes for the lecture part of this session. There should also be:

- tutorial (approximately 1 hour);
- workshop (approximately 1 hour 30 minutes).

Objectives

By the end of this session the participant will be able to:

- understand the concept of function in SSADMv4/4+;
- identify functions;
- classify functions;
- specify functions (this involves function definitions and I/O.

Lecturer

The lecturer should have a good knowledge and experience of SSADM(v4/4+).

Method of Presentation

This session should be a lecture based on the material provided.

During the workshop, students should develop I/O Structures (for example, for Easybooks functions).

Material List

Visuals

V6. 1 Introduction
V6. 2 Contents
V6. 3 Universal Function Model
V6. 4 Classification of Functions in SSADM
V6. 5 Function Identification – 1
V6. 6 Function Identification – 2
V6. 7 Function Definition Form
V6. 8 I/O Structure Diagram – 1
V6. 9 I/O Structure Description – 1
V6. 10 I/O Structure Diagram – 2
V6. 11 I/O Structure Description – 2
V6. 12 Summary

Handouts

6.1 Easybooks – LDM
A Handout is also provided showing all the visuals presented during this session.

Equipment Required

The following equipment is required:

- overhead projector and screen;
- flipchart stand, flipchart and broad-tipped felt pens.

Bibliography

M Goodland & C Slater, *SSADM (v4) A Practical Approach*, McGraw-Hill, 1995.

P Weaver, N Lambrou, M Walkley *Practical SSADM4+*, Financial Times Pitman Publishing, 1998.

Session Summary

1 Introduction 6-1
1.1 Summary of Topics to be Covered 6-1
2 SSADM4/4+ – Functions 6-1
3 Types of Function 6-2
4 Identification of Functions 6-3
4.1 Identifying Update Functions 6-3
4.2 Identifying Enquiry Functions 6-3
5 Events and Functions 6-4
6 Documenting Functions 6-4
6.1 Function Definitions 6-4
6.2 I/O Structures 6-5
6.2.1 I/O Structure Diagram for *Add Book Copy* 6-5
6.3 Other User Interface Specification Techniques 6-6
7 Summary 6-7
Handout 6.1 – Easybooks – LDM 6-9
Visuals Handout

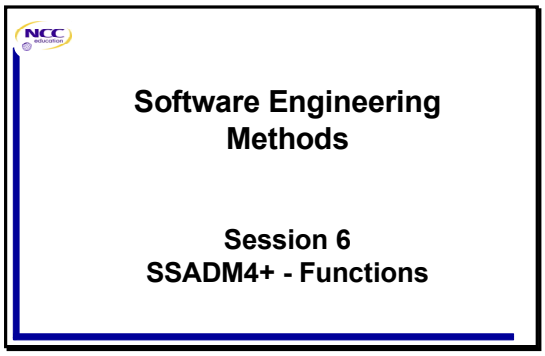
Session 6

SSADM4+ – Functions

1 Introduction

(5 minutes)

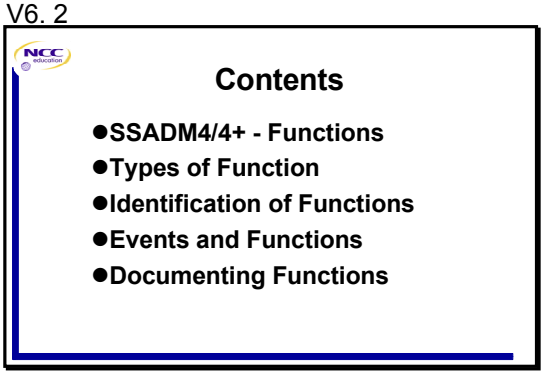
V6. 1



Inform students that a handout containing a full set of visuals will be provided to them at the end of this lecture.

1.1 Summary of Topics to be Covered

V6. 2



The topics detailed in the visual will be discussed during this session.

2 SSADM4/4+ – Functions

(15 minutes)

Functions are identified and specified in Stage 3 (Requirements Specification). They are becoming the basic blocks of processing now. All of the processing is defined through functions in and *after* Stage 3. Functions are like *processing specification packages*. They collect together elementary processes and user interfaces. In some sense a function corresponds to a *service* (a menu option) provided by the final system.

SSADM defines a function as:

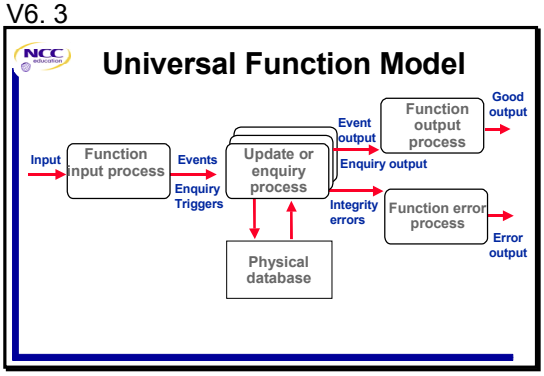
"A set of processing which the users wish to schedule together, to support their business activity",

or as:

"A set of processing designed to handle the effects of an event (or group of events) on the system" (this event-oriented definition is more useful in entity-event modelling).

SSADM introduces the concept of the *Universal Function Model* to describe the structure of a function. The standard components are shown in Visual V6. 3.

V6. 3



In Step 330 the following components are defined:

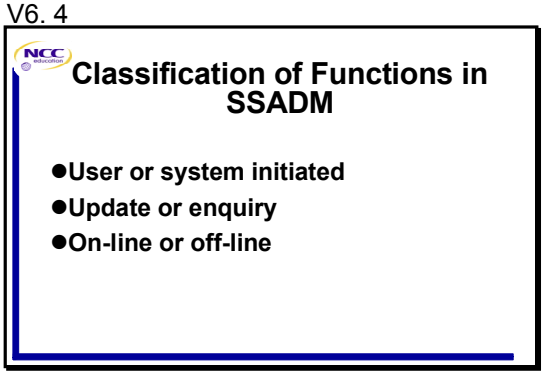
- Input.
- Valid output.
- Event or enquiry triggers.
- The function boundary.

We define other function components later. For example, in Stage 5 Update and Enquiry Processes are defined; in Stage 6 I/O Processes and Error Processes are defined.

3 Types of Function

(10 minutes)

V6. 4



Functions can be classified in three ways:

- *User or system initiated* – functions are triggered either by external events caused by users, for example, a user places an order, or by internal events, for example, a monthly report date has been reached.
- *Update or enquiry* – functions can either modify system data or retrieve system data.
- *On-line* or *off-line* – functions can operate in either interactive or non-interactive mode.

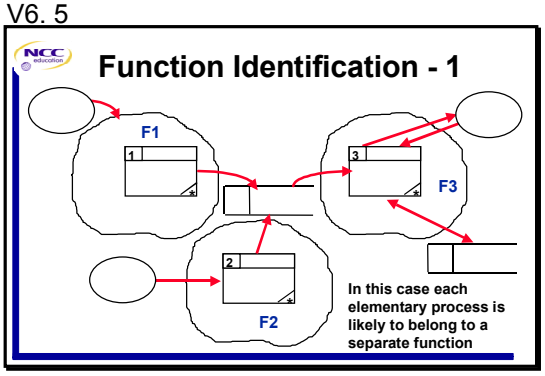
4 Identification of Functions

(15 minutes)

Functions are initially identified from the Required System Data Flow Diagrams (mainly update functions) and the Requirements Catalogue (mainly enquiry functions).

4.1 Identifying Update Functions

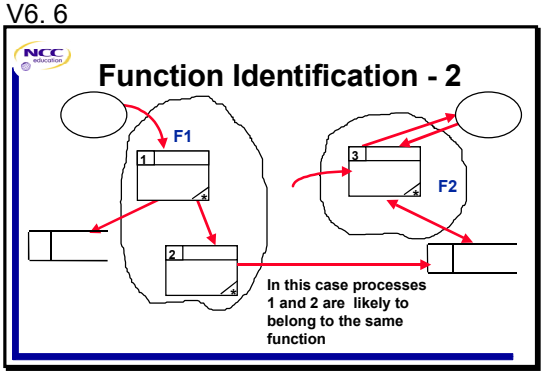
V6. 5



It is difficult to give precise guidelines for grouping elementary processes into functions, but some general guidelines are:

- Users decide which activities (processes) should be scheduled together.
- Group together processes performed together (for example, if two processes are linked by a direct data flow they should belong to the same function).

V6. 6



In general we group processes that *fit* together. They will constitute a function. Usually, however, functions will correspond to single elementary processes.

4.2 Identifying Enquiry Functions

Enquiries are originally documented in the Requirements Catalogue (produced in earlier stages). They will become separate enquiry functions initially. Later some enquiry functions can be merged or added to the definitions of update functions. Please note that some update functions may consist of update *and* enquiry processes.

5 Events and Functions

(5 minutes)

Events become very important at this stage.

Events are triggers of update processes. These triggers can be external (i.e., external events) or internal (i.e. internal events). Most events are external. The external events trigger update processes by input data flows.

Processes belong to functions. Therefore it is easy to allocate events to functions.

In simple systems the mapping of events, processes and functions is usually one-to-one-to-one. However, it is possible to have a number of events associated with a single function.

6 Documenting Functions

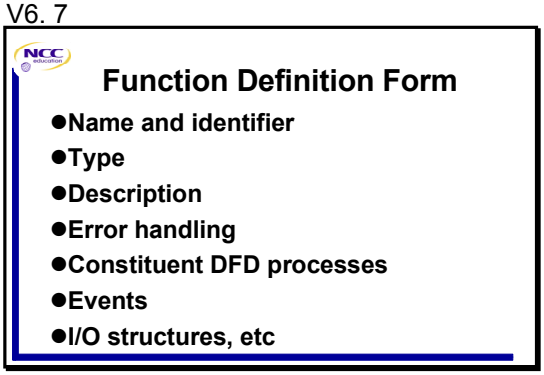
(30 minutes)

The two main function documents are Function Definitions and I/O structures.

6.1 Function Definitions

These are forms with the following sections (only main sections are discussed):

V6. 7



- *Function Name and Function Identifier* – should be unique within the project.
- *Type* – classification of the function (see Section 3 – Types of Function).
- *Description* – an informal description of the function's processing. This description is very similar to Elementary Process Descriptions (for update processes) and Requirements Catalogue entries (for enquiry processes).
- *Error handling* – very informal.
- *Constituent DFD Processes* – identifiers of processes included in this function.
- *Events* – names of events associated with this function.
- *I/O Structures* – references to I/O Structures describing the user-function interface, etc.

6.2 I/O Structures

I/O Structures specify user interfaces (i.e. interfaces between users and functions) using two tools:

- I/O Structure Diagram (the main tool);
- I/O Structure Description (an additional tool).

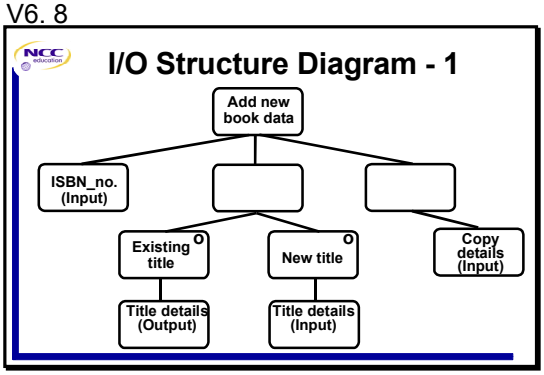
I/O Structure Diagrams use Jackson notation to describe precisely the structure of the user interface.

Normally there is one I/O Structure *per* function, although functions with on-line and off-line components will have at least two I/O Structures (for each type of interface).

The example below shows I/O Structure Diagram for function *Add Book Copy* (in the Easybooks system).

6.2.1 I/O Structure Diagram for *Add Book Copy*

V6. 8

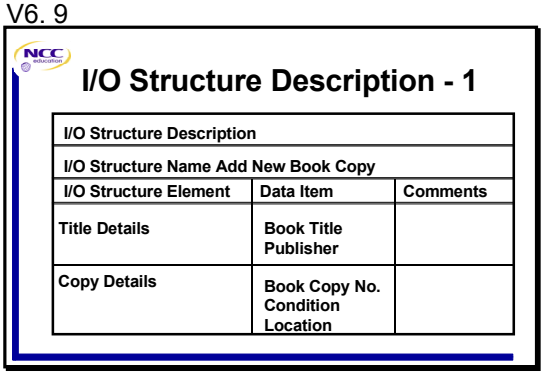


Explanation:

The top box contains the name of the I/O Structure. There are some structure (or intermediate) boxes to show how the data items are structured within the interface (for example, iteration boxes, selection boxes).

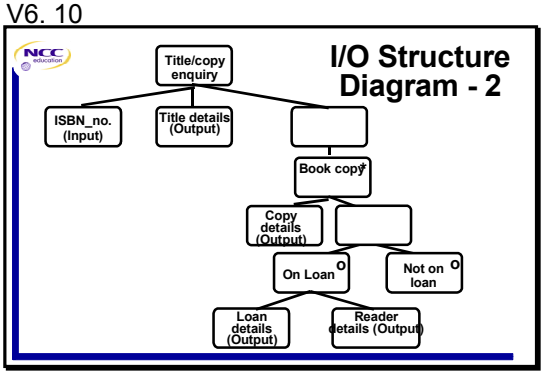
The bottom boxes represent actual groups of data items. Labels are used to indicate whether a particular group of data items is an input or an output group.

V6. 9



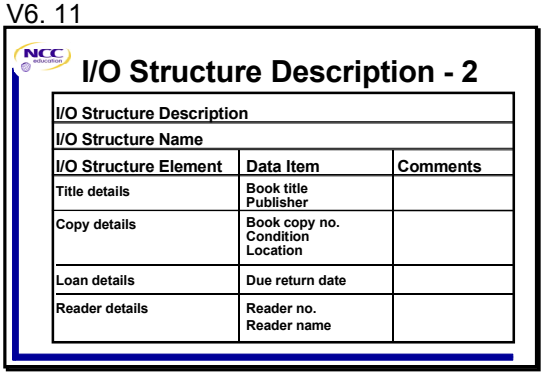
In order to specify the content of each input (or output) group, I/O Structure Descriptions are used (see Visual V6.9).

V6. 10



Another example is given in Visual V6.10 and V6.11 – I/O Structure for *Title Enquiry.*

V6. 11



6.3 Other User Interface Specification Techniques

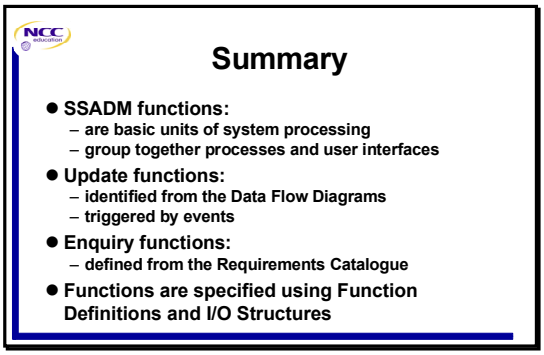
(5 minutes)

I/O structures are sufficient to represent relatively simple user-function interfaces/interactions. Richer screen environments (windows, icons, menus etc.) allow users to perform tasks involving more *intensive* interaction with a system. Therefore some extra user interface specification techniques may be required. These include task modelling, user object modelling, function navigation model, etc.

7 Summary

(5 minutes)

V6. 12



The concept of SSADM functions has been introduced. Functions are basic units of system processing (from the user perspective). They group together processes and user interfaces. Update functions are identified from the Data Flow Diagrams. Update functions are triggered by events. Enquiry functions are defined from the Requirements Catalogue. Functions are specified using Function Definitions and I/O Structures.

Issue Handout 6.1 – Easybooks LDM.

Handout 6.1 – Easybooks – LDM

Entity Descriptions

Reader = ReaderNo + ReaderName + ReaderAddr + NoOfLoans + NoOfReminders

- NoOfLoans means the number of past and current loans;
- NoOfReminders means the total number of reminders received (from the date of registration).

Title = ISBNNo + BookTitle + Publisher *etc*.

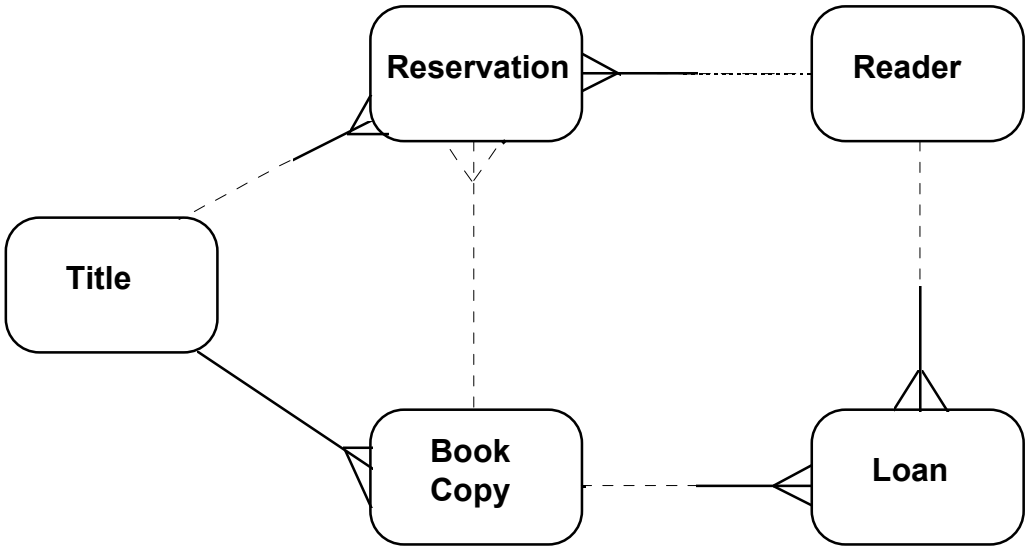
BookCopy = BookCopyNo + ISBNNo* + Condition + Location + NoOfLoans *etc*.

- NoOfLoans indicates how many times this copy has been on loan

Loan = ReaderNo + BookCopyNo + DueReturnDate + DateOfReminder1 + DateOfReminder2 *etc*.

Reservation = ReaderNo + ISBNNo + DateOfReservation + BookCopyNo*

Entity Relationship Diagrams (ERD)



Note: *Reservation* is linked with *Book Copy* when a copy of the reserved title is returned.

SSADM4 - Functions - 6.1

Software Engineering Methods

Session 6
SSADM4+ - Functions

V2.0 © NCC Education Limited, 2001

SSADM4 - Functions - 6.2

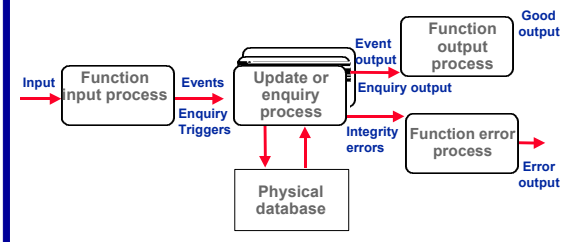
Contents

- SSADM4/4+ - Functions
- Types of Function
- Identification of Functions
- Events and Functions
- Documenting Functions

V2.0 © NCC Education Limited, 2001

SSADM4 - Functions - 6.3

Universal Function Model



V2.0 © NCC Education Limited, 2001

SSADM4 - Functions - 6.4

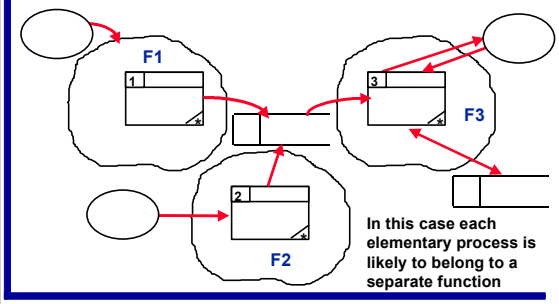
Classification of Functions in SSADM

- User or system initiated
- Update or enquiry
- On-line or off-line

V2.0 © NCC Education Limited, 2001

SSADM4 - Functions - 6.5

Function Identification - 1

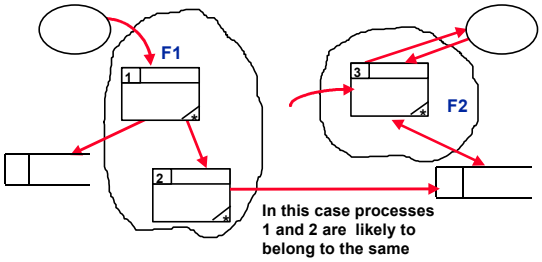


In this case each elementary process is likely to belong to a separate function

V2.0 © NCC Education Limited, 2001

SSADM4 - Functions - 6.6

Function Identification - 2



In this case processes 1 and 2 are likely to belong to the same function

V2.0 © NCC Education Limited, 2001

SSADM4 - Functions - 6.7

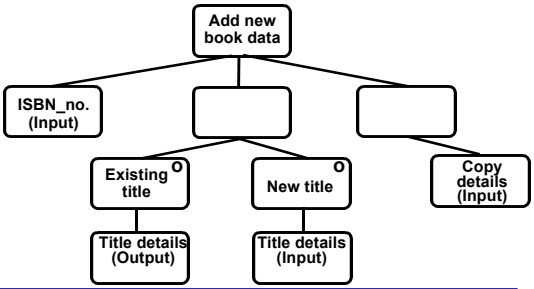
Function Definition Form

- Name and identifier
- Type
- Description
- Error handling
- Constituent DFD processes
- Events
- I/O structures, etc

V2.0 © NCC Education Limited, 2001

SSADM4 - Functions - 6.8

I/O Structure Diagram - 1



V2.0 © NCC Education Limited, 2001

SSADM4 - Functions - 6.9

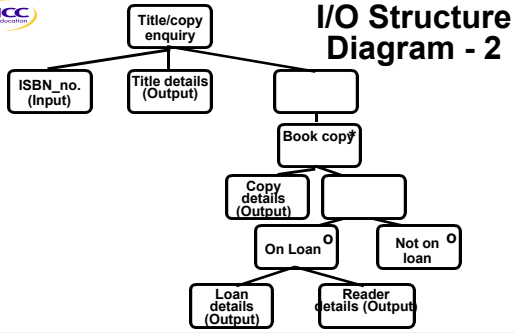
I/O Structure Description - 1

I/O Structure Description		
I/O Structure Name Add New Book Copy		
I/O Structure Element	Data Item	Comments
Title Details	Book Title Publisher	
Copy Details	Book Copy No. Condition Location	

V2.0 © NCC Education Limited, 2001

SSADM4 - Functions - 6.10

I/O Structure Diagram - 2



V2.0 © NCC Education Limited, 2001

SSADM4 - Functions - 6.11

I/O Structure Description - 2

I/O Structure Description		
I/O Structure Name		
I/O Structure Element	Data Item	Comments
Title details	Book title Publisher	
Copy details	Book copy no. Condition Location	
Loan details	Due return date	
Reader details	Reader no. Reader name	

V2.0 © NCC Education Limited, 2001

SSADM4 - Functions - 6.12

Summary

- SSADM functions:
  - are basic units of system processing
  - group together processes and user interfaces
- Update functions:
  - identified from the Data Flow Diagrams
  - triggered by events
- Enquiry functions:
  - defined from the Requirements Catalogue
- Functions are specified using Function Definitions and I/O Structures

V2.0 © NCC Education Limited, 2001

Software Engineering Methods

Session 7

SSADM4+ – Entity/Event Modelling – 1

© NCC Education Limited, 2001

Session 7 – Modification History

Revision	Date	Revision Description
V1.0	19 August, 1997	For issue
V2.0	17 September 2001	Updated for new syllabus V2/2001.

© NCC Education Limited, 2001

All Rights Reserved

The copyright in this document is vested in NCC Education Limited. The document must not be reproduced by any means, in whole or in part, or used for manufacturing purposes, except with the prior written permission of NCC Education Limited and then only on condition that this notice is included in any such reproduction.

Information contained in this document is believed to be accurate at the time of publication, but no liability whatsoever can be accepted by NCC Education Limited arising out of any use made of this information.

Trademarks

NCC Education acknowledge that the trademarks and registered trademarks of products mentioned in this material are held by the companies producing them. Use of a term in this material should not be regarded as affecting the validity of any trademark or service mark.

Copyright of any screen captures in this material are the property of the software's manufacturer.

This material may contain some clipart, which is copyright to the Corel Corporation.

Session 7

SSADM4+ – Entity/Event Modelling – 1

Lecturer Guide

Session Specification

Duration

Allow approximately 2 hours for the lecture part of this session. There should also be:

- tutorial (approximately 1 hour);
- workshop (approximately 1 hour).

Objectives

By the end of this session the participant will be able to:

- understand the concept of Entity Life Histories (ELHs), Effect Correspondence Diagrams (ECDs), and Enquiry Access Paths (EAPs) in SSADM4;
- use Entity/Event modelling techniques to produce ELHs.

Lecturer

The lecturer should have a good knowledge and experience of SSADM(v4/v4+).

Method of Presentation

This session should be a lecture based on the material provided.

During the workshop, students should produce more ELHs (for example, for Easybooks system).

Material List

Visuals

V7. 1 Introduction
V7. 2 Contents
V7. 3 ELHs and ECDs – Event/Entity Matrix
V7. 4 Entity/Event Modelling
V7. 5 Development of ELHs
V7. 6 Event/Entity Matrix
V7. 7 ELH for Reader
V7. 8 Death of the Master
V7. 9 Death of the Last Detail
V7. 10 Elementary ELH Operations (Weaver) – 1
V7. 11 Elementary ELH Operations (Weaver) – 2
V7. 12 ELH for Reader with Operations
V7. 13 State Indicators
V7. 14 Summary

Handouts

A Handout is provided showing all the visuals presented during this session.

Equipment Required

The following equipment is required:

- overhead projector and screen;
- flipchart stand, flipchart and broad-tipped felt pens.

Bibliography

M Goodland & C Slater, *SSADM (v4) A Practical Approach*, McGraw-Hill, 1995.

P Weaver, N Lambrou, M Walkley *Practical SSADM+*, Financial Times Pitman Publishing, 1998.

Session Summary

1 Introduction 7-1
1.1 Summary of Topics to be Covered 7-1
2 SSADM4+ – Entity/Event Modelling 7-1
3 Developing ELHs 7-2
4 Summary 7-6

Visuals Handout

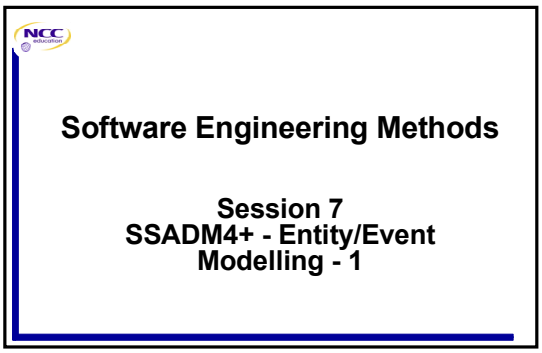
Session 7

SSADM4+ – Entity/Event Modelling – 1

1 Introduction

(5 minutes)

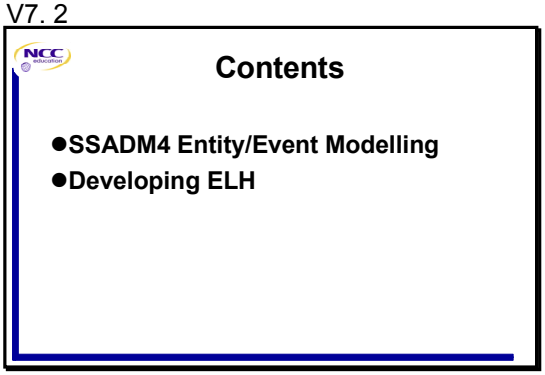
V7. 1



Inform students that a handout containing a full set of visuals will be provided to them at the end of this lecture.

1.1 Summary of Topics to be Covered

V7. 2



The topics detailed in the visual will be discussed during this session.

2 SSADM4+ – Entity/Event Modelling

(20 minutes)

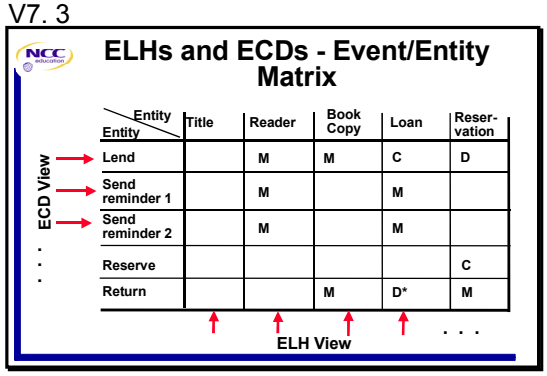
Using this technique (in step 360) we produce a detailed specification of the required system processing. We do not specify update and enquiry processes yet (they are going to be specified in Stage 5), but the picture of processing becomes much clearer as we describe how each entity is affected by events (using ELHs) and how each event affects different entities (using ECDs). Thus the update processing for the required system is specified in more detail.

A detailed specification for enquiries is produced using EAPs.

Entity/Event Modelling is a technique which *links* System Analysis with System Design. It ends the analysis stage and starts the design.

Entity/Event modelling brings together processing and data *views* to form the third view of the required system. This third view shows the system's dynamics. It shows how events affect the system's data (i.e. entities).

V7. 3

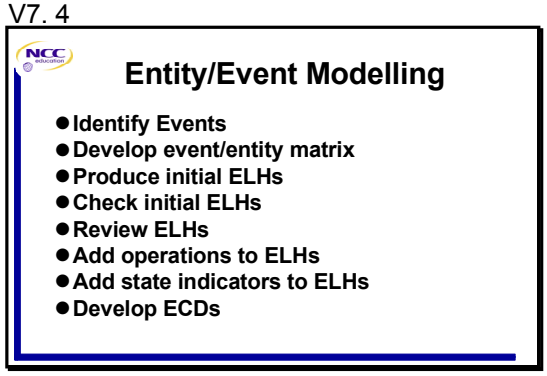


The entity/event modelling technique includes two sub-techniques/sub-models:

- Entity Life Histories;
- Effect Correspondence Diagrams.

The relationship between these two sub-models is shown in Visual V7.3.

V7. 4



A recommended approach (**M Goodland, C Slater**) to entity/event modelling is shown in Visual V7.4.

Note: Event/Entity Matrix is called Entity Access Matrix in SSADM v4+.

3 Developing ELHs

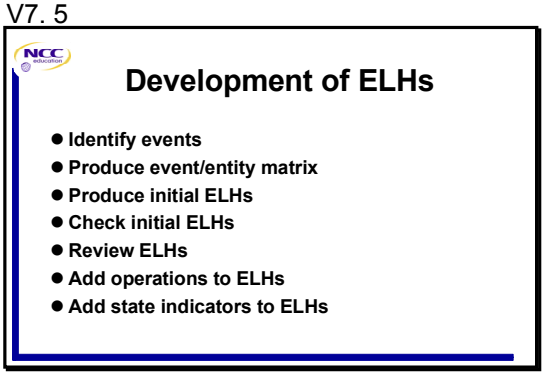
(60 minutes)

An Entity Life History is a chart/diagram showing all events which can affect an entity, from creation to deletion. ELHs show the allowable sequence, selection and iteration of these events.

ELHs use the same basic Jackson notation as for I/O Structures.

The development of ELHs is best carried out in the following steps:

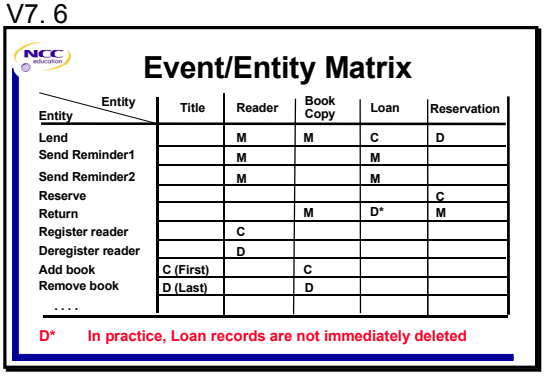
V7. 5



- *Initial identification of events.* The events were identified and allocated to functions in Step 330.
- *Develop Event/Entity matrix (Entity Access matrix).* All entities from the Logical Data Model (LDM) are placed along one axis. All events are placed along the other axis. The intersections show the effect of the event on the entity (C – create effect, D – delete effect, M – modification). To decide whether an event modifies an entity we must check if attributes of this entity are affected by this event.

Note: Version 4+ *introduced* a few more *subtle* effects such as B – bury, R – read etc.

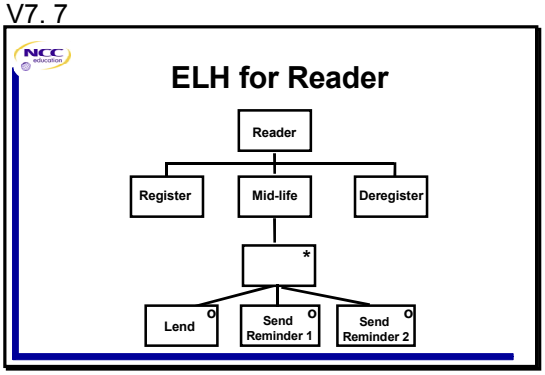
V7. 6



- *Produce initial ELHs.* Starting from the bottom of the Required System LDM (i.e. with detail entities) we develop simple/initial ELHs. Usually we develop a simple/initial ELH from left to right i.e. the *birth* section is drawn first. Next, the *middle life* section is developed. Finally the *death* events are added.

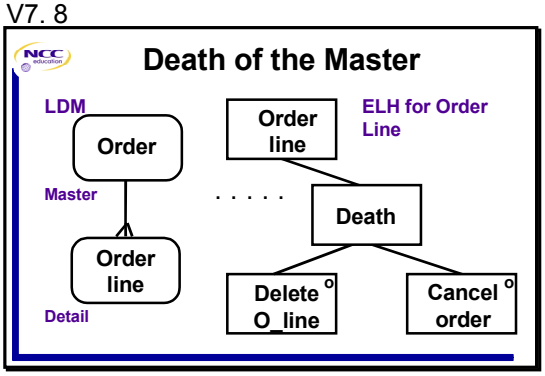
  If there is more than one event causing the *birth* of the entity instances, then the *birth* section will have a selection of these events. Similarly for multiple death events we will use selection in the *death* section. The *middle life* section is usually more complex than others and the order of events there reflects business rules.

V7. 7



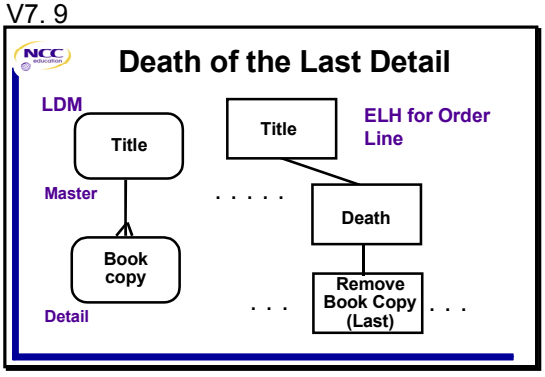
- *Check initial ELHs.* Having developed initial ELHs we have to check that all of the events have been identified. In other words we look for other events by asking such questions as:
  - Which events are responsible for the creation of attributes (i.e. for initialisations)?
  - Which events are respon-sible for modifications of attributes?
  - Which events establish relationships with master entities, and which events remove such relationships?
- *Review ELHs.* We review ELHs starting from master entities (i.e., from the top of the LDM). In particular we concentrate on interactions between master and detail entities such as:
  - death of a master causing the death of all its detail entities;
  - death of the last detail entity causing the death of its master.

V7. 8



In the first case, V7.8, the master's death event must be included in the ELH of the detail.

V7. 9



In the second case, V7.9, the detail death event should be added to the ELH of the master.

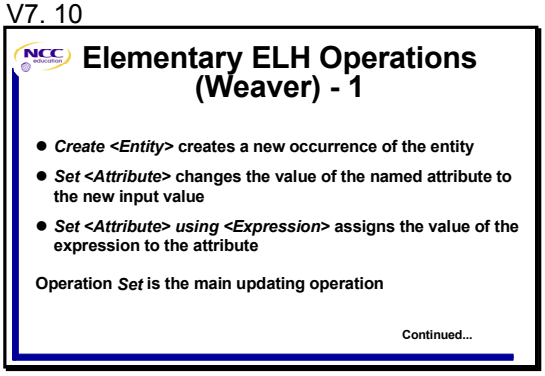
Add operations to ELHs. Basic operations are added to ELHs to show in more detail what actually happens when an event *hits* the entity (for example, the detailed effect is shown).

Every entity attribute should be updated by at least one operation.

- *Add operations to ELHs.*

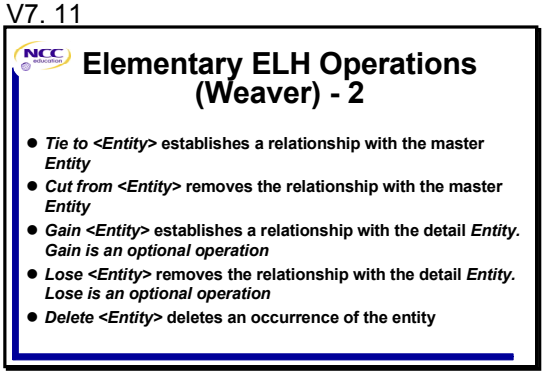
  SSADM provides a standard set of ELH operations:

V7. 10

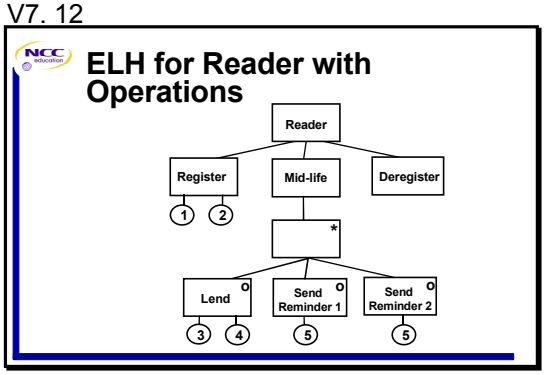


Visuals V7.10 and V7.11 show types of ELH operations.

V7. 11



V7. 12



Visual V7.12 shows the ELH for *Reader* (in Easybooks) with allocated operations.

Operations:

1. Create Reader
2. Set ReaderNo, Reader Name, Reader Addr = input
3. Set NoOfLoans =NoOfLoans+ 1
4. Gain Loan – Optional
5. Set NoOfReminders = NoOfReminders + 1.

- *Add State Indicators*

  ELHs specify the *right* order of events for each entity type. Of course when events occur in the *wrong* order then errors should be reported. These are called Integrity errors (for example, event *Delete Purchase Order* must occur after event *Create Purchase Order*, otherwise an Integrity Error should be reported). To detect these errors we must know the current state of the entity instance, and which events are *valid* in the current state. When a valid event

happens, we know which state follows the current one and change the current state appropriately.

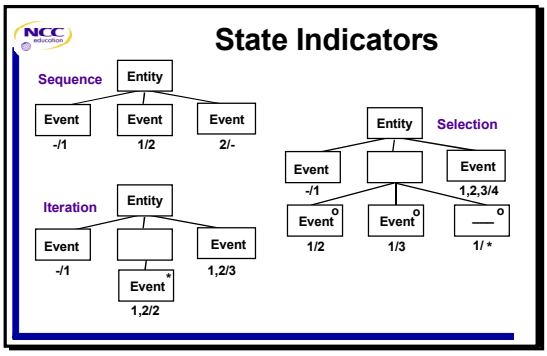
There are a few techniques to handle this (for example, State Transition Diagrams). In SSADM, *State Indicators* are used.

A State Indicator is an extra entity attribute. The initial value of the State Indicator is Null. To specify how State Indicators change we use the following notation:

Valid Previous Values/Value Set by Event.

- Valid Previous Values represent a list of the valid State Indicator values for the event to be allowed.
- Value Set by Event specifies the new State Indicator value set on completion of the event.

V7. 13

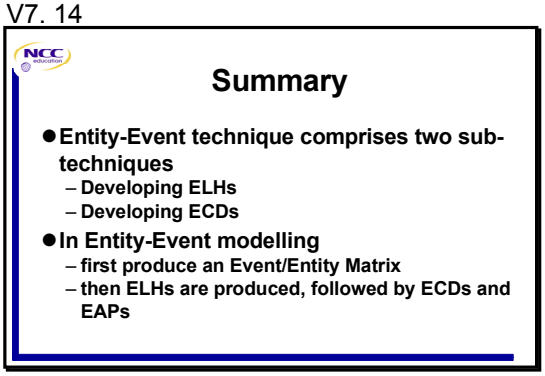


The example in Visual V7.13 illustrates how State Indicator expressions are added to ELHs representing sequence, selection, and iteration of events.

4 Summary

(5 minutes)

V7. 14



The Entity-Event technique has been introduced. This is a rigorous technique comprising of two sub-techniques:

- Developing ELHs.
- Developing ECDs.

The starting activity in Entity-Event modelling is to produce Event/Entity Matrix. Next ELHs are produced, followed by ECDs. EAPs can also be drawn at this stage.

Software Engineering Methods

Session 7
SSADM4+ - Entity/Event Modelling - 1

© NCC Education Limited, 2001

Contents

- SSADM4 Entity/Event Modelling
- Developing ELH

© NCC Education Limited, 2001

ELHs and ECDs - Event/Entity Matrix

Entity / Entity	Title	Reader	Book Copy	Loan	Reser-vation
Lend		M	M	C	D
Send reminder 1		M		M	
Send reminder 2		M		M	
Reserve					C
Return			M	D*	M

View ECD

ELH View

. . .

© NCC Education Limited, 2001

Entity/Event Modelling

- Identify Events
- Develop event/entity matrix
- Produce initial ELHs
- Check initial ELHs
- Review ELHs
- Add operations to ELHs
- Add state indicators to ELHs
- Develop ECDs

© NCC Education Limited, 2001

Development of ELHs

- Identify events
- Produce event/entity matrix
- Produce initial ELHs
- Check initial ELHs
- Review ELHs
- Add operations to ELHs
- Add state indicators to ELHs

© NCC Education Limited, 2001

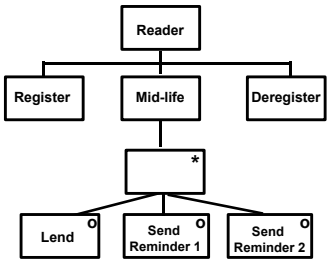
Event/Entity Matrix

Entity / Entity	Title	Reader	Book Copy	Loan	Reservation
Lend		M	M	C	D
Send Reminder1		M		M	
Send Reminder2		M		M	
Reserve					C
Return			M	D*	M
Register reader		C			
Deregister reader		D			
Add book	C (First)		C		
Remove book	D (Last)		D		
. . . .					

D* In practice, Loan records are not immediately deleted

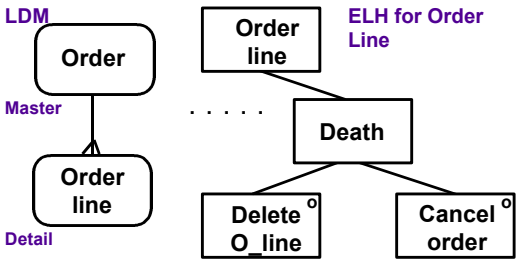
© NCC Education Limited, 2001

ELH for Reader



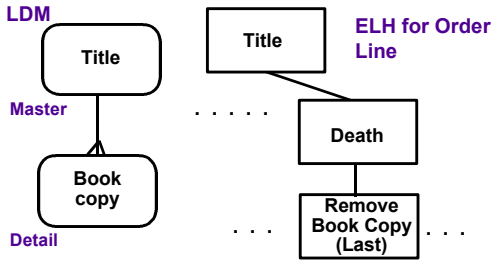
© NCC Education Limited, 2001

Death of the Master



© NCC Education Limited, 2001

Death of the Last Detail



© NCC Education Limited, 2001

Elementary ELH Operations (Weaver) - 1

- *Create <Entity>* creates a new occurrence of the entity
- *Set <Attribute>* changes the value of the named attribute to the new input value
- *Set <Attribute> using <Expression>* assigns the value of the expression to the attribute

Operation *Set* is the main updating operation

Continued...

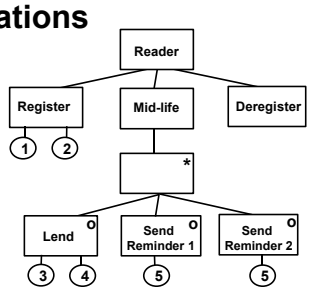
© NCC Education Limited, 2001

Elementary ELH Operations (Weaver) - 2

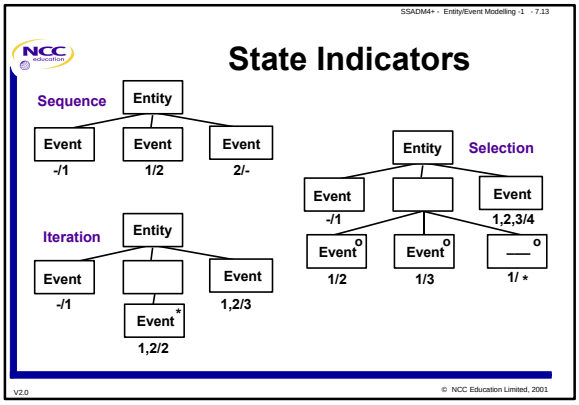
- *Tie to <Entity>* establishes a relationship with the master *Entity*
- *Cut from <Entity>* removes the relationship with the master *Entity*
- *Gain <Entity>* establishes a relationship with the detail *Entity*. *Gain is an optional operation*
- *Lose <Entity>* removes the relationship with the detail *Entity*. *Lose is an optional operation*
- *Delete <Entity>* deletes an occurrence of the entity

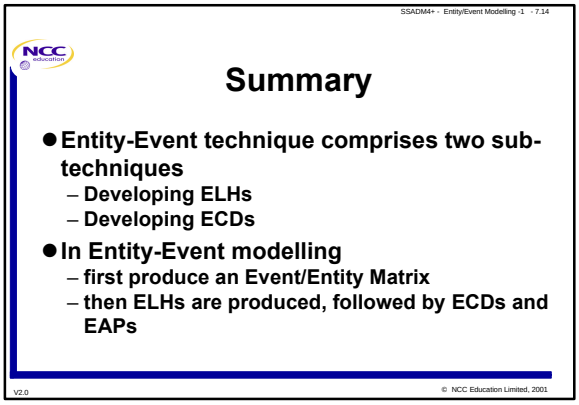
© NCC Education Limited, 2001

ELH for Reader with Operations



© NCC Education Limited, 2001


SSADM4+ - Entity/Event Modelling -1 - 7.13
State Indicators
Sequence
Entity
Event
-/1
Event
1/2
Event
2/-
Iteration
Entity
Event
-/1
Event*
1,2/2
Event
1,2/3
Selection
Entity
Event
-/1
Event
1,2,3/4
Event o
1/2
Event o
1/3
— o
1/ *
V2.0
© NCC Education Limited, 2001


SSADM4+ - Entity/Event Modelling -1 - 7.14
Summary
Entity-Event technique comprises two sub-techniques
– Developing ELHs
– Developing ECDs
In Entity-Event modelling
– first produce an Event/Entity Matrix
– then ELHs are produced, followed by ECDs and EAPs
V2.0
© NCC Education Limited, 2001

Software Engineering Methods

Session 8

SSADM4+ – Entity Event Modelling – 2 & Logical Process Design

© NCC Education Limited, 2001

Session 8 – Modification History

Revision	Date	Revision Description
V1.0	19 August, 1997	For issue
V2.0	18 September 2001	Updated for new syllabus V2/2001.

© NCC Education Limited, 2001

All Rights Reserved

The copyright in this document is vested in NCC Education Limited. The document must not be reproduced by any means, in whole or in part, or used for manufacturing purposes, except with the prior written permission of NCC Education Limited and then only on condition that this notice is included in any such reproduction.

Information contained in this document is believed to be accurate at the time of publication, but no liability whatsoever can be accepted by NCC Education Limited arising out of any use made of this information.

Trademarks

NCC Education acknowledge that the trademarks and registered trademarks of products mentioned in this material are held by the companies producing them. Use of a term in this material should not be regarded as affecting the validity of any trademark or service mark.

Copyright of any screen captures in this material are the property of the software's manufacturer.

This material may contain some clipart, which is copyright to the Corel Corporation.

Session 8

SSADM4+ – Entity Event Modelling – 2 & Logical Process Design

Lecturer Guide

Session Specification

Duration

Allow approximately 1 hour 30 minutes for the lecture part of this session. There should also be:

- tutorial (approximately 1 hour 30 minutes);
- workshop (approximately1 hour).

Objectives

By the end of this session the participant will be able to:

- use Entity Event modelling techniques to produce ECDs and EAPs;
- apply the Logical Process Design technique to produce Update and Enquiry Process Models;
- understand the rôle and importance of this technique in SSADM.

Lecturer

The lecturer should have a good knowledge and experience of SSADMv4/v4+.

Method of Presentation

This session should be a lecture based on the material provided.

During the workshop, students should develop selected ECDs, EAPs and Process Models (for example, for Easybooks system). In particular they should derive an Update Process Model (UPM) from the second ECD for Lend Book event (see Handout 8.3).

Material List

Visuals

V8. 1 Introduction
V8. 2 Contents
V8. 3 ECD for Lend Book
V8. 4 'Extra' ECD operations
V8. 5 ECD operations (summary)
V8. 6 ECD for Lend Book with operations and conditions
V8. 7 EAP for Title/Copy enquiry
V8. 8 EAP for Title/Copy enquiry with operations and conditions
V8. 9 Universal Function Model
V8. 10 Group Accesses and convert to EPM
V8. 11 EPM with operations and conditions
V8. 12 Group Effects and convert to UPM
V8. 13 UPM with operations and conditions
V8. 14 Summary

Handouts

Handout 8.1 ECD for Lend Book with Operations and Conditions
Handout 8.2 Update Process Model (UPM) for Lend Book with Operations and Conditions
Handout 8.3 ECD for Lend Book (more *sophisticated*)
A Handout is also provided showing all the visuals presented during this session.

Equipment Required

The following equipment is required:

- overhead projector and screen;
- flipchart stand, flipchart and broad-tipped felt pens.

Bibliography

M Goodland & C Slater, *SSADM (v4) A Practical Approach*, McGraw-Hill, 1995.

P Weaver, N Lambrou, M Walkley *Practical SSADM4+*, Financial Times Pitman Publishing, 1998.

Session Summary

1 Introduction 8-1
1.1 Summary of Topics to be Covered 8-1
2 Developing Effect Correspondence Diagrams 8-1
2.1 Allocating Conditions and Operations 8-2
3 Developing Enquiry Access Paths (EAPs) 8-4
3.1 Allocating Conditions and Operations 8-4
4 SSADM4+ – Logical Process Design 8-5
4.1 Logical Process Modelling (Logical Design of Update/Enquiry Processes) 8-6
5 Enquiry Process Modelling 8-7
5.1 Carry Forward EAPs 8-7
5.2 Group Accesses in One-to-One Correspondence 8-7
5.4 Allocate Operations and Conditions 8-7
6 Update Process Modelling 8-8
6.1 Carry Forward ECDs 8-8
6.2 Group effects on ECD which are in One-to-One Correspondence 8-8
6.3 Allocate Operations and Conditions to the Processing Structure 8-8
7 Summary 8-9
Handout 8.1 – ECD for Lend Book with Operations and Conditions 8-10
Handout 8.2 – Update Process Model for Lend Book with Operations and Conditions 8-12
Handout 8.3 – ECD for Lend Book (more 'sophisticated') 8-14
Visuals Handout

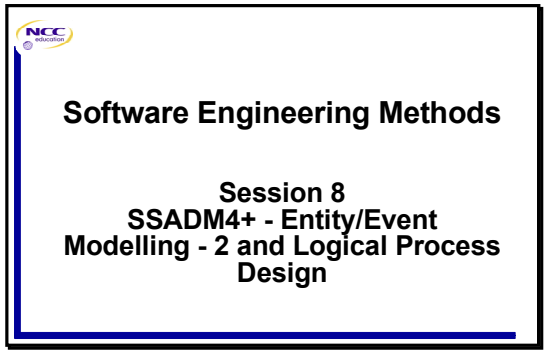
Session 8

SSADM4+ – Entity Event Modelling – 2 & Logical Process Design

1 Introduction

(5 minutes)

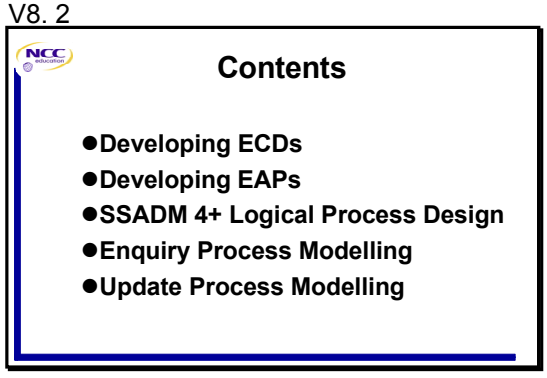
V8. 1



Inform students that a handout containing a full set of visuals will be provided to them at the end of this lecture.

1.1 Summary of Topics to be Covered

V8. 2



The topics detailed in the visual will be discussed during this session.

2 Developing Effect Correspondence Diagrams

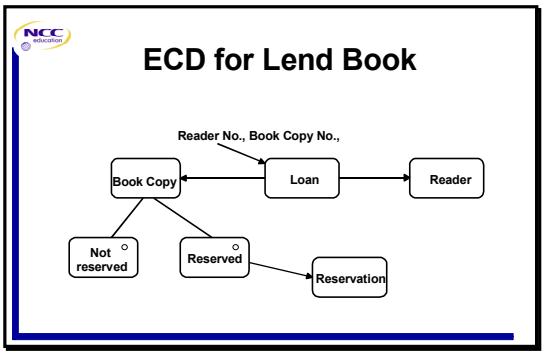
(20 minutes)

As discussed previously, ELHs correspond to columns in the Event/Entity Matrix (Entity Access Matrix). ECDs correspond to rows. It means that the Effect Correspondence Diagrams (ECDs) include all the entities affected by the particular event.

In some sense ECDs represent update navigation paths through the Logical Data Model (i.e. a network of entities). They show how a particular event *propagates* through the LDM.

Notation used in ECDs is based on Jackson notation but there are some differences (remember that Jackson diagrams are hierarchical diagrams while ECDs are *network* diagrams). Visual V8.3 shows an ECD for Lend Book (without conditions and operations which are allocated later).

V8. 3



This diagram shows that the event *Lend Book* is triggered by the arrival of data items Reader No. and Book Copy No. This event affects instances of Loan, Reader and Book Copy. If the Book Copy has been reserved, the corresponding instance of Reservation is also affected.

SSADM recommends the following steps for drawing ECDs (**Weaver, Goodland/Slater**):

- Draw a box for each entity affected by the event (i.e. for each entity in the corresponding row of the Event/Entity matrix). These boxes are called *effects*.
- Draw separate boxes for simultaneous effects (no need to discuss).
- Include optional effects (selections) if an event affects an entity in one or more mutually exclusive ways.
- Show iterations where an event affects several instances of the same entity.
- Identify effects which are in one-to-one correspondence, and connect the affected entities with an arrow.
- Add non-updated entities (i.e. entities needed for enquiries).
- Show the entry point and add input data under the entry point to the ECD.

We *finalise* ECDs by adding conditions and operations.

2.1 Allocating Conditions and Operations

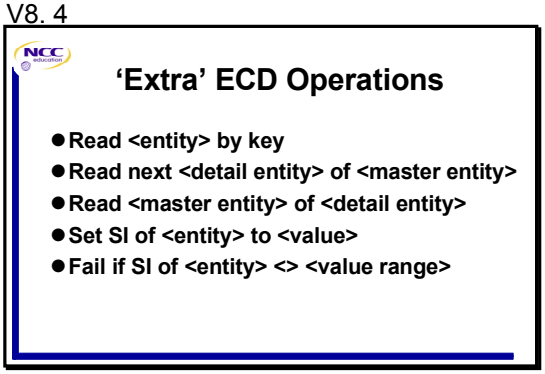
A condition is added to each selection (option) and iteration. The condition should specify the circumstances under which an option is to occur, or an iteration is to continue.

Operations are added to each entity box. In fact, most of the operations are carried forward from corresponding ELHs. For example, if entities EntA, EntB, and EntC *belong* to the ECD for event EvX, then the operations added to the EntA box (on the ECD) are ELH operations already allocated to the EvX box on the ELH for

EntA. Similarly operations already allocated to the EvX boxes on the ELHs for EntB and EntC are added (i.e. copied) to the EntB and EntC boxes on the ECD for EvX.

In addition to ELH operations, database operations are added to read and write entity occurrences, and integrity operations to check and set state indicator values.

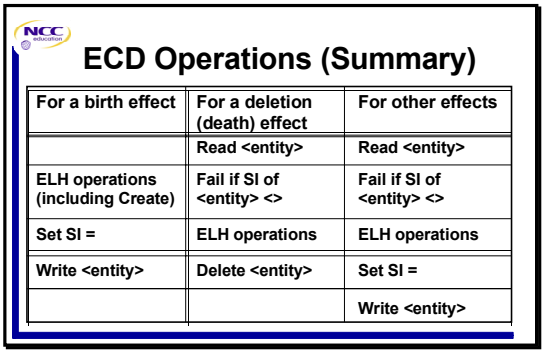
V8. 4



These extra operations are briefly presented below (see also Visual V8.4).

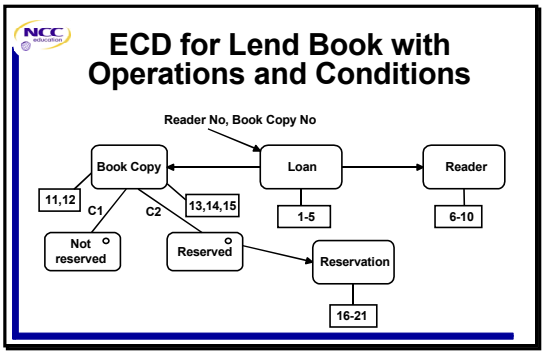
- *Read <entity type> by key* i.e. Read using the primary key.
- *Read next <detail> of <master> [via <relationship>]* i.e. Read the next occurrence of detail entity in the set.
- Read <master> of <detail> [via <relationship>
- Set SI of <entity> to <value> i.e. Set State Indicator.
- *Fail if State Indicator (SI) of <entity type> outside <value-range>*
- Report an error (and possibly quit processing) if the state indicator is outside a preset value range.

V8. 5



The table in Visual V8.5 gives a brief summary of the necessary operations that should be allocated to entities with a birth (i.e. create), death (i.e. delete), and other effects (modification in particular).

V8. 6



The final ECD (with conditions and operations) for the event Lend Book is shown in Visual V8.6.

Issue Handout 8.1.

3 Developing Enquiry Access Paths (EAPs)

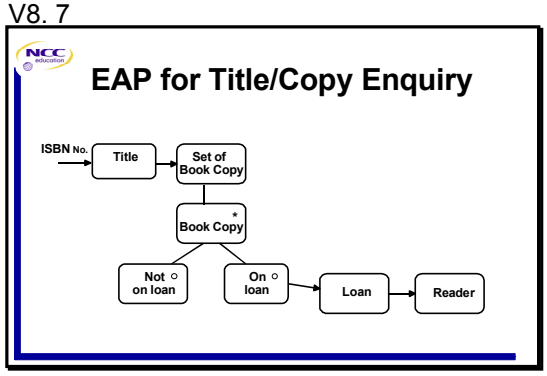
(20 minutes)

In fact EAPs do not belong to Entity/Event Modelling, but they are usually produced together with ECDs.

EAPs represent enquiry navigation paths through the LDM. An EAP shows the route through the LDM required for a particular enquiry function. EAP notation is very similar to ECD notation.

Visual V8.7 shows an EAP for the Title/Copy enquiry in Easybooks system (see Handout 5.3).

V8. 7



Visual V8. 7 shows that in order to satisfy this enquiry an instance of the specified Title (ISBN no.) should be 'visited' *first* followed by several Book Copies. For each instance of Book Copy on loan, the corresponding Loan and Reader instances must be accessed. The entry to the EAP is *via* Title.

SSADM recommends the following steps for drawing EAPs (**Weaver, Goodland/Slater**):

- Name the enquiry.
- Define the enquiry trigger.
- Identify entities required for access.
- Draw the extract of the LDM represented by these entities.
- Redraw this extract using Jackson notation (i.e. identify iterations and selections).
- Show the entry point and annotate the entry point with input data items.

We *finalise* EAPs by adding conditions and operations.

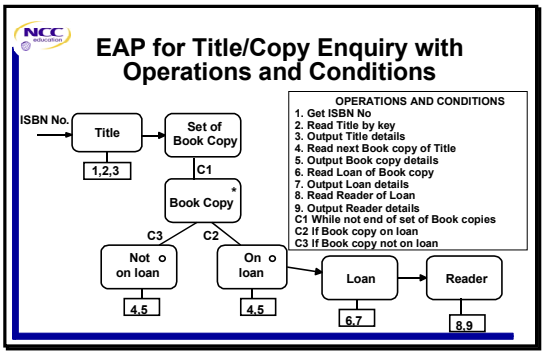
3.1 Allocating Conditions and Operations

Conditions are added to each selection (option) and iteration.

The operations which we add to EAPs are a subset of those allocated to ECDs. One Read operation is needed for each arrow on an EAP.

There are three allowable read operations (see also 2.1):

V8. 8



- Read <entity type > by key
- Read next <detail> of <master>
- Read <master> of <detail>

In addition to read operations we can also allocate operations to output data items (e.g. values of attributes) required by an enquiry, and operations to get inputs.

Operations should be added to each entity box (from left to right), in the order in which they will be *executed*.

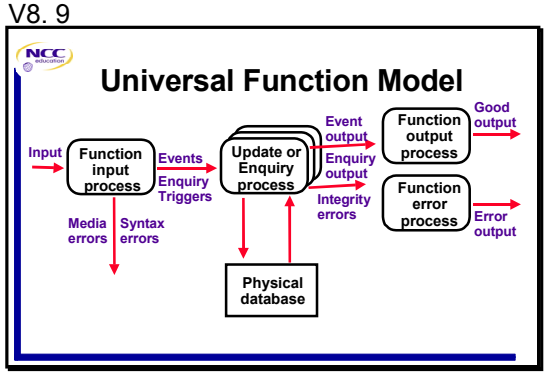
Visual V8.8 shows the final EAP for Title/Copy enquiry (i.e. with conditions and operations).

4 SSADM4+ – Logical Process Design

(20 minutes)

Logical Process Design is the main technique used in Stage 5 of SSADM. It comprises two (similar) sub-techniques : Update Process Modelling (used in Step 520 – Define Update Processes) and Enquiry Process Modelling (used in Step 530 – Define Enquiry Processes).

V8. 9



Both techniques (steps) produce a set of detailed implementation-independent specifications of update and enquiry processes.

As you remember, Functions have been introduced in Stage 3 (Step 330). They are the basic units of system processing. We have identified two types of functions - Update and Enquiry functions. Functions *consist* of many components (see Visual V8.9).

In Step 330 we have defined the following components of a function:

- Input, Valid Output (documented in I/O structures).
- Events and Enquiry Triggers, Function Definition Boundary (documented in Function Definitions).

Now in Stage 5 (Steps 520/530) we concentrate on Update and Enquiry Processes and their Outputs (i.e. Event and Enquiry Output and Integrity Errors).

I/O Processes and all Error Outputs are specified in Stage 6.

ELHs and ECDs in particular (developed in Step 360) give us a picture of the impact of events on the system's logical data model (logical database). In Step 520 we enhance this picture by adding further detail of how the system will apply and control the effects of each event.

The vehicle/technique used to specify detailed event processing is Update Process Modelling.

EAPs give us a picture of the navigation through the logical data model (logical database). Basically the EAP is a diagram which shows which entities should be accessed, and in which order, to satisfy an enquiry. In Step 530 we enhance this picture by adding further detail of how the system will satisfy the enquiries. The vehicle/technique used to specify detailed enquiry processing is Enquiry Process Modelling.

Logical Process Models can be used as actual program specifications in 4GL environments. In such environments, the translation from logical processing and databases to physical processing (programs) and databases is usually handled by the development software itself (for example, Application Generators). In a procedural (3GL) environment the process models will be translated into environment-specific program specifications during Physical Design (Stage 6).

Logical Process Models serve as:

- the basis for physical design and implementation of the new system;
- effective maintenance documentation once the system is up and running.

4.1 Logical Process Modelling (Logical Design of Update/Enquiry Processes)

Our main task now is to design processes which access the database.

The main objective of logical process design is to produce the detailed specification for each process (update or enquiry). This specification should be rigorous and unambiguous. Since any piece of processing (no matter how complex) can be precisely described using combinations of sequences, iterations, and selections (of actions), we are going to use Jackson-like notation again, this time to specify processing structure. The specific steps in designing processes are as follows:

- Carry forward ECDs (for update processes) or EAPs (for enquiry processes).
- Group effects on ECDs (or accesses on EAPs) which are in One-to-One correspondence.
- Convert ECDs (or EAPs) with grouped effects (or accesses) to Jackson-like structure (i.e. Processing Structure).
- Allocate Operations and Conditions to Processing Structure.

5 Enquiry Process Modelling

(10 minutes)

5.1 Carry Forward EAPs

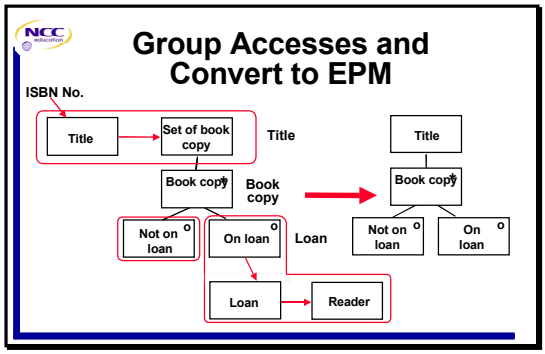
We will use the EAP for the Title/Book copy enquiry (see 3) to illustrate how Enquiry Process Models (EPMs) are created.

5.2 Group Accesses in One-to-One Correspondence

All accesses (i.e. entity boxes) connected via arrows are in one-to-one correspondence. We want to group together processing that can be carried out as a single module. Boxes linked with arrows form a group. This means that the entities can be accessed in any order as a group. All such groups should be given a meaningful name. Stand alone boxes will form a group of their own. Visual V8.10 shows groups.

5.3 Convert to Jackson-like Structure (i.e. Processing Structure)

V8. 10



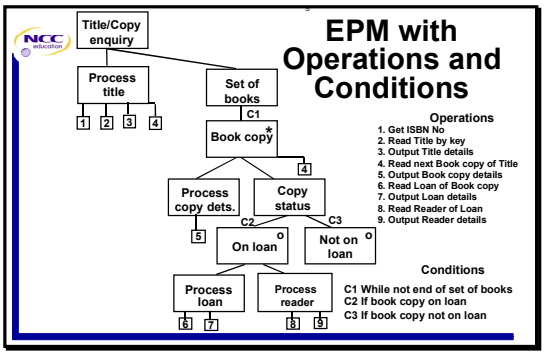
This is again a simple procedure. Each processing group becomes a box on the Jackson-like structure which represents the Processing structure for this enquiry (see Visual V8.10).

The resulting structure should adhere to the usual rules regarding combinations of box types, so a few new boxes may need to be added.

5.4 Allocate Operations and Conditions

Operations are carried across from the Enquiry Access Path to the corresponding Processing structure i.e. Enquiry Process Model (EPM).

V8. 11



The resulting structure i.e. EPM should adhere to the usual rules regarding combinations of box types, so a few new boxes may need to be added.

Visual V8.11 shows the EPM for the Title/Book copy enquiry with allocated operations and conditions.

6 Update Process Modelling

(10 minutes)

6.1 Carry Forward ECDs

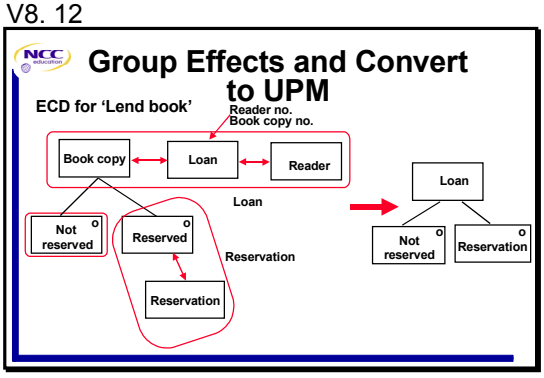
We will use the ECD for Lend Book (see 2) to illustrate how Update Process Models (UPMs) are created.

6.2 Group Effects on ECD which are in One-to-One Correspondence

We want to group together processing that can be carried out as a single module. Boxes linked with arrows form a group. All such groups should be given a meaningful name. Stand alone boxes will form a group of their own. Visual V8.12 shows groups.

6.3 Convert to Jackson-like (Processing) Structure

V8. 12



This is again a simple procedure. Each processing group becomes a box on the Jackson-like structure which represents the Processing structure for this updating (see Visual V8.12).

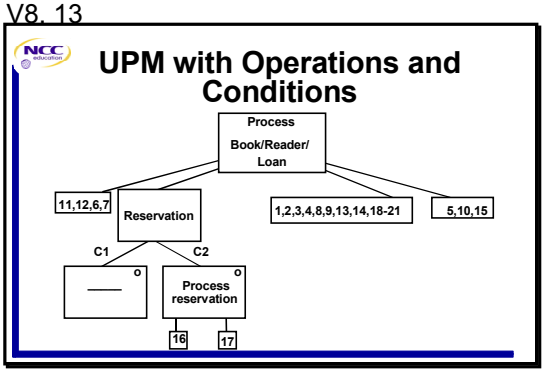
The resulting structure should adhere to the usual rules regarding combinations of box types, so a few new boxes may need to be added.

6.4 Allocate Operations and Conditions to the Processing Structure

Operations are carried across from the Effect Correspondence Diagram to the corresponding Processing structure i.e. Update Process Model (UPM).

The resulting structure i.e. UPM should adhere to the usual rules regarding combinations of box types, so a few new boxes may need to be added.

V8. 13



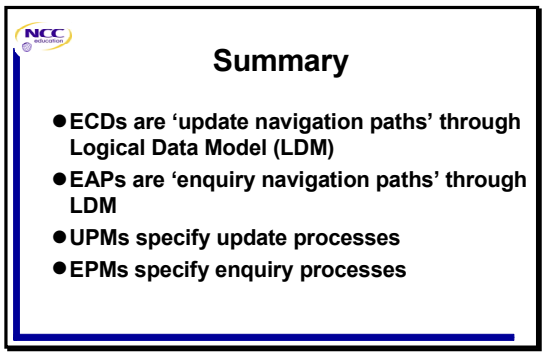
Visual V8.13 shows the UPM for the Lend Book with allocated operations and conditions.

As discussed in Section 2, the main classes/types of update operations are *reading* operations, *SI checking* operations, *ELH* operations (main updating operations), *SI setting* operations, and *writing* operations.

Issue Handouts 8.2.

7 Summary

V8. 14



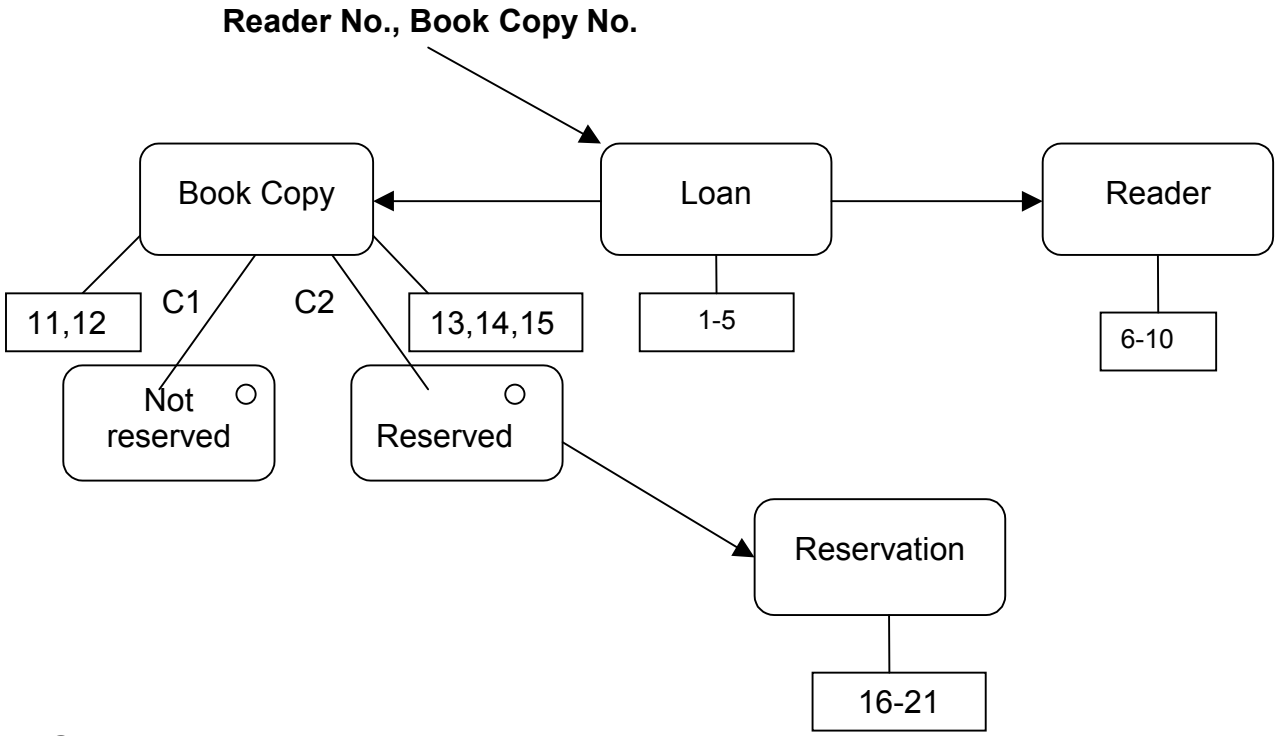
(5 minutes)

Effect Correspondence Diagrams, Enquiry Access Paths and the Logical Process Modelling techniques have been discussed. ECDs represent *update navigation paths* through the Logical Data Model. An ECD shows all of the entities affected by an event. EAPs are *enquiry navigation paths*.

An EAP shows all entities accessed (*visited*) by an enquiry. Update Process Models are derived from ECDs. A UPM is a Jackson diagram that specifies processing required to access and update data in response to an event. Enquiry Process Models are derived from EAPs. They are similar to UPMs.

Issue Handout 8.3 for use in the workshop part of the session. Ask students to derive an Update Process Model (UPM) from the ECD for Lend Book event, shown in this handout.

Handout 8.1 – ECD for Lend Book with Operations and Conditions



Operations

Loan operations
1 Create Loan
2 Tie Loan to Reader
3 Tie Loan to Book Copy
4 Set SI of Loan to ...
5 Write Loan

Reader operations
6 Read Reader by Key
7 Fail if SI of Reader <> ...
8 Set NoOfLoans to NoOfLoans +1
9 Set SI of Reader to ...
10 Write reader

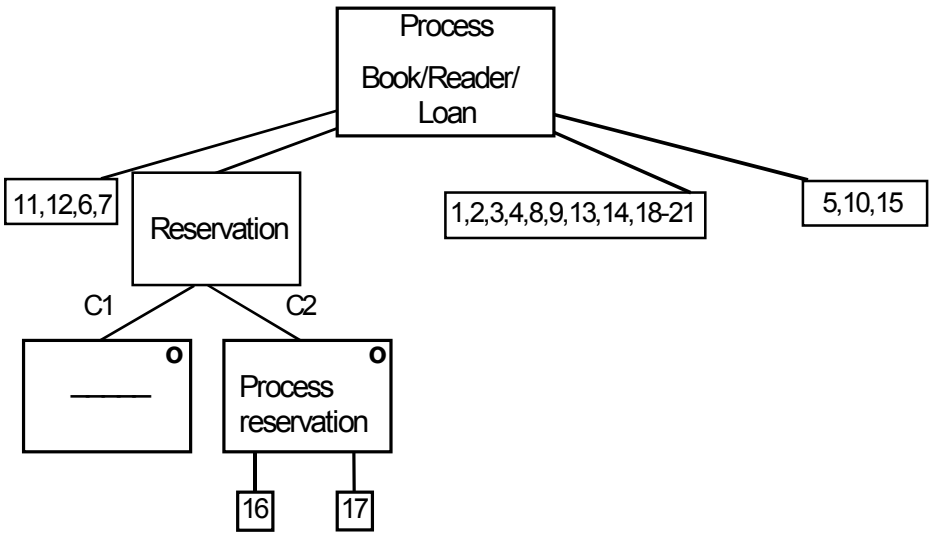
Book Copy operations
11 Read Book Copy by Key
12 Fail if SI of Book Copy <> ..
13 Set NoOfLoans to NoOfLoans + 1
14 Set SI of Book Copy to ....
15 Write Book Copy

Reservation operations
16 Read next Reservation of Book Copy
17 Fail if SI of Reservation <> ...
18 Cut Reservation from Book Copy
19 Cut Reservation from Title
20 Cut Reservation from Reader
21 Delete Reservation

Conditions

C1 If Book Copy not reserved
C2 If Book Copy reserved

Handout 8.2 – Update Process Model for Lend Book with Operations and Conditions



Operations

Loan operations
6 Create Loan
7 Tie Loan to Reader
8 Tie Loan to Book Copy
9 Set SI of Loan to ...
10 Write Loan

Reader operations
11 Read Reader by Key
12 Fail if SI of Reader <> ...
13 Set NoOfLoans to NoOfLoans +1
14 Set SI of Reader to ...
15 Write reader

Book Copy operations
16 Read Book Copy by Key
17 Fail if SI of Book Copy <> ..
18 Set NoOfLoans to NoOfLoans + 1
19 Set SI of Book Copy to ....
20 Write Book Copy

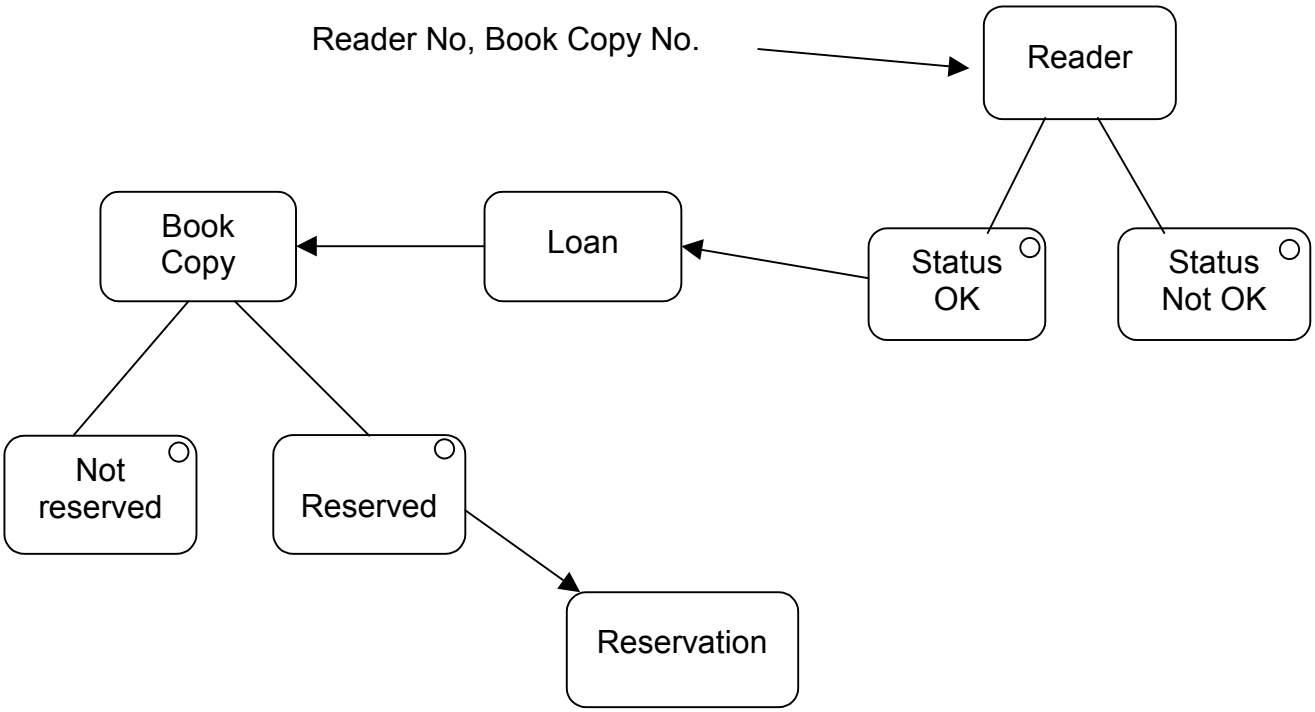
Reservation operations
22 Read next Reservation of Book Copy
23 Fail if SI of Reservation <> ...
24 Cut Reservation from Book Copy
25 Cut Reservation from Title
26 Cut Reservation from Reader
27 Delete Reservation

Conditions

C1 If Book Copy not reserved
C2 If Book Copy reserved

Handout 8.3 – ECD for Lend Book (more 'sophisticated')

This ECD shows that a *full* processing of the event takes place only if the Reader's status is OK (i.e. no overdue loans and number of books on loan<10).



Software Engineering Methods

Session 8
SSADM4+ - Entity/Event Modelling - 2 and Logical Process Design

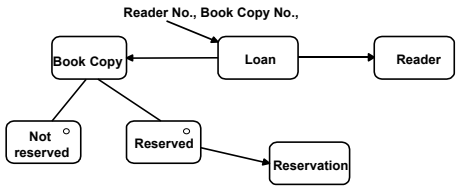
© NCC Education Limited, 2001

Contents

- Developing ECDs
- Developing EAPs
- SSADM 4+ Logical Process Design
- Enquiry Process Modelling
- Update Process Modelling

© NCC Education Limited, 2001

ECD for Lend Book



© NCC Education Limited, 2001

'Extra' ECD Operations

- Read <entity> by key
- Read next <detail entity> of <master entity>
- Read <master entity> of <detail entity>
- Set SI of <entity> to <value>
- Fail if SI of <entity> <> <value range>

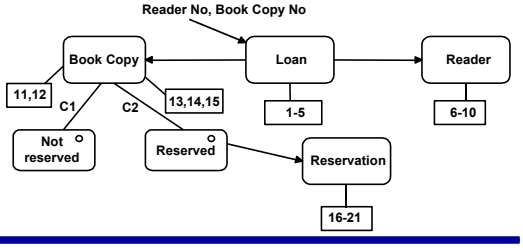
© NCC Education Limited, 2001

ECD Operations (Summary)

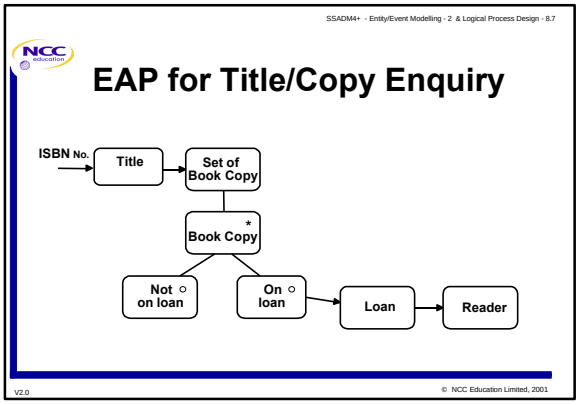
For a birth effect	For a deletion (death) effect	For other effects
	Read <entity>	Read <entity>
ELH operations (including Create)	Fail if SI of <entity> <>	Fail if SI of <entity> <>
Set SI =	ELH operations	ELH operations
Write <entity>	Delete <entity>	Set SI =
		Write <entity>

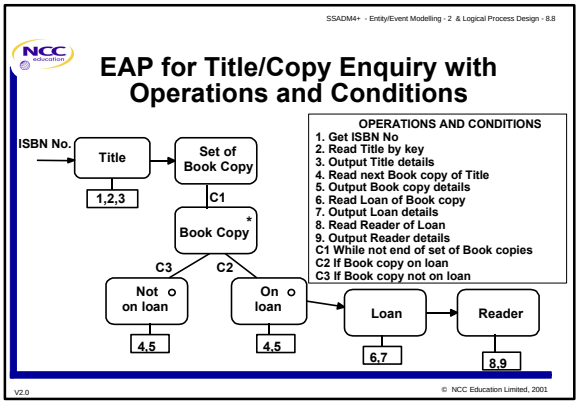
© NCC Education Limited, 2001

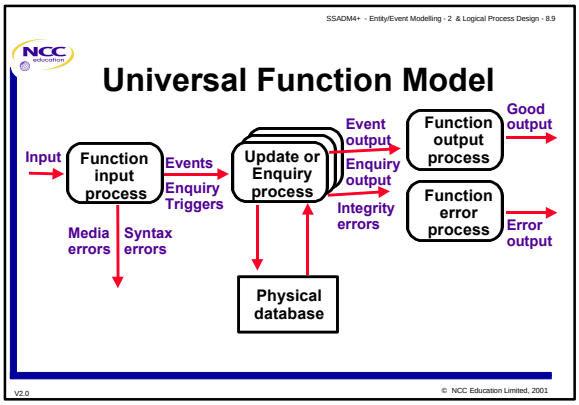
ECD for Lend Book with Operations and Conditions

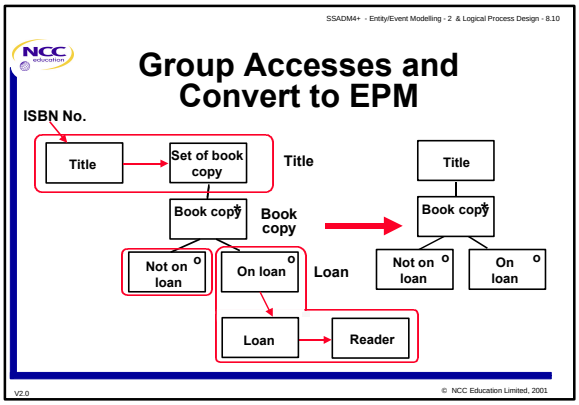


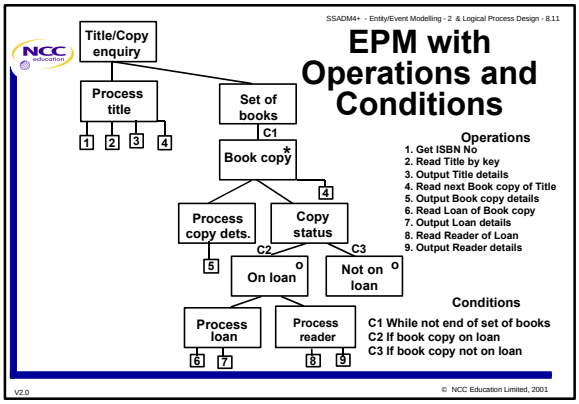
© NCC Education Limited, 2001

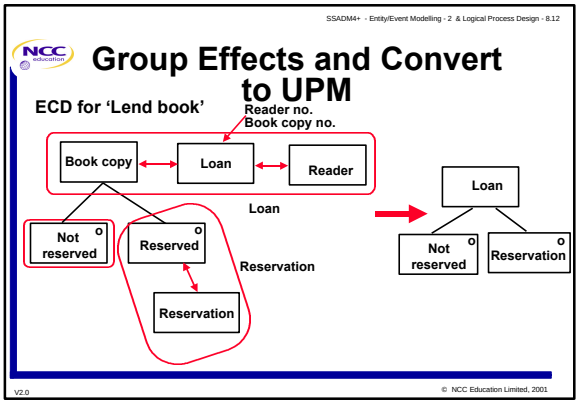

SSADM4+ - Entity/Event Modelling - 2 & Logical Process Design - 8.7
EAP for Title/Copy Enquiry
ISBN No.
Title
Set of Book Copy
Book Copy
Not on loan
On loan
Loan
Reader
V2.0
© NCC Education Limited, 2001


SSADM4+ - Entity/Event Modelling - 2 & Logical Process Design - 8.8
EAP for Title/Copy Enquiry with Operations and Conditions
OPERATIONS AND CONDITIONS
1. Get ISBN No
2. Read Title by key
3. Output Title details
4. Read next Book copy of Title
5. Output Book copy details
6. Read Loan of Book copy
7. Output Loan details
8. Read Reader of Loan
9. Output Reader details
C1 While not end of set of Book copies
C2 If Book copy on loan
C3 If Book copy not on loan
ISBN No.
Title
Set of Book Copy
1,2,3
C1
Book Copy
C3
C2
Not on loan
On loan
Loan
Reader
4,5
4,5
6,7
8,9
V2.0
© NCC Education Limited, 2001

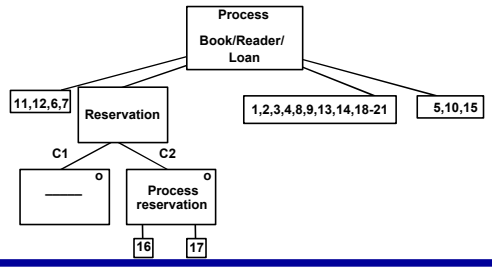

SSADM4+ - Entity/Event Modelling - 2 & Logical Process Design - 8.9
Universal Function Model
Input
Function input process
Events
Enquiry Triggers
Update or Enquiry process
Event output
Enquiry output
Integrity errors
Function output process
Good output
Function error process
Error output
Media errors
Syntax errors
Physical database
V2.0
© NCC Education Limited, 2001


SSADM4+ - Entity/Event Modelling - 2 & Logical Process Design - 8.10
Group Accesses and Convert to EPM
ISBN No.
Title
Set of book copy
Book copy
Not on loan
On loan
Loan
Reader
Title
Book copy
Not on loan
On loan
V2.0
© NCC Education Limited, 2001


SSADM4+ - Entity/Event Modelling - 2 & Logical Process Design - 8.11
EPM with Operations and Conditions
Title/Copy enquiry
Process title
Set of books
Book copy
Process copy dets.
Copy status
On loan
Not on loan
Process loan
Process reader
Operations
1. Get ISBN No
2. Read Title by key
3. Output Title details
4. Read next Book copy of Title
5. Output Book copy details
6. Read Loan of Book copy
7. Output Loan details
8. Read Reader of Loan
9. Output Reader details
Conditions
C1 While not end of set of books
C2 If book copy on loan
C3 If book copy not on loan
V2.0
© NCC Education Limited, 2001


SSADM4+ - Entity/Event Modelling - 2 & Logical Process Design - 8.12
Group Effects and Convert to UPM
ECD for 'Lend book'
Reader no.
Book copy no.
Book copy
Loan
Reader
Not reserved
Reserved
Reservation
Loan
Not reserved
Reservation
V2.0
© NCC Education Limited, 2001

UPM with Operations and Conditions



© NCC Education Limited, 2001

Summary

- ECDs are 'update navigation paths' through Logical Data Model (LDM)
- EAPs are 'enquiry navigation paths' through LDM
- UPMs specify update processes
- EPMs specify enquiry processes

© NCC Education Limited, 2001

Software Engineering Methods

Session 9

Introduction to JSD

© NCC Education Limited, 2001

Session 9 – Modification History

Revision	Date	Revision Description
V1.0	19 August, 1997	For issue
V2.0	18 September 2001	Updated for new syllabus V2/2001.

© NCC Education Limited, 2001

All Rights Reserved

The copyright in this document is vested in NCC Education Limited. The document must not be reproduced by any means, in whole or in part, or used for manufacturing purposes, except with the prior written permission of NCC Education Limited and then only on condition that this notice is included in any such reproduction.

Information contained in this document is believed to be accurate at the time of publication, but no liability whatsoever can be accepted by NCC Education Limited arising out of any use made of this information.

Trademarks

NCC Education acknowledge that the trademarks and registered trademarks of products mentioned in this material are held by the companies producing them. Use of a term in this material should not be regarded as affecting the validity of any trademark or service mark.

Copyright of any screen captures in this material are the property of the software's manufacturer.

This material may contain some clipart, which is copyright to the Corel Corporation.

Session 9

Introduction to JSD

Lecturer Guide

Session Specification

Duration

Allow approximately 1 hour 30 minutes for the lecture part of this session. There should also be:

- tutorial (approximately 1 hour 30 minutes);
- workshop (approximately 1 hour).

Objectives

By the end of this session the participant will:

- understand the structure of JSD;
- understand the main concepts used in JSD;
- be familiar with JSD techniques and tools.

Lecturer

The lecturer should have a good knowledge and experience of JSD.

Method of Presentation

This session should be a lecture based on the material provided.

Material List

Visuals

V9. 1 Introduction
V9. 2 Contents
V9. 3 Jackson's View of a System
V9. 4 PSD – Sequence
V9. 5 PSD – Iteration
V9. 6 PSD – Selection
V9. 7 PSD for Bank Account
V9. 8 Structure Text for Account
V9. 9 Datastreams and State Vector Connections
V9.10 Notation to Specify Multiple Instances of a Single Process Type
V9. 11 Structure of JSD
V9.12 JSD System – Logical View
V9.13 Summary

Handouts

A Handout is provided showing all the visuals presented during this session.

Equipment Required

The following equipment is required:

- overhead projector and screen;
- flipchart stand, flipchart and broad-tipped felt pens.

Bibliography

L Ingevaldsson, *Software Engineering Fundamentals – the Jackson Approach*, Chartwell-Bratt Limited, 1990

M A Jackson, *System Development*, Prentice Hall, 1983

A Sutcliffe, *Jackson System Development*, Prentice Hall, 1988

Session Summary

1 Introduction 9-1

1.1 Summary of Topics to be Covered 9-1

2 Introduction to JSD 9-1

3 Concepts 9-2

3.1 Entities and Actions 9-2
3.1.1 Tools to Specify Entities 9-3
3.2 Long-Running (or Model) Processes 9-4
3.3 Short-Running Processes 9-5
3.3.1 Tools to Specify Short-Running Processes 9-5
3.4 Inter-Process Communications 9-5
3.4.1 Tool to Specify Inter-Process Communications - SSD 9-5

4 Overview of JSD 9-7

5 Summary 9-8

Visuals Handout

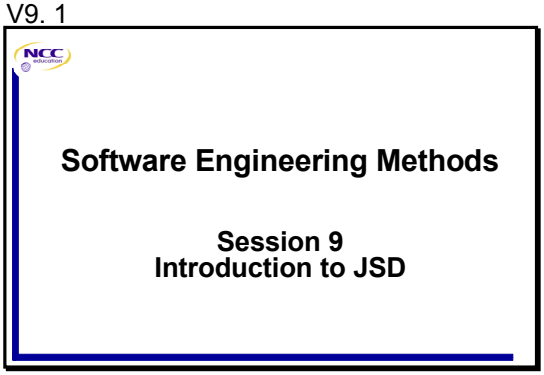
Session 9

Introduction to JSD

1 Introduction

(5 minutes)

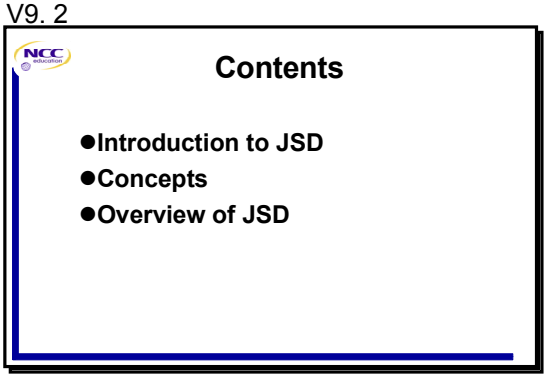
V9. 1



Inform students that a handout containing a full set of visuals will be provided to them at the end of this lecture.

1.1 Summary of Topics to be Covered

V9. 2



The topics detailed in the visual will be discussed during this session.

2 Introduction to JSD

(5 minutes)

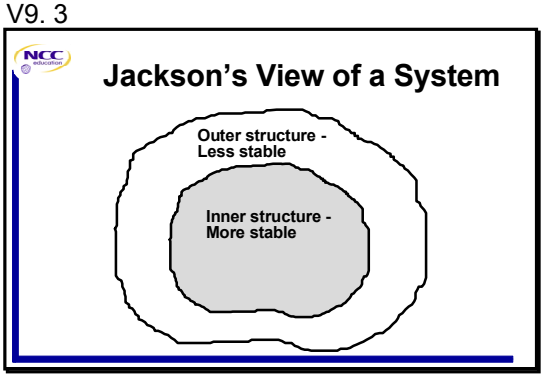
Jackson System Development (JSD) was launched in 1983 in **Michael Jackson**'s book *System Development*, and gained acceptance in many organisations.

JSD is a system specification and design method covering the system's life cycle from analysis to implementation.

It differs in many respects from other methods in the *structured* school. It places more emphasis on *dynamic modelling* than other methods and relatively less

emphasis on process and data modelling. It uses its own techniques and tools (quite different from techniques/tools used in other methods).

V9. 3



The main concept behind JSD is its drive to model the system's *inner* structure, not its *surface manifestations*, usually identified as functions. This *inner* structure will constitute the core of the system. **Jackson** argues that such a core is more stable than functions are (and that systems based on such *inner* models are easier to maintain!).

Functions such as, for example, to validate a customer's order or to produce a sales report, represent some aspects of a system's functionality but tell us little about the underlying structure.

3 Concepts

(60 minutes)

3.1 Entities and Actions

The main components of the system's inner structure are entities.

An *entity* is an object of interest in the system which will undergo or cause change during the system's activity.

Entities respond to events in the real world. The concept of event in JSD is similar to SSADM's (see Sessions 6 and 7).

Entities are composed of *actions*. Actions correspond to events and can be viewed as responses to events in the real world. Actions *within* an entity are specified in a time-ordered sequence.

Actions are the fundamental building blocks of the JSD model. Entities are *bigger* blocks.

Examples of entities and actions:

- Entity – Order; Actions – Place, Confirm, Allocate, Despatch.
- Entity – Bank Account; Actions – Open, Withdraw, Deposit, Close.
- Entity – Reader; Actions – Register, Borrow, Return, Deregister, *etc*.

Note that JSD entities are dynamic and active (they are composed of actions!). They can be viewed as processes whose executions start with the first event/action. The execution of an entity process continues through *middle life* events/actions and is terminated by the last event/action.

SSADM's entities (like database entities) are static. They do not perform actions. Actions are performed by update processes!

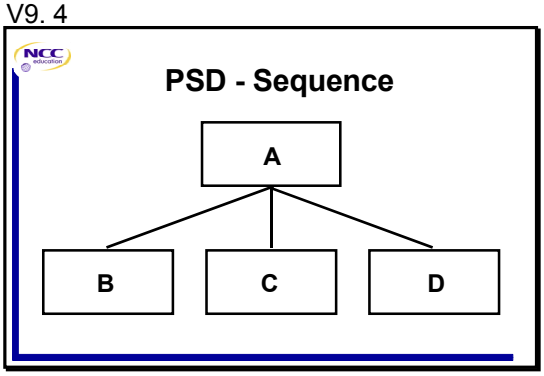
JSD entities are therefore more similar to objects in Object Oriented methods than to database entities.

3.1.1 Tools to Specify Entities

The main tools for specifying entities and actions are Process Structure Diagrams and Structure text.

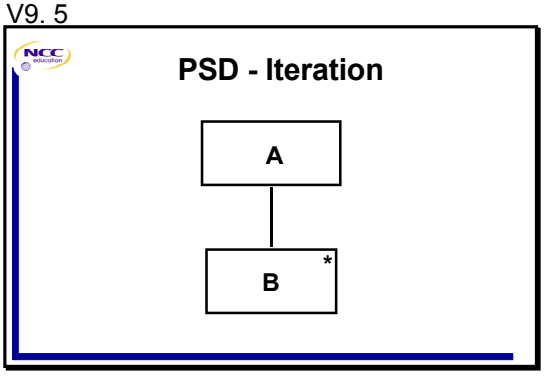
3.1.1.1 Process Structure Diagrams (PSDs)

V9. 4



PSDs are similar to ELHs (in SSADM). They show the order of actions for each entity type.

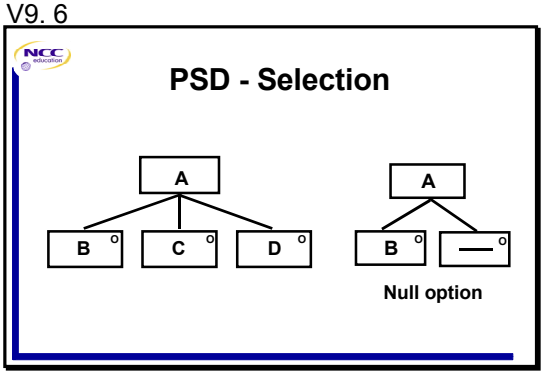
V9. 5



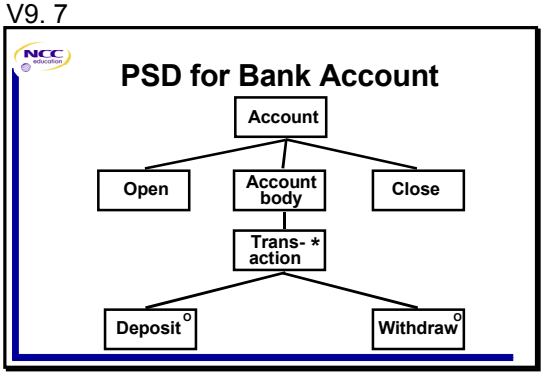
There are three basic diagrams to represent three different *configurations* of actions (structured constructs):

- sequence;
- iteration;
- selection.

V9. 6



V9. 7

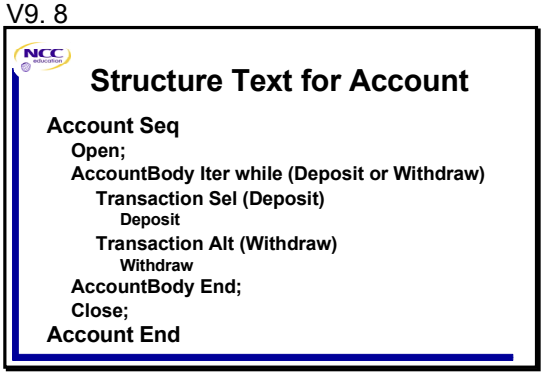


A PSD for entity Bank Account is as shown in Visual V9.7.

PSDs are also used to specify functions (functional processes).

3.1.1.2 Structure Text (Pseudocode)

V9. 8



Structure text is a textual form of PSDs. It is an alternative form of specification for entities and functional processes. It uses indentation to show structure, and has reserved words for different structured constructs (Seq, Sel, Iter).

A Structure text for entity Bank Account is as shown in Visual V9.8.

3.2 Long-Running (or Model) Processes

JSD entities can be viewed as *long-running* processes *i.e.* with long life histories. For example a typical instance/occurrence of the Bank Account entity lasts a few years, i.e. its execution is very long. It does not mean that JSD processes are in a state of perpetual execution. Most of the time they wait for the next event (therefore action) to occur. In some sense the actual execution is represented by actions separated by a waiting state. Entity (i.e. long-running) processes are also called *model processes*.

Each entity (i.e. model process) is viewed as a type which may have many instances/occurrences. For example, there may be several instances of the entity

Reader in a library system, several instances of the entity Bank Account in a banking system, etc.

Long-running processes (i.e. entities) belong to the *inner structure*/core of a system (see V9.3).

3.3 Short-Running Processes

The *outer structure* of a system (see V9.3) includes enquiry functions i.e. *output* processes, and so called filters i.e. *input* processes.

Enquiry functions read user requests, retrieve information stored in long-running processes (entities) and display output.

The main tasks of input processes i.e. filters is to provide the user interface and to validate input data (to *protect* long-running processes/entities from incorrect input).

Execution times of enquiries and filters are very short compared with the *execution* of long-running processes. That is why they are called short-running processes.

3.3.1 Tools to Specify Short-Running Processes

Short-running processes can be specified using Process Specification Diagrams (PSDs) and/or Structure Text (pseudocode) (see 3.1.1).

3.4 Inter-Process Communications

JSD views a system as a set of concurrent (long and short running) processes. These processes communicate with each other and with the outside world (users other systems). JSD provides the two classes of interprocess communication:

- datastream connections;
- state vector inspections.

System Specification Diagrams (SSDs) are used to show a network of the interconnected processes.

3.4.1 Tool to Specify Inter-Process Communications – SSD

SSDs consist of three components:

- *Processes* (Shown as rectangular boxes for both model and function processes).
- *Datastreams*. A datastream is a message channel with a FIFO (First In First Out) buffer. Each message produced/written by an upstream process (or a user) must be eventually read/consumed by a downstream process. The

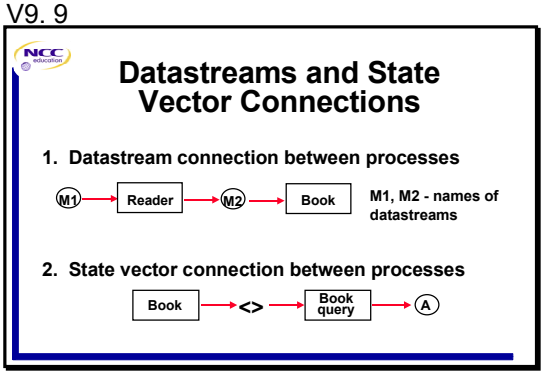
downstream process is blocked if there are no messages in a buffer. Datastreams are represented as arrows with circles. The name of the datastream is placed in the circle.

There are three types of messages: *event* messages, *data* messages, and *control* messages.

- Event messages *trigger* corresponding actions and carry input data. Datastreams pass event messages to long-running/entity processes that consist of corresponding actions.
- Data messages consist of data only. Usually they are *produced* by enquiry functions i.e. output processes. They flow via datastreams from enquiries to users.
- Control messages are used in real-time control systems.

- *State vector* inspections are the alternative process connections. The state vector is a collection of the entity attributes (or data items owned by the model process). The state vector of one process can be inspected by another process. This is of course a *read only* connection.

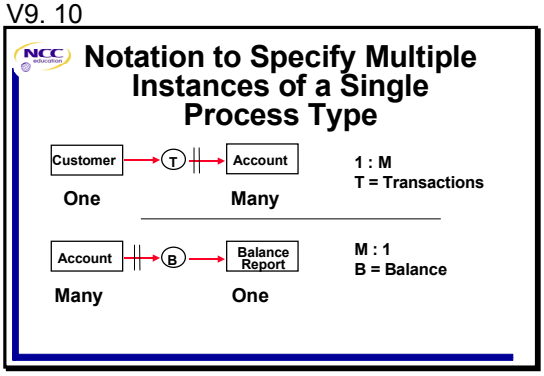
V9. 9



The SSD shows the sources and destinations of datastreams and which processes are inspecting other processes' state vectors.

Connections can be marked to show multiple instances of a single process type at either end of the connection.

V9. 10

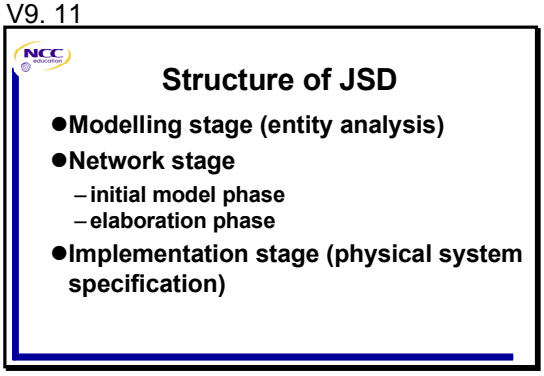


State vector connections are usually used by *enquiry* processes (to obtain information stored in model processes for reports, displays, etc.).

4 Overview of JSD

(15 minutes)

V9. 11



JSD divides the system development process into three main stages. Each stage has a set of activities with clear start and end points. This facilitates project control, as deliverables can be defined for each stage.

- *Modelling stage.* The real world problem is described and the important entities and actions within the system are identified (PSDs are used to describe entity life histories).

- *Network stage.* The system is developed gradually as a series of subsystems. First the major model/long-running processes (entities) are taken from the modelling stage and input and outputs are added. Some of these processes are connected (via their inputs and outputs) to form the core (inner structure) of the system.

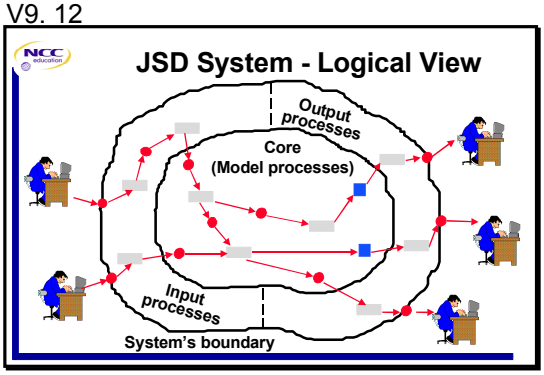
  This is followed by the analysis of the output subsystem which provides information. Output processes/functions are added and connected with the model/long-running processes *via* datastreams or state vector inspections.

  Next, the input subsystem which handles the user interface and validation is specified. Input processes/functions (i.e. filters) are added and connected with model/long-running processes *via* datastreams.

  So the entire system is specified as a network of communicating concurrent processes. This network consists of three parts/subsystems:

  – the input subsystem;
  – the core;
  – the output subsystem.

V9. 12

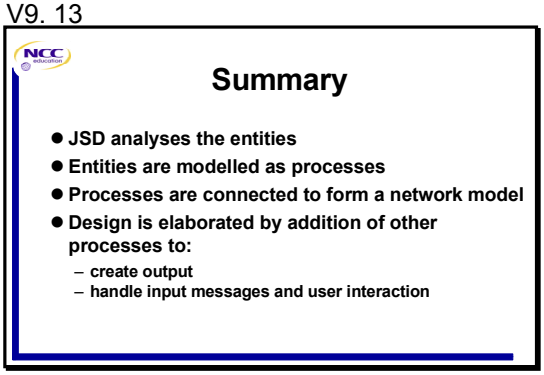


- *Implementation stage.* In this stage the logical specification, represented by a network of concurrently communicating processes, is transformed into a sequential design using the technique of scheduling. This is followed by further detailed design (using JSP) and coding.

5 Summary

(5 minutes)

V9. 13



JSD begins by analysing the major objects in the problem domain/space; the entities. The entities are modelled as long-running (or model) processes. Then these processes are connected together to create a network model of the system. The design is elaborated by addition of other processes to create output, and to handle input messages and user interaction.

"Physical design considerations are delayed to the last stage in JSD. Until then no assumptions are made about the machine the system is to run on and the logical view of the system is one of many concurrent communicating processes. In the last stage this view is transformed into a physical design, a step referred to as *implementation* in JSD. Design detail may be added in earlier stages; however, the remaining specification details are added during implementation before coding can begin." (**Sutcliffe**)

Introduction to JSD - 9.1

Software Engineering Methods

Session 9
Introduction to JSD

V2.0 © NCC Education Limited, 2001

Introduction to JSD - 9.2

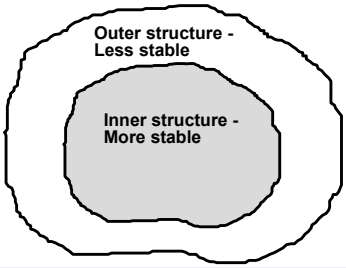
Contents

- Introduction to JSD
- Concepts
- Overview of JSD

V2.0 © NCC Education Limited, 2001

Introduction to JSD - 9.3

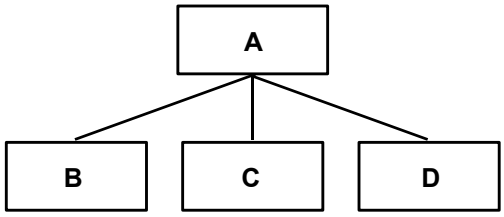
Jackson's View of a System



V2.0 © NCC Education Limited, 2001

Introduction to JSD - 9.4

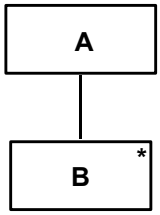
PSD - Sequence



V2.0 © NCC Education Limited, 2001

Introduction to JSD - 9.5

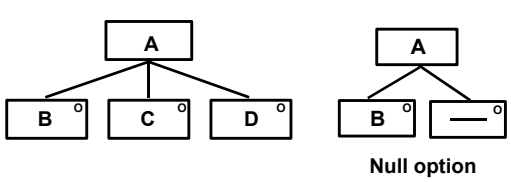
PSD - Iteration



V2.0 © NCC Education Limited, 2001

Introduction to JSD - 9.6

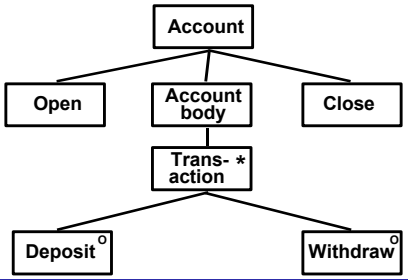
PSD - Selection



V2.0 © NCC Education Limited, 2001

Introduction to JSD - 9.7

PSD for Bank Account



V2.0 © NCC Education Limited, 2001

Introduction to JSD - 9.8

Structure Text for Account

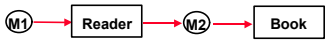
```
Account Seq
   Open;
   AccountBody Iter while (Deposit or Withdraw)
      Transaction Sel (Deposit)
         Deposit
      Transaction Alt (Withdraw)
         Withdraw
   AccountBody End;
   Close;
Account End
```

V2.0 © NCC Education Limited, 2001

Introduction to JSD - 9.9

Datastreams and State Vector Connections

1. Datastream connection between processes



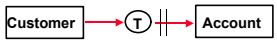
M1, M2 - names of datastreams

2. State vector connection between processes

V2.0 © NCC Education Limited, 2001

Introduction to JSD - 9.10

Notation to Specify Multiple Instances of a Single Process Type



One Many

1 : M
T = Transactions

Many One

M : 1
B = Balance

V2.0 © NCC Education Limited, 2001

Introduction to JSD - 9.11

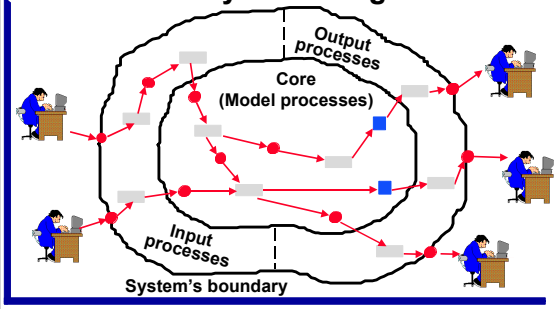
Structure of JSD

- Modelling stage (entity analysis)
- Network stage
  - initial model phase
  - elaboration phase
- Implementation stage (physical system specification)

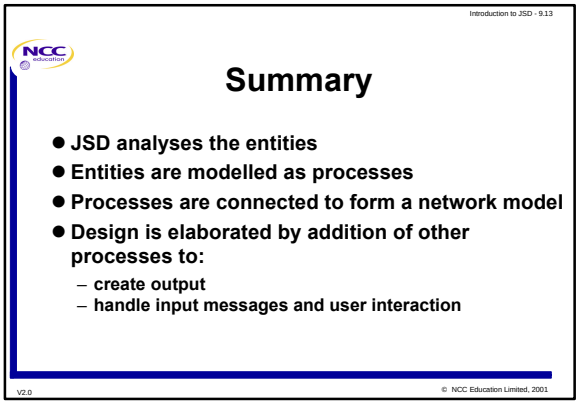
V2.0 © NCC Education Limited, 2001

Introduction to JSD - 9.12

JSD System - Logical View



V2.0 © NCC Education Limited, 2001


Introduction to JSD - 9.13
Summary
JSD analyses the entities
Entities are modelled as processes
Processes are connected to form a network model
Design is elaborated by addition of other processes to:
create output
handle input messages and user interaction
V2.0
© NCC Education Limited, 2001

Software Engineering Methods

Session 10

Object Oriented Methods and Techniques

© NCC Education Limited, 2001

Session 10 – Modification History

Revision	Date	Revision Description
V1.0	19 August, 1997	For issue
V2.0	18 September 2001	Updated for new syllabus V2/2001.

© NCC Education Limited, 2001

All Rights Reserved

The copyright in this document is vested in NCC Education Limited. The document must not be reproduced by any means, in whole or in part, or used for manufacturing purposes, except with the prior written permission of NCC Education Limited and then only on condition that this notice is included in any such reproduction.

Information contained in this document is believed to be accurate at the time of publication, but no liability whatsoever can be accepted by NCC Education Limited arising out of any use made of this information.

Trademarks

NCC Education acknowledge that the trademarks and registered trademarks of products mentioned in this material are held by the companies producing them. Use of a term in this material should not be regarded as affecting the validity of any trademark or service mark.

Copyright of any screen captures in this material are the property of the software's manufacturer.

This material may contain some clipart, which is copyright to the Corel Corporation.

Session 10 Object Oriented Methods and Techniques Lecturer Guide

Session Specification

Duration

Allow approximately 1 hour 45 minutes for the lecture part of this session. There should also be:

- tutorial (approximately 1 hour);
- workshop (approximately 1 hour).

Objectives

By the end of this session the participant will:

- be familiar with the main object oriented concepts;
- be familiar with the main UML techniques;
- understand a simple object-oriented development process.

Lecturer

The lecturer should be an experienced software engineer with a sound knowledge of object oriented techniques.

Method of Presentation

This session should be a lecture based on the material provided.

Material List

Visuals

V10. 1 Introduction
V10. 2 Contents
V10. 3 Main Concepts in Object Orientation
V10. 4 Use Case Diagram
V10. 5 Class Diagram – Class
V10. 6 Class Diagram – Associations
V10. 7 Class Diagram – Aggregation
V10. 8 Class Diagram – Generalisation/Inheritance
V10. 9 Sequence Diagram
V10.10 Collaboration Diagram
V10.11 State Diagram
V12. 12 Object Oriented 'View' of a System
V10.13 Summary

Handouts

A Handout is provided showing all the visuals presented during this session.

Equipment Required

The following equipment is required:

- overhead projector and screen;
- flipchart stand, flipchart and broad-tipped felt pens.

Bibliography

S Bennett, S McRobb, R Farmer, *Object Oriented Systems Analysis and Design using UML,* McGraw-Hill, 1999.

P Stevens, R Pooley, *Using UML – Software Engineering with Objects and Components,* Pearson Education Ltd, 2000.

Session Summary

1 Introduction 10-1

1.1 Summary of Topics to be Covered 10-1

2 Object Oriented Methods 10-1

3 Basic Concepts 10-2

4 Overview of UML 10-3

4.1 Use Case Model/Diagram 10-3
4.2 Class Model/Diagram 10-4
4.3 Interaction Models/Diagrams 10-5
4.4 State Models/Diagrams 10-6

5 A Simple Object Oriented Development Process 10-6

6 Summary 10-7

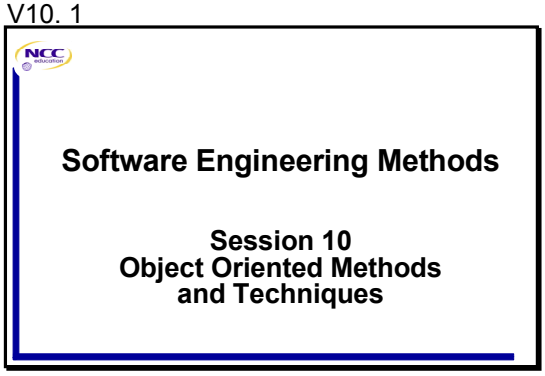
Visuals Handout

Session 10 Object Oriented Methods and Techniques

1 Introduction

(5 minutes)

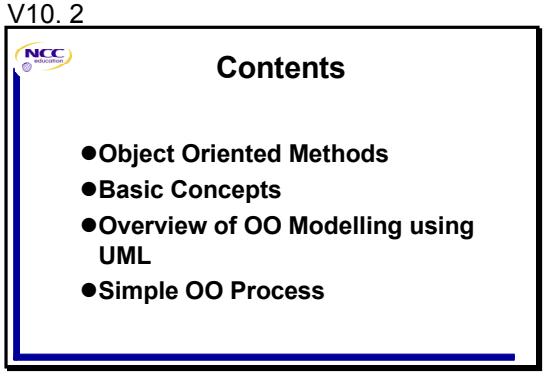
V10. 1



Inform students that a handout containing a full set of visuals will be provided to them at the end of this lecture.

1.1 Summary of Topics to be Covered

V10. 2



The topics detailed in the visual will be discussed during this session.

2 Object Oriented Methods

(5 minutes)

Object Orientation is still a *hot* subject.

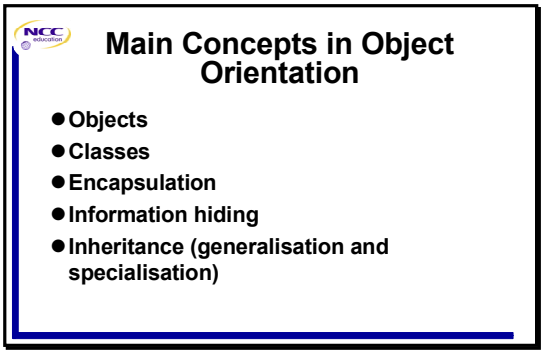
The OO *revolution* started in the programming area. *Simula 67* was the main predecessor of modern OO programming languages. As it was the beginning of the *structured* era, only a few people fully appreciated the importance of Simula 67 at that time. Simula 67 was followed by *Smalltalk* (in the 70s) and plenty of other languages in the 80s. By then, OO techniques started to emerge, and eventually OO methods were born. The most important early OO methods were:

OMT (Rumbaugh *et al*), OOA (Coad, Yourdon), OOD (Booch), and OOSE (Jacobson). They were followed by Fusion (Coleman) and finally by the Unified Process (Booch, Jacobson, Rumbaugh). A de facto standard set of OO modelling techniques is now UML (Unified Modelling Language).

3 Basic Concepts

(30 minutes)

V10. 3



The main concepts in *object orientation* are listed in the visual.

- *Object* – an entity which has a state and which offers a set of operations (services) to either examine or update this state. The state of the object is stored in the object's attributes. The set of operations represents the object's behaviour.

  Objects communicate *via* messages (usually implemented as service/operation calls).

- *Class* – (or object type) represents a template for several objects and describes how these objects are structured internally. Objects of the same class (*i.e.* class instances) have the same definition for both their operations and their information structure/attributes.

- *Encapsulation* and *Information hiding* means that objects (*i.e. encapsulated* entities) *hide* their internal structure from their surroundings, for example, other objects/clients. They *reveal* their interfaces, that is, some operations/services and some attributes. Only *revealed* facilities can be used/accessed by other objects/clients. Class declarations specify what is *revealed*/public and what is *hidden*/private.

- *Inheritance* – a relationship among classes, wherein one class shares the information structure/attributes and operations/behaviour defined in another class. If class B inherits class A, then both the operations/services and the information structure described in class A will become part of class B. Inheritance defines a *kind_of/is_a* relationship/hierarchy among classes in which a *subclass* inherits from its *superclass*. A subclass typically augments/extends and/or redefines the existing structure and behaviour of its superclass.

  Inheritance can be viewed as *generalisation* from the subclass viewpoint (as its superclass defines a more general *concept*) or as *specialisation* from the superclass perspective (as its subclass defines a more specific/specialised *concept*).

A containing (*aggregation*) relationship implies that objects/instances of one class consist of objects/instances of another class(es). For example, a house consists of/has a roof, 4 walls, a door, *etc*. Tom (for example, an object of class person) consists of/has 2 legs, a head, 2 arms, *etc*. Basically this relationship means that we can use *old* classes when developing *new* classes, not by inheriting them, but by creating instances of them and using these instances in definitions of new classes.

Containing relationships/hierarchies are very useful for showing how the system is constructed, and should be used together with an inheritance hierarchy.

Genericity – Generic Classes – A generic class is one whose operations/services can operate on *any* data type. Generic classes are like templates for deriving data specific classes. For example, a generic stack represents a class of stacks of items of *any* type, *i.e*. stacks of reals, stacks of characters, stacks of payroll records, *etc.* From such a generic class/template some data specific classes can be derived, for example, a class *stacks of reals* if our application requires such stacks. Typically generic classes have a formal *generic* parameter (which represents *any* type), that is replaced by the actual concrete type required in our application.

Usually generic classes represent collections (for example, list, stack, queue, table, set, bag, *etc*).

4 Overview of UML

(45 minutes)

UML was jointly developed by G Booch, I Jacobson and J Rumbaugh in the late 90s. It is a modelling (graphical) language.

UML provides a wide range of diagrams to model different aspects of the system and at different stages of its development (analysis, design etc.). Some of these diagrams are mainly used during requirements analysis, some others during design.

UML is discussed in full in Unit 5 (OOAD). Here a brief overview of the main analysis diagrams (i.e. analysis modelling techniques) is provided.

4.1 Use Case Model/Diagram

A use case diagram of a system shows users/actors of the system, use cases, links between actors and use cases, and the system boundary. It is a very high level view of the system.

Actors represent users or other systems that exchange information with the system.

A use case is a sequence of actions required of the system. Use cases represent and define functional requirements and describe the way the system is employed

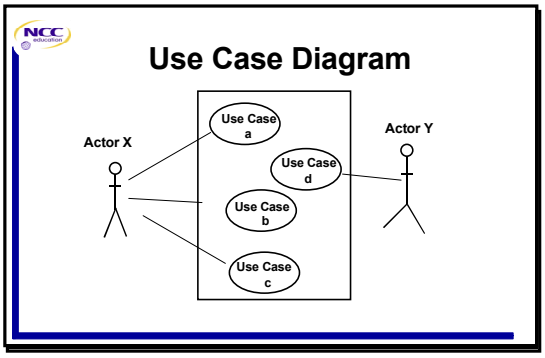
by its actors to achieve their goals. On the diagram each use case is shown as a horizontal ellipse.

Links show which actors use which use cases.

The system boundary is represented by a rectangle showing the scope of the system.

Visual V10.3 below shows the main components of a typical Use Case Diagram.

V10. 4



Note: Concrete examples of use case diagrams should be discussed here e.g. library, bank etc.

4.2 Class Model/Diagram

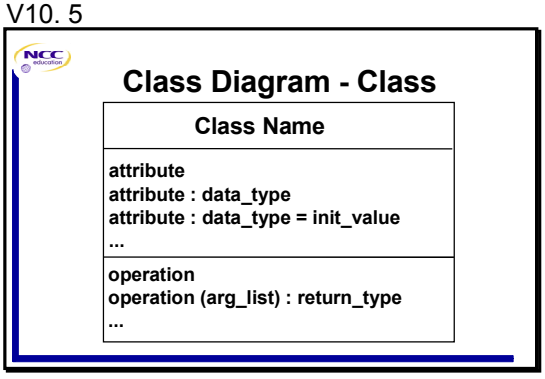
"The class model describes the static structure of classes in a system – their identity, their relationships to other classes, their attributes, and their operations." (**Rumbaugh**)

The notation for:

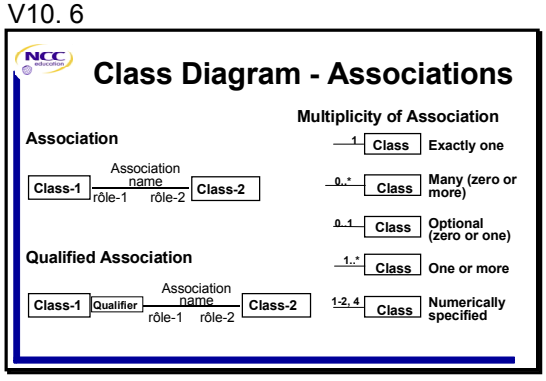
- Class.
- Associations.
- Aggregation.
- Generalisation/Inheritance.

is shown in visuals V10.5, V10.6, V10.7, V10.8.

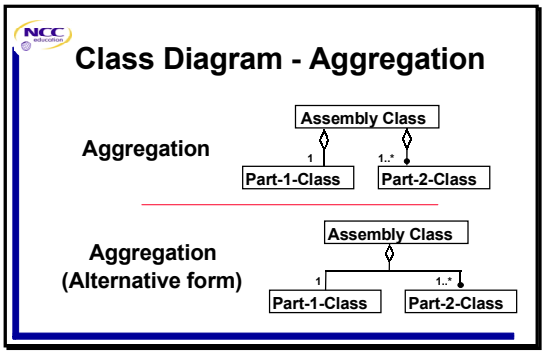
V10. 5



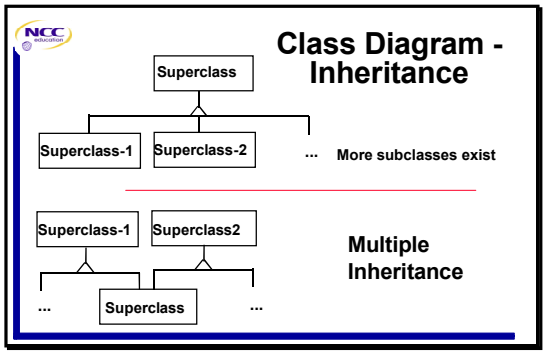
V10. 6



V10. 7



V10. 8



Note: Concrete examples of class diagrams should be discussed here, for example, library, bank etc.

4.3 Interaction Models/Diagrams

UML's interaction diagrams show how different objects interact to *perform* /realise use cases. Usually we produce one interaction diagram per one use case.

An interaction diagram shows all objects *participating* in the realisation of the corresponding use case and their interactions (for example, messages exchanged between objects).

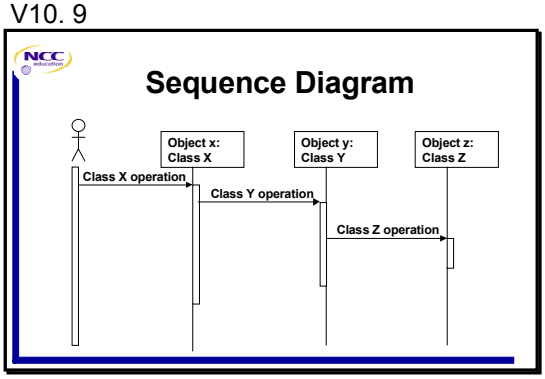
There are two types of interaction diagrams in UML:

- Sequence diagrams.
- Collaboration diagrams.

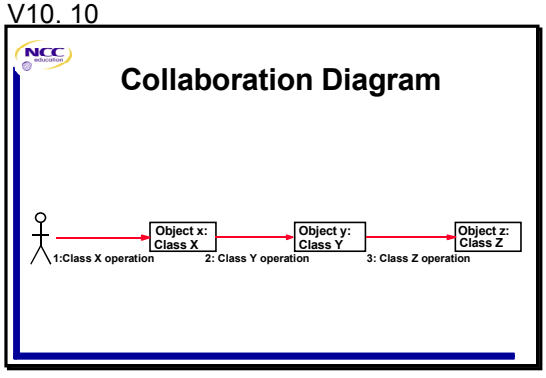
They show almost the same information, but sequence diagrams show the *timing* of interaction more explicitly.

Visuals V10.9 and V10.10 show the notation for *sequence* and *collaboration* diagrams.

V10. 9



V10. 10



Note: Concrete examples of interaction diagrams should be discussed here e.g. in a library, bank system etc.

4.4 State Models/Diagrams

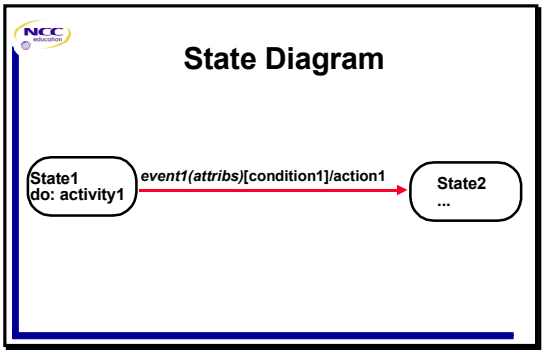
So far we have discussed:

- *Use Cases* and *Use Case models/diagrams.* They are very useful for modelling functional requirements.
- *Class models/diagrams*. They show the static structure of a system.
- *Interaction diagrams.* They show how objects interact to satisfy the functional requirements.

What we also need are diagrams to model object life cycles (i.e. dynamics).

All objects have a state. The state is determined by the current values of the attributes and the links with other objects. The current state of an object depends on the events that have occurred to the object. When the new event *hits* the object i.e. the object receives a message, then the object's reaction depends on its current state. If the event is invalid in the current state, then an error occurs; if the event is valid then an appropriate action is *taken* by the object (for example, an operation is executed). As the result of this action, the object may change its state.

V10. 11



In UML state diagrams are used to model dependencies between states and events.

The notation for State Diagrams is shown in Visual V10. 11.

Note: Concrete examples of state diagrams should be discussed here, for example, in a library, bank system etc.

5 A Simple Object Oriented Development Process

(20 minutes)

A simple OO process includes analysis, design and implementation stages.

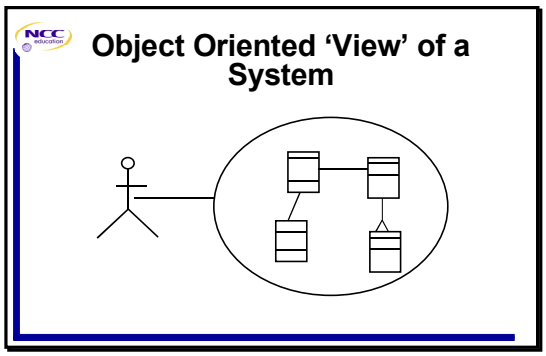
- *Analysis.* The starting point for the analyst is a set of functional requirements. From these requirements the analyst builds a Use Case model. Also a sketchy Class Model can be produced at this stage. The classes in the Class Model correspond to the main application domain classes at this stage. Relationships (*i.e.* associations, aggregations and generalisation/inheritance) among classes are identified and specified. Some obvious class attributes are specified as well.

Detailed descriptions of use cases help to identify more attributes (and some obvious operations). Interaction diagrams are produced. They provide the *main source* of operations. State diagrams for more complex classes are also produced and cross checked with other models. They usually *add* more operations (and attributes) to classes.

- *System design.* The overall architecture is designed. The system is divided into subsystems and allocated to processors and tasks/processes. Some important strategic decisions are taken, for example, on the use of Database Management Systems (DBMS).
- *Object design.* This is the detailed design stage. Algorithms and data structures are designed for individual classes. New implementation/computer domain classes are added. Interfaces among classes are designed in detail.
- *Implementation.* The object submodel (refined and enhanced) is finally translated into a particular programming language and/or database. The translation process is aided by several guidelines provided by the method.

The process is sequential only for really trivial systems. So it is an iterative process delivering the required functionality in increments. For example, the first iteration can result in the delivery of the most important functionality i.e. the most important use cases are included in the first increment. The second iteration delivers less important functionality etc.

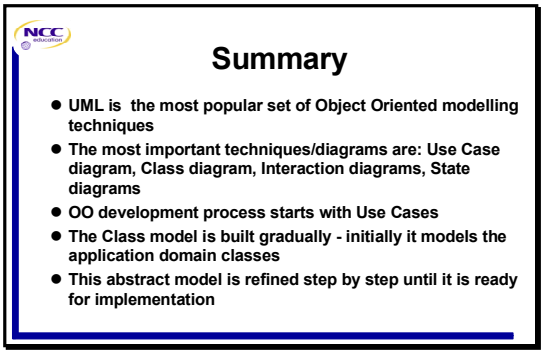
V10. 12



Please note that any system from *object-oriented perspective* looks like a collection of interrelated classes (see Visual V10. 12).

6 Summary

V10. 13 *(5 minutes)*



UML is the most popular set of OO modelling techniques and diagrams.

The most important analysis techniques (and diagrams) have been briefly discussed. They include:

- Use cases.
- Class diagrams.
- Interaction diagrams.
- State diagrams.

In the development process, most of the emphasis is placed on *use case* modelling and *class* modelling. The class model is built gradually. Initially it models the application domain classes. This is quite abstract specification. This abstract model is refined step by step with the help of other techniques until it is *ready* for implementation.

Software Engineering Methods

Session 10
Object Oriented Methods and Techniques

© NCC Education Limited, 2001

Contents

- Object Oriented Methods
- Basic Concepts
- Overview of OO Modelling using UML
- Simple OO Process

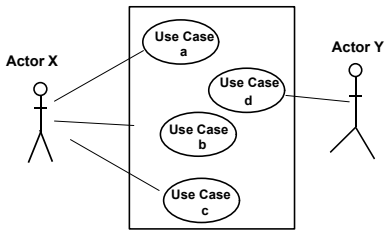
© NCC Education Limited, 2001

Main Concepts in Object Orientation

- Objects
- Classes
- Encapsulation
- Information hiding
- Inheritance (generalisation and specialisation)

© NCC Education Limited, 2001

Use Case Diagram



© NCC Education Limited, 2001

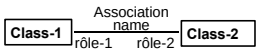
Class Diagram - Class

Class Name
attribute attribute : data_type attribute : data_type = init_value ...
operation operation (arg_list) : return_type ...

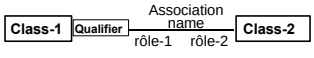
© NCC Education Limited, 2001

Class Diagram - Associations

Association



Qualified Association



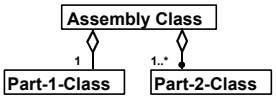
Multiplicity of Association

1 Class	Exactly one
0..* Class	Many (zero or more)
0..1 Class	Optional (zero or one)
1..* Class	One or more
1-2, 4 Class	Numerically specified

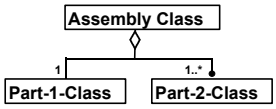
© NCC Education Limited, 2001

Class Diagram - Aggregation

Aggregation

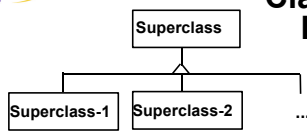


Aggregation (Alternative form)

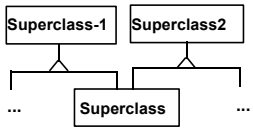


© NCC Education Limited, 2001

Class Diagram - Inheritance



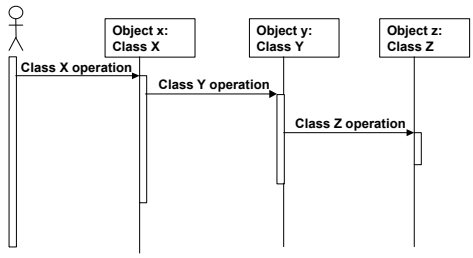
More subclasses exist



Multiple Inheritance

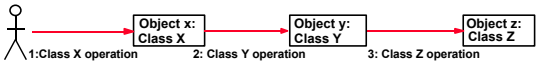
© NCC Education Limited, 2001

Sequence Diagram



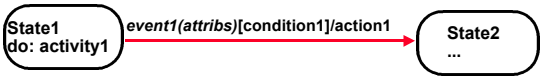
© NCC Education Limited, 2001

Collaboration Diagram



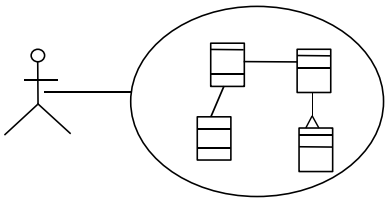
© NCC Education Limited, 2001

State Diagram

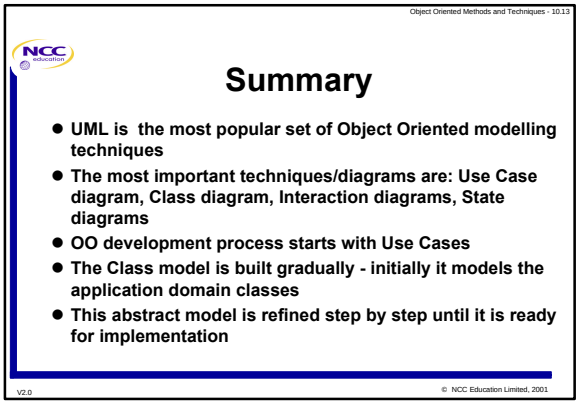


© NCC Education Limited, 2001

Object Oriented 'View' of a System



© NCC Education Limited, 2001


Object Oriented Methods and Techniques - 10.13
Summary
UML is the most popular set of Object Oriented modelling techniques
The most important techniques/diagrams are: Use Case diagram, Class diagram, Interaction diagrams, State diagrams
OO development process starts with Use Cases
The Class model is built gradually - initially it models the application domain classes
This abstract model is refined step by step until it is ready for implementation
V2.0
© NCC Education Limited, 2001