# *A METHODOLOGY FOR INTEGRATING LEGACY SYSTEMS WITH THE CLIENT/SERVER ENVIRONMENT*

by

**MELINDA REDELINGHUYS**

submitted in partial fulfilment of the requirements

for the degree of

**MASTER OF SCIENCE**

in the subject

**INFORMATION SYSTEMS**

at the

**UNIVERSITY OF SOUTH AFRICA**

**SUPERVISOR: PROFESSOR A.L. STEENKAMP**

**June 1996**

# ABSTRACT

The research is conducted in the area of software methodologies with the emphasis on the integration of legacy systems with the client/server environment. The investigation starts with identifying the characteristics of legacy systems in order to determine the features and technical characteristics required of an integration methodology. A number of existing methodologies are evaluated with respect to their features and technical characteristics in order to derive a synthesis for a generic methodology. This evaluation yields the meta primitives of a generic methodology. The revised spiral model (Boehm,1986; Du Plessis & Van der Walt,1992) is customised to arrive at a software process model which provides a framework for the integration of legacy systems with the client/server environment. The integration methodology is based on this process model.

**Key Terms:**

Methodology; Integration; Legacy System; Client/server Environment; Object-Orientation; Software Process Model; Software Development Life-cycle; Mainframe Environment; Integration Trends; Project Management

# ACKNOWLEDGEMENTS

To my Lord, Jesus Christ, all the glory for giving me the strength to do this work and for providing in all my needs.

A special word of thanks to my husband, Johan, for his generous support and encouragement throughout the course of my studies.

Sincere thanks to my supervisor, Professor A. L. Steenkamp, for her invaluable advice, guidance and helpful attitude throughout this project.

Finally, a word of thanks to all my colleagues at SASOL for their help and interest in my studies.

# TABLE OF CONTENTS

**CHAPTER 2 LEGACY SYSTEMS**

## CHAPTER 3 METHODOLOGIES FOR SOFTWARE DEVELOPMENT



## CHAPTER 4 THE INTEGRATION METHODOLOGY

**CHAPTER 5 METHODS, TECHNIQUES, PROCEDURES AND AUTOMATED SUPPORT FOR THE ESMI**



**CHAPTER 6 SUMMARY AND CONCLUSIONS**

# PREFACE

This dissertation is done in partial fulfilment of the MSc-degree in Information Systems at the University of South Africa. The MSc-degree in Information Systems has a total weight of ten modules, the dissertation representing a weight of five modules and course work comprising the additional five modules. The latter includes:

| | |
|---|---|
| INF417-N | Software Engineering |
| INF483-Y | Software Engineering Environments |
| INF404-H | Data Communication and Network Design |
| INF414-K | Object-Orientation |
| INF416-K | A special topic module on Open Systems |

The study forms part of the Object-Oriented Information Systems Engineering Environment (OOISEE) project in the Department of Computer Science and Information Systems at the University of South Africa. The objective of the study is to develop a methodology which can be used to integrate legacy systems with the client/server environment. The masculine form of the third person is used throughout this dissertation to represent both genders.

# LIST OF FIGURES

# LIST OF TABLES

# LIST OF EXHIBITS

# LIST OF ACRONYMS

| | |
|---|---|
| 4GL | 4th Generation Language |
| ANSI | American National Standards Institute |
| API | Application Programming Interface |
| BPR | Business Process Re-engineering |
| CAD | Computer-aided Design |
| CASE | Computer-aided Software Engineering |
| CBS | Cost Breakdown Structure |
| CEN | Comite European de Normalisation |
| CICS | Customer Information Control System |
| CLOS | Common Lisp Object System |
| CMM | Capability Maturity Model |
| COBOL | Common Business-Oriented Language |
| COCOMO | Constructive Cost Model |
| CRC | Class / Responsibilities / Collaborators |
| DADES | Data Oriented Design |
| DBMS | Database Management System |
| DDE | Dynamic Data Exchange |
| DFA | Data Flow Analysis |
| DFD | Data Flow Diagram |
| DOS | Disk Operating System |
| DSS | Decision Support System |
| EDFD | Entity Data Flow Diagram |
| EIS | Executive Information System |
| ER | Entity-relationship |
| ERA | Entity Relationship Analysis |
| ESMI | Enhanced Spiral Model for Integration |
| FS | Sastech Financial |

| | |
|---|---|
| GB | Gigabyte (= 1000 MB) |
| GL | General Ledger |
| GUI | Graphical User-Interface |
| IFD | Information Flow Diagram |
| IPC | Interprocess Communication |
| IS/T | Information Systems and Technology |
| ISO | International Organisation for Standardisation |
| JCL | Job Control Language |
| LAN | Local Area Network |
| MB | Megabyte |
| MIMS | Maintenance Information Management System |
| MIPS | Million Instructions Per Second |
| MMS | Sastech Material Management |
| MVS | Multiple Virtual Storage |
| ODBC | Open Database Connectivity |
| OMT | Object Modeling Technique |
| OO | Object-Orientation |
| OOA | Object-Oriented Analysis |
| OOD | Object-Oriented Design |
| OOISEE | Object-Oriented Information Systems Engineering Environment |
| OSF | Open Software Foundation |
| PAMM | Purchase Ordering Accounts Payable Material Management |
| PERT | Program Evaluation and Review Technique |
| RE | Risk Exposure |
| PRMP | Project Risk Management Plan |
| RPC | Remote Procedure Call |
| RRL | Risk Reduction Leverage |
| SASOL | South African Coal, Oil and Gas Corporation |
| SASTECH | SASOL Technology (Pty) Ltd |
| SDLC | Software Development Life-Cycle |
| SEE | Software Engineering Environment |
| SPMP | Software Project Management Plan |

| | |
|---|---|
| SQA | Software Quality Assurance |
| SQAP | Software Quality Assurance Plan |
| SQL | Structured Query Language |
| SSF | SASOL Synthetic Fuels (Pty) Ltd |
| V & V | Verification and Validation |
| WBS | Work Breakdown Structure |

# GLOSSARY OF TERMINOLOGY

**Client/server Environment**: a computer environment where application processing is split between a client and a server, i.e. two or more computers work together across a network to provide a user-friendly interface residing on the desktop, application functionality shared across multiple machines as required, and database capabilities residing on the more powerful non-desktop machine or machines (servers). It provides a decentralised architecture that enables users to gain transparent access to information within a multi-vendor environment.

**BPR**: the restructuring of organisational (business) processes through the innovative use of information systems and technology (IS/T) in order to secure an organisation's economical survival and competence.

**DBMS**: a structured software component designed to coordinate and facilitate the management of data for multiple applications. DBMS products usually conform to one of several models: hierarchical, relational, object-oriented, or network.

**Downloading**: the process of sending software or data from a central source (e.g. a mainframe computer) to remote stations.

**GUI**: a user-interface for a terminal or PC built with menus, windows and pointing devices.

**LAN**: a data communications system confined to a limited geographic area with moderate to high data rates.

**Legacy System**: in the context of this dissertation, a mainframe-based computer application.

**Integration Trend**: in the context of this dissertation, a tendency in the IS/T industry towards using capable downsizing and client/server- enabling platforms, tools and applications for the purpose of integrating legacy systems with client/server environments.

**Methodology**: a collection of methods and techniques which provides a systematic, unifying approach coordinated by management techniques to guide the developing and improving of software. It is usually based upon an underlying intellectual model (the paradigm).

**SDLC:** in the context of this dissertation, an instance of a software process model.

**Software Process Model:** a phased or cycled development framework for a series of orderly, interrelated activities which facilitates the development, implementation and maintenance of a software system.

**SQA**: in the context of this dissertation, the process of ensuring the extent or degree to which an integrated system is in conformity with established requirements.

**SQAP**: in the context of this dissertation, the plan to determine and measure the extent of SQA in an integration life-cycle.

**SQL**: a standard, widely accepted relational database management language for data definition and manipulation. It is used in varying forms by most vendors of relational DBMS products.

**WBS**: a hierarchical structure useful for organising project activity elements for the purposes of project budgetary planning and control.

# TRADEMARKS

All trademarks are those of their respective owners, including:

**Activity Modeling / Behavior Modeling** is a registered trademark of the Norwegian Institute of Technology, Norway

**Ada** is a registered trademark of the US Department of Defense

**Application-to-Application Interface** is a registered trademark of Micro Focus Limited

**C++** is a registered trademark of AT&T Bell Laboratories

**Compuware XA** is a registered trademark of Compuware Corporation

**Crystal Reports** is a registered trademark of Crystal Computer Services

**DADES** is a registered trademark of the University of Barcelona, Spain

**Datacom** is a registered trademark of Computer Associates Incorporation

**Delphi** is a registered trademark of Borland Incorporation

**Easel** is a registered trademark of Easel Corporation

**Ehllapi** is a registered trademark of International Business Machines

**Eiffel** is a registered trademark of Interactive Software Engineering Incorporation

**Evolutionary Technologies Toolset** is a registered trademark of Extract Corporation

**Excel** is a registered trademark of Microsoft Corporation

**Excelerator** is a registered trademark of Intersolv Corporation

**Forte'** is a registered trademark of Forte' Software Incorporation

**HP-Ux** is a registered trademark of Hewlett-Packard Company

**IBM** is a registered trademark of International Business Machines

**Ideal** is a registered trademark of Computer Associates Incorporation

**Lightship** is a registered trademark of Pilot Software

**Macintosh** is a registered trademark of Apple Computer Incorporation

**MERISE** is a registered trademark of Sema-Metra, France and Gamma International, France

**Method/1** is a registered trademark of Arthur Anderson Corporation

**MVS** is a registered trademark of International Business Machines Corporation

**Object Management Workbench** is a registered trademark of Intellicorp Corporation

**OMTool** is a registered trademark of OMTool Incorporation

**Open Server** is a registered trademark of Sybase Incorporation

**Oracle** is a registered trademark of Oracle Corporation

**OS/2** is a registered trademark of International Business Machines Corporation

**PARTS** is a registered trademark of Digitalk Incorporation

**Powerbuilder** is a registered trademark of Powersoft Incorporation

**Presentation Manager** is a registered trademark of International Business Machines Incorporation

**Prism Warehouse Manager** is a registered trademark of Prism's Warehouse Manager

**Projects** is a registered trademark of Microsoft Corporation

**PSL/PSA** is a registered trademark of the University of Michigan, USA

**Ramba** is a registered trademark of Wall Data Incorporated

**REMORA** is a registered trademark of the University of Paris-1, France

**Renaissance** is a registered trademark of Viasoft Incorporation

**Rose** is a registered trademark of Rational Software Corporation

**Simula** is a registered trademark of the Norwegian Computing Centre, Norway

**Smalltalk** is a registered trademark of Xerox Corporation

**SQL Connect** is a registered trademark of Oracle Corporation

**Sybase** is a registered trademark of Sybase Incorporation

**Unix** is a registered trademark of Unix System Laboratories Inc., a subsidiary of AT & T

**VIA/Insight** is a registered trademark of Viasoft Incorporation

**Visual Basic** is a registered trademark of Microsoft Corporation

**Visual Reengineering Toolset** and **Visual Testing Toolset** are registered trademarks of McCabe and Associates Incorporation

**Windows** and **Windows NT** are registered trademarks of Microsoft Corporation

**X-Windows** is a registered trademark of the Massachusetts Institute of Technology (MIT)

# CHAPTER 1

# CONTEXT OF RESEARCH

## 1.1 Introduction

Most organisations have large investments in traditional legacy systems[1] and as a result organisational strategies usually do not totally eliminate them. High maintenance costs, which are constantly rising and which include the cost of the large personnel component necessary to operate the computer system, are significant with legacy systems. These systems are furthermore inflexible, difficult to maintain and their data is inaccessible. Insufficient, or a total lack of documentation adds to the difficulty of maintaining legacy systems and usually very few, if any, of the system analysts and programmers involved in the development of these systems are still employed at the time that maintenance is required.

Since the 1990's many organisations have based strategic information system decisions on client/server computing. The primary focus of new systems development is on client/server applications designed to run on more cost-effective platforms. A powerful driver behind client/server computing has been the promise of cost savings through the downsizing of corporate legacy systems. Client/server computing exploits the relative advantages of each type of platform while maximising an organisation's investments in existing systems.

As a result of the large capital investments represented by legacy systems as well as tight time schedules, it is not always feasible to replace these systems without delay. Interoperability with these legacy systems is required. Interoperability in this context refers to the ability to interconnect platforms in order to be able to exchange data. Organisations are therefore in need of user-interfaces to legacy systems.

## 1.2 Problem Statement and Relevance of Solution

Different strategies for integrating legacy systems with the client/server environment exist (Simonds,1992; Xephon,1993; Forge,1995; Kavanagh,1995). Decisions regarding a strategy for integrating a specific legacy system with the client/server environment are mainly influenced by the

[1] In the context of this dissertation, a legacy system is a mainframe-based computer application.

characteristics of the relevant legacy system. In order to manage and control the integration process, a methodology is needed.

### 1.2.1 Problem Statement

This research project proposes to explore the requirements of a methodology for integrating legacy systems with the client/server environment in a petrochemical organisation, the petrochemical organisation being South African Coal, Oil and Gas Corporation (SASOL).

The following subproblems were identified:

1. the construction of a reliable questionnaire to identify the characteristic properties of the legacy systems existing within SASOL;

2. the identification of the main trends for integrating legacy systems with the client/server environment in the Information Systems and Technology (IS/T) industry;

3. the identification of the basic concepts of a generic methodology as well as the various modelling perspectives which methodologies allow;

4. the identification of the meta-primitives of a methodology for integrating legacy systems with the client/server environment;

5. the proposal of a software process model on which the integration methodology will be based;

6. the identification and description of the most relevant methods, techniques, procedures and deliverables for an integration methodology.

### 1.2.2 Relevance of the Solution

IS/T managers and engineers need a methodology to guide them through the process of integrating legacy systems with the client/server environment. The methodology must assist the IS/T managers and engineers in making decisions regarding a specific integration with a client/server environment, guide them during the migration process as well as highlight possible pitfalls. It should include a description of methods, techniques, associated deliverables as well as activities and their sequence. In addition it should have a degree of generality as it should be possible to apply or adapt to a variety of business domains.

## 1.3 Hypotheses and Objectives

In terms of the context of the problem statement the following hypotheses and objectives were formulated:-

### 1.3.1 Hypotheses

1. As illustrated in Figure 1.1, a methodology for integrating legacy systems with the client/server environment can be developed.

2. It is possible to define a software process model on which such a methodology is based.

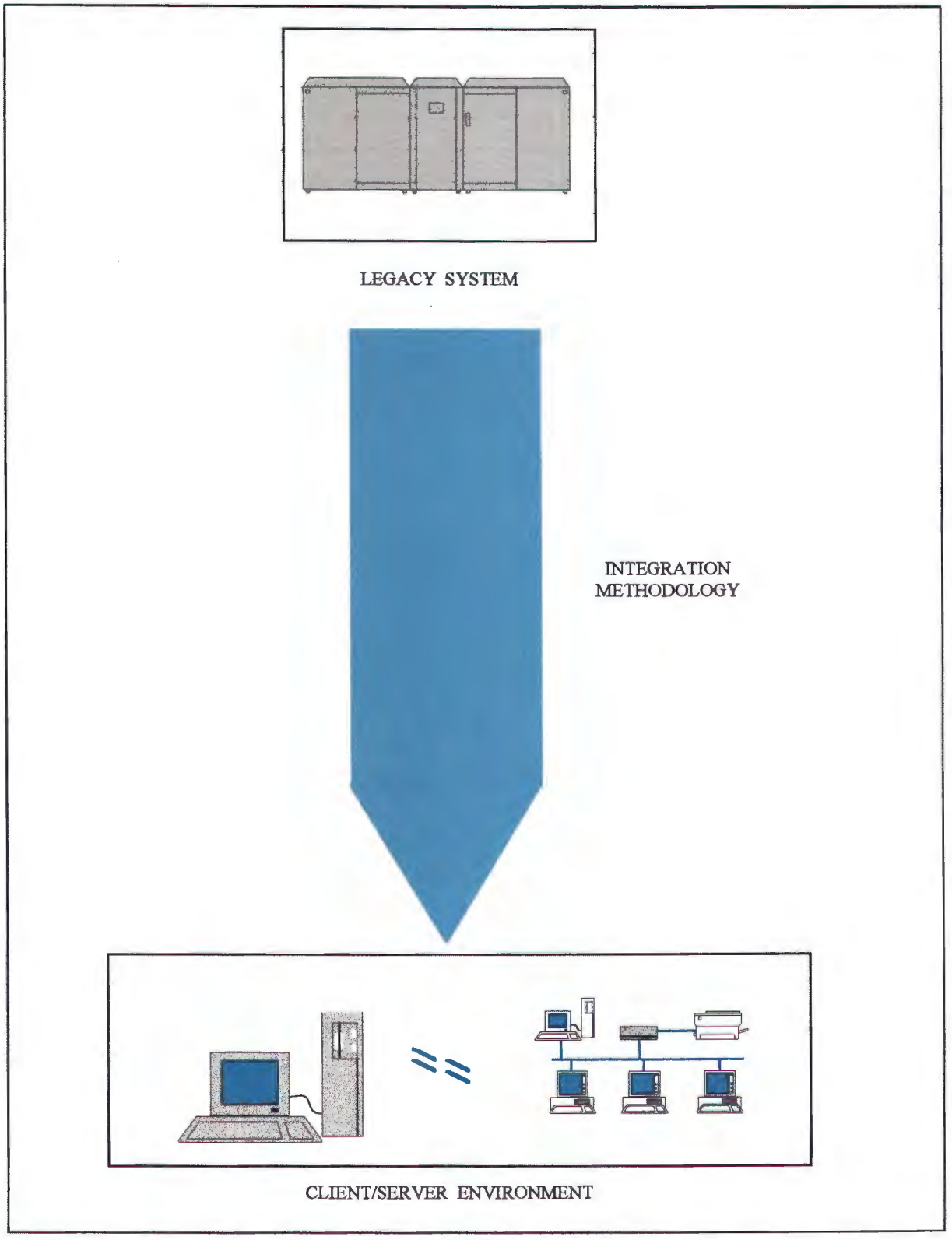


**Figure 1.1** Conceptualisation of Integration

### 1.3.2 Objectives

1. The strengths and weaknesses of both mainframe and client/server environments will be determined.

2. The characteristic properties of legacy systems existing within SASOL will be determined.

3. The generic concepts of methodologies as well as the various perspectives taken by particular methodologies will be considered.

4. A methodology as well as a software process model for integrating legacy systems with the client/server environment will be proposed.

## 1.4 Constraints

The following factors constrained the investigation:-

1. The application of the proposed methodology to a real world application will not form part of the investigation.

2. The research is to be done within the parameters of the OOISEE project in the Department of Computer Science and Information Systems at the University of South Africa. Object-oriented software development is a parameter of the OOISEE project which influenced the investigation.

3. The scope of the research is to meet the requirements for a partial dissertation.

## 1.5 Assumptions

The following assumptions were made:

1. Business Process Re-engineering (BPR) has been done (using a specific methodology) for all divisions within SASOL which are currently using the involved legacy systems as informational resources;

2. legacy systems are to be found in most application domains and these systems have characteristic properties.

## 1.6 Method of Investigation

Based on the context of the problem statement the hypotheses and objectives of the research were formulated. The investigation was conducted in terms of the constraints and assumptions outlined above, and proceeded in the following way:-

1. The relevant issues of the problem domain were identified. These were legacy systems, methodologies, trends for integrating legacy systems with the client/server environment in the IS/T industry, object-orientation (OO) as well as client/server environments.

2. The literature survey of the identified topics yielded a large number of references, which were analysed and interpreted for their relevance.

3. A preliminary screening of all legacy systems within SASOL yielded the identification of five legacy systems in various application domains. A questionnaire was devised for identifying the characteristic properties of these systems. The purpose here was to use this data, along with data from the literature, to determine the features and technical characteristics required of an integration methodology.

4. A number of methodologies were evaluated with respect to their features and technical characteristics in order to derive a synthesis for an integration methodology. This evaluation yielded the meta primitives of the integration methodology.

5. A software process model for integration was proposed.

6. The hypotheses and objectives of the research were validated.

## 1.7 Structure of the Dissertation

The dissertation consists of six chapters.

Chapter 1 defines the context of research and includes the statement of the problem as well as the relevance of the solution. The hypotheses and objectives are formulated in the context of the problem statement. Constraints of the investigation are defined and assumptions are made. This is followed by the method of investigation which guided the research. The chapter concludes with an overview of the content of the dissertation.

Chapter 2 deals with legacy systems and their characteristic properties. The mainframe environment is briefly discussed with emphasis on its strengths and weaknesses. The design aspects and distribution procedure of the questionnaire are presented after which the results of the questionnaire are presented, analysed, interpreted and summarised. A few relevant integration trends in the IS/T industry are reviewed. The client/server environment is briefly discussed with emphasis on its strengths and shortcomings. In the final section of this chapter several strategies for the integration of legacy systems with client/server environments are reviewed.

Methodologies for software development are covered in Chapter 3. The basic concepts of a sound generic methodology are identified after which the various perspectives taken by existing methodologies are discussed. The roles of both the software process model and software

development life-cycle (SDLC) are explained after which the different levels of the Capability Maturity Model (CMM) proposed at the Carnegie-Mellon University, are summarised.

Chapter 4 is devoted to the integration methodology. The features and technical requirements of the integration methodology are identified. A presentation of the generic tasks to be undertaken during the integration life-cycle follows. The Enhanced Spiral Model for Integration (ESMI) is proposed as a software process model for integrating legacy systems with the client/server environment, followed by a synopsis of the representation schema as well as the most relevant methods, techniques, procedures and deliverables of this software process model. Project management is addressed in the final section of the chapter.

In Chapter 5 the most relevant methods, techniques and procedures of the ESMI are discussed in detail. Automated support for the ESMI is covered in the final section of the chapter.

The research results are summarised and the conclusions drawn during the investigation are stated in Chapter 6. Areas for further investigation are proposed.

# CHAPTER 2

# LEGACY SYSTEMS

# CHAPTER 2

# LEGACY SYSTEMS (CONTINUED)

## 2.1 Introduction

The investigation of the business processes, the legacy application systems that support these processes as well as the technology supporting these systems are all prerequisites for deciding on an integration strategy for a legacy system with the client/server environment. Kavanagh (1995)[1] suggests that the following questions be answered for each business process:-

1. Is the basic business process necessary?
2. Is it working well enough?
3. Is the legacy application working well?
4. Is additional functionality needed?
5. Is additional ease of use needed?
6. Is the underlying technology working well?
7. Is the technology expensive or obsolete?

The investigation of the business processes, i.e. Business Process Re-engineering (BPR), was outside the scope of this investigation, but an empirical survey was conducted focusing on five legacy systems which exist in various application domains within SASOL. This was done by means of a questionnaire which aimed at answering the above set of questions for the five legacy systems, as well as identifying their characteristic properties.

In this chapter the strengths and shortcomings of mainframe environments are discussed. The design of the questionnaire and the procedures for distributing it are explained, followed by a presentation of the results that were obtained. These results are summarised and interpreted after

[1] Many of the ideas regarding the integration of legacy systems with client/server environments were derived from those of Kavanagh (1995).

which trends in the IS/T industry relevant to integrating legacy systems with the client/server environment, are considered. The strengths and weaknesses of client/server environments are reviewed. An overview of the different strategies for integrating legacy systems with the client/server environment follows.

## 2.2 Mainframe Environments

Mainframe environments are host-based and are generally considered "old technology". They have existed since the late 1960's. They represent large investments in application software, networks, staff, system software and hardware in organisations.

They are known for their stability and secure nature. The data to be protected is centralised and practices and tools have evolved to defend this security. Mainframe system software (e.g. the MVS operating system, CICS database management system and COBOL compiler) are mature and well-understood. Terminals are cheaper than workstations and they provide for the limited, fixed computing needs of users at a reasonable price. Mainframes offer easy organisation-wide communications without the need for intercomputer communication. Mainframe systems' strengths are batch processing and automated job scheduling.

A major problem with mainframe environments is the high maintenance costs involved. The large number of personnel necessary to operate the computer can be up to 60% of the total IS/T budget (Kavanagh,1995). Legacy systems are inflexible and the data is inaccessible. The lack of user-friendly and mouse-driven graphical user-interfaces (GUI's ) is another major disadvantage. Companies are frequently locked into a particular vendor's technology with the concomitant disadvantage of being unable to consider competing technologies. In such an environment it is not possible to entertain the options offered by open systems offerings in the marketplace.

## 2.3 Questionnaire Design

The questionnaire was based on information obtained from the literature survey (Simonds,1992; Xephon,1993; Grosvenor,1994; Kavanagh,1995) as well as six years of personal experience maintaining legacy systems. It was designed to elicit information in thirteen broad areas:

- personal details;
- legacy system details;
- size;
- growth;
- integration with other systems;
- cost;
- quality;
- critical factors;
- business support;
- available skills;
- uniqueness;
- maintenance history, as well as
- a comments area.

The questionnaire comprised a total of 45 questions. Even numbers of alternate response options were given to avoid neutral answers thereby ensuring that the expression of views were either negative or positive. A respondent was either a user of the system or responsible for maintaining the system. A user of the system was requested only to complete the questions in the sections on personal details, cost, quality, critical factors, business support, uniqueness, maintenance history and comments. A person responsible for maintaining the system was requested to complete all sections. The questionnaire is presented in Appendix A.

## 2.4 Questionnaire Procedure

As a result of the limited time available, the survey was conducted in the form of a questionnaire. This approach was deemed adequate within the context of this dissertation. The questionnaire shown in Appendix A was given to the project leaders of the five identified legacy systems, namely the Sastech Material Management System (MMS), the Sastech Financial System (FS), the General Ledger System (GL), the Purchase Ordering Accounts Payable Material Management System (PAMM) as well as the Maintenance Information Management System (MIMS). After discussing the questionnaire with these project leaders, they were asked to distribute copies of the questionnaire among a few users whom they regard as having a thorough knowledge of the system involved, as well as among the maintenance personnel of the system. The project leaders were also prepared to assist users and maintenance personnel in the completion of the questionnaire.

A total of 39 questionnaires were sent out and respondents were given six weeks for the completion and return of the questionnaires. When considering the number of users of the five systems, the number of questionnaires sent out may seem very small. Very few users, however, are able to answer questions regarding factors such as cost, critical factors, business support, uniqueness and maintenance history, the reason being that most users only use a small part of a legacy system and they therefore do not have the global view of the system which was required to complete the questionnaire.

## 2.5 Presentation of Questionnaire Results

A total of 26 responses were received of which seven were for the MMS System, six for the FS System, three for GL System, four for the PAMM System and six for the MIMS System. The questionnaire responses are presented in Appendix B and the results are summarised in Table 2.1. The results for each of the five systems will now be considered.

| Criteria | MMS | FS | GL | PAMM | MIMS |
|---|---|---|---|---|---|
| **Response Profile** | | | | | |
| • Users (Nbr) | 5 | 4 | 1 | 2 | 5 |
| • Maintenance Personnel (Nbr) | 2 | 2 | 2 | 2 | 1 |
| **Legacy System Details** | | | | | |
| • Hardware | IBM | IBM | IBM | IBM | IBM |
| • Operating System | MVS | MVS | MVS | MVS | MVS |
| • DBMS | Datacom | Datacom | Datacom | Datacom | Datacom |
| • Application Language | Ideal | Ideal | COBOL | COBOL | COBOL |
| • Methodology | Method/1 | Method/1 | None | Method/1 | None |
| **Size** | | | | | |
| • Users (Nbr) | 69 | 58 | 160 | 1 600 | 1 200 |
| • Online Programs (Nbr) | 541 | 466 | 236 | 460 | 527 |
| • Batch Programs (Nbr) | 129 | 161 | 140 | 30 | 180 |
| • Database (MB) | 935 | 747 | 5 269 | 6 160 | 8 645 |
| • Transactions / Day (Nbr) | 5 182 | 3 273 | 5 000 | 2 500 | 130 000 |
| **Growth** | | | | | |
| • Monthly Database Growth (%) | 15 | 13 | 6 | 9,75 | 5 |
| • High Volume Updates | Yes | Yes | Yes | Yes | No |
| **Integrated with:** | | | | | |
| • Legacy Systems | FS | MMS GL | PAMM MIMS FS | GL MIMS | GL PAMM |
| • Other Systems (Nbr) | 1 | 2 | 10 | 1 | 2 |

**Table 2.1** Summary of Questionnaire Results

| Criteria | MMS | FS | GL | PAMM | MIMS |
|---|---|---|---|---|---|
| **Cost** | | | | | |
| • Age of System (Years) | 7 | 7 | 8 | 8 to 9 | 6 |
| • Annual Operating (Rm) | 1,4 | 2,9 | 3,4 | 5,7 | 10,0 |
| • Total Development & Maintenance (Rm) | 14,7 | 21,0 | 6,4 | 10,0 | 30,0 |
| • Total Hardware (Rm) | 2,7 | 3,8 | 14,6 | 30,5 | 50,0 |
| **Quality** | | | | | |
| • Satisfaction of Requirements | GS | GS | GS | GS | GS |
| • Accuracy of Original Specifications | GS | GS | GS | GS | GS |
| • Need for Additional Functionality | Yes | No | Yes | Yes | Yes |
| • Need for Additional Ease of Use | Yes | Yes | No | * | Yes |
| • Underlying Technology | GS | GS | GS | GS | U |
| • System Documentation | GS | GU to GS | CS | GS to CS | * |
| • Data | GS | GS | CS | GS | * |

**Table 2.1** Summary of Questionnaire Results (Continued)

* Responses varied to such an extent that a result cannot be given.

U - Unsatisfactory
GU - Generally Unsatisfactory
GS - Generally Satisfactory
CS - Completely Satisfactory

| Criteria | MMS | FS | GL | PAMM | MIMS |
|---|---|---|---|---|---|
| **Critical Factors** | | | | | |
| • Cost | C | EC | FC to C | C | FC |
| • Security | NC to FC | C to EC | C | C | C |
| • Availability | * | C | C to EC | C to EC | C to EC |
| • System Performance | C to EC | C | C to EC | C | C to EC |
| **Business Support** | | | | | |
| • Support for Organisational Processes | GS | GS | GS | GS | * |
| • Mission-critical | Yes | Yes | No | Yes | Yes |
| **Available Skills** | | | | | |
| • Difficulty to obtain internally skilled people | FD | FD | FD | VE | FD |
| • Difficulty to obtain externally skilled people | FD | FD | FD | FD | ED |

**Table 2.1** Summary of Questionnaire Results (Continued)

* Responses varied to such an extent that a result cannot be given.

U - Unsatisfactory
GU - Generally Unsatisfactory
GS - Generally Satisfactory
CS - Completely Satisfactory

NC - Not Critical
FC - Fairly Critical
C - Critical
EC - Extremely Critical

VE - Very Easy
FE - Fairly Easy
FD - Fairly Difficult
ED - Extremely Difficult

| Criteria | MMS | FS | GL | PAMM | MIMS |
|---|---|---|---|---|---|
| **Uniqueness** | | | | | |
| • Complexity | Fair to Medium | Medium | Medium | Fair to Medium | Medium |
| • Functional Aspects | FU to U | FU | * | FU | * |
| • Existence of Packaged Software | No | No | Yes | No | Yes |
| **Maintenance History** | | | | | |
| • Corrective Maintenance | FS | FS | * | FO | FO |
| • Limited to Certain Parts | Yes | No | No | Yes | No |

**Table 2.1** Summary of Questionnaire Results (Continued)

* Responses varied to such an extent that a result cannot be given

| | |
|---|---|
| NU - Not Unique | VS - Very Seldom |
| FU - Fairly Unique | FS - Fairly Seldom |
| U - Unique | FO - Fairly Often |
| EU - Extremely Unique | VO - Very Often |

### 2.5.1 The MMS System

The MMS System has a material management nature and is used for the purchase, procurement and management of project material within SASOL Technology (SASTECH) which is a member of the SASOL Group of Companies.

**Legacy System Details:** the MMS System is running on an IBM mainframe computer, with operating system MVS and DBMS, the latest version of Datacom. The application language is Ideal, which is a 4GL. Method/1 was used as methodology for development.

**Response Profile:** five users and two employees responsible for maintaining the system responded.

**Size:** the system has 69 users and consists of 541 online and 129 batch programs. The size of the database is 935 MB and there are 5 182 transactions a day.

**Growth:** the database grows at a rate of 15% a month and high volume updates are a characteristic of the system.

**Integration with Other Systems:** the system is tightly integrated with the FS System as well as with a DOS-based decision support system which runs on a LAN.

**Cost:** the MMS System has been in use in a production environment for about seven years. Some respondents reckoned it to be less, but that may be because they have used the system for less than seven years. The annual total cost to run the system amounts to at least 1,4 million rand. The estimated total cost of development and maintenance to date amounts to 14,7 million rand with a total expenditure on hardware for 2,7 million rand.

**Quality:** as far as requirements are concerned, all respondents felt the system to be generally satisfactory and that original specifications describe the existing functionality of the system fairly accurately. Four users reckoned that additional functionality was needed whereas the two employees responsible for maintenance as well as one user felt there was no need for additional

functionality. The vote for additional ease of use was four against one, with the maintenance personnel voting one in favour and the other against additional ease of use. Four respondents (including one maintenance employee - the other one left this answer blank) reckoned the underlying technology to be obsolete but satisfactory, while two users felt it was current and satisfactory. In general, it was felt that system documentation was satisfactory. All respondents agreed that the quality of the data was generally satisfactory.

**Critical Factors:** as illustrated in Figure 2.1, system performance is critical to extremely critical while the cost of running the system is regarded as critical.

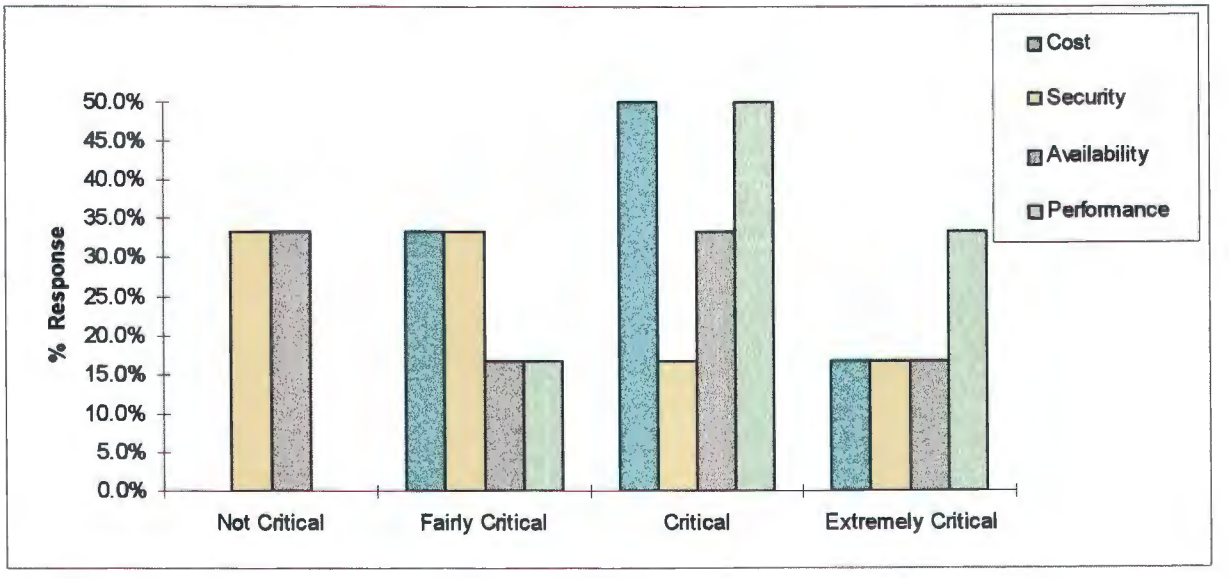


**Figure 2.1** Critical Factors of MMS

**Business Support:** all respondents considered the support provided by the system to organisational processes to be generally satisfactory. The system is mission-critical to SASOL.

**Available Skills:** less than 50% of the employees involved in the original development and maintenance of the system is still employed by SASOL and the same percentage with the necessary skills to maintain the system is still employed by SASOL. It is fairly difficult to find people with the appropriate skills to maintain the system in the external market.

**Uniqueness:** the MMS System has fair to medium complexity and the functional aspects of the system are fairly unique to unique to SASOL. Packaged software with similar functionality is available but satisfies only 80% of the requirements.

**Maintenance History:** corrective maintenance is fairly seldom required and is limited to certain parts of the system.

### 2.5.2 The FS System

The FS System has a financial nature and is used for accounts payable and project accounting within SASTECH. All material ordered and received through the MMS System is payed through the FS System.

**Legacy System Details:** the FS System is running on an IBM mainframe, with operating system MVS and DBMS, the latest version of Datacom. The application is developed in Ideal. Method/1 was used as a methodology for development.

**Response Profile:** four users and two employees responsible for the maintenance of the system responded.

**Size:** the system has 58 users and consists of 466 online and 161 batch programs. The size of the database is 747 MB and there are 3 273 transactions a day.

**Growth:** the database grows at a rate of 13% a month and high volume updates are a characteristic of the system.

**Integration with Other Systems:** the system is tightly integrated with the MMS System and supplies data to the GL System as well as two DOS-based systems running on LAN's.

**Cost:** the FS System has been in use in a production environment for about seven years. The annual total cost to run the system amounts to 2,9 million rand. The estimated total cost of

development and maintenance to date amounts to 21 million rand with a total expenditure on hardware of approximately 3,8 million rand.

**Quality:** as far as requirements are concerned, the system is generally satisfactory and original specifications describe the existing functionality of the system fairly accurate. Little additional functionality is needed, but the ease of use can be improved. The underlying technology is satisfactory. System documentation is generally unsatisfactory to satisfactory and the quality of the data is satisfactory.

**Critical Factors:** as illustrated in Figure 2.2, the cost of running the system is extremely critical while security is critical to extremely critical. Availability and system performance are also regarded as critical.

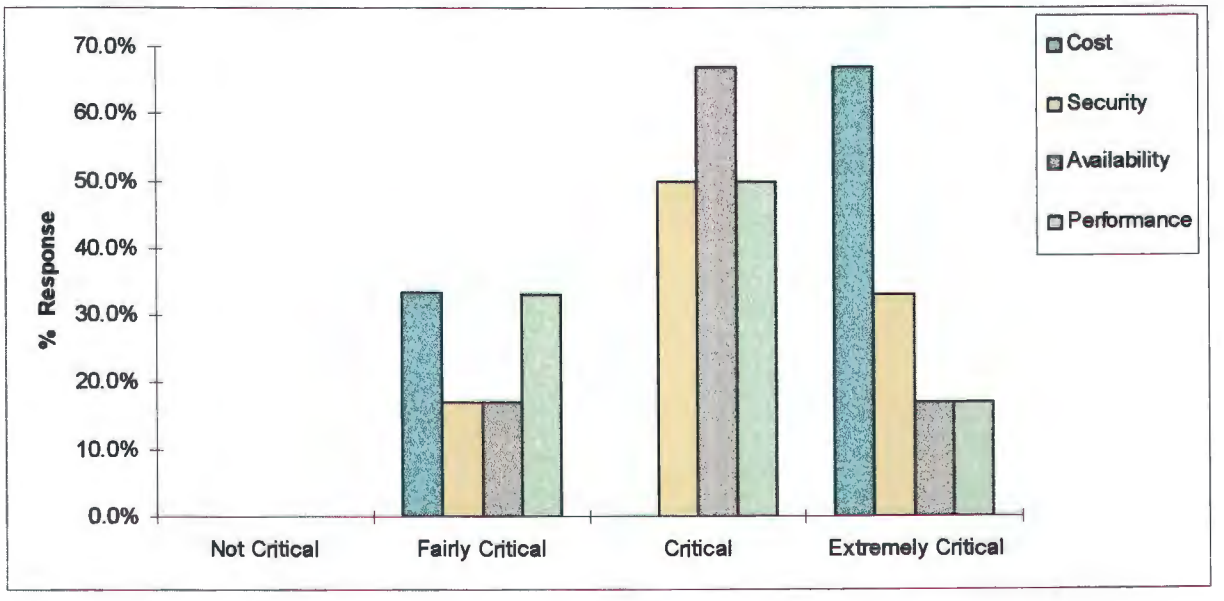


**Figure 2.2** Critical Factors of FS

**Business Support:** the support provided by the system to organisational processes is generally satisfactory and the system is regarded as mission-critical to SASOL's business.

**Available Skills:** less than 50% of the employees involved in the original development and maintenance of the system is still employed by SASOL and the same percentage with the necessary skills to maintain the system is still employed. It is fairly difficult to find people with the appropriate skills to maintain the system in the external market.

**Uniqueness:** the system is of medium complexity. The functional aspects of the system are fairly unique to SASOL, therefore packaged software with similar functionality only satisfies 60% of the requirements.

**Maintenance History:** corrective maintenance is fairly seldom required and is not limited to certain parts of the system.

### 2.5.3 The GL System

The GL System has a financial application domain and is used within SASOL Synthetic Fuels (SSF) which is a member of the SASOL Group of Companies.

**Legacy System Details:** the GL System is packaged software which was customised for SASOL's business. It is running on an IBM mainframe computer, with operating system MVS and DBMS Datacom. The application language is COBOL.

**Response Profile:** one user and two employees responsible for the maintenance of the system responded.

**Size:** the system has 160 users and consists of 236 online and 140 batch programs. The size of the database is 5 269 MB and there are an average of 5 000 transactions a day.

**Growth:** the database grows at a rate of 6% a month and high volume updates are a characteristic of the system.

**Business Support:** the support provided by the system to organisational processes is generally satisfactory and the system is not regarded mission-critical to the business.

**Available Skills:** less than 50% of the employees involved in the original development and maintenance of the system is still employed by SASOL and the same percentage with the necessary skills to maintain the system is still employed. It is fairly difficult to find people with the appropriate skills to maintain the system in the external market.

**Uniqueness:** GL is of medium complexity. Some parts of the system are not unique to SASOL whereas other parts are unique and packaged software exists for a large part of the system.

**Maintenance History:** some respondents reckoned corrective maintenance to be required fairly often whereas others were of the opinion that it was required very seldom. It was, however, not limited to certain parts of the system.

### 2.5.4 The PAMM System

The PAMM System has a material management application domain and is used within SSF for the purchase, procurement and management of material. In contrast with the MMS System which only provides for project stores (i.e. only material ordered for a project is kept in store), the PAMM System supports full store maintenance and management (i.e. material is reordered whenever a predefined level is reached). All material ordered and received through the PAMM System is paid through the GL System.

**Legacy System Details:** the PAMM System is packaged software which was customised for SASOL's business. It is running on an IBM mainframe computer, with operating system MVS and DBMS Datacom. The application language is COBOL, which is not the latest version available. Method/1 was used as methodology for customising the system. One respondent commented that in practice no methodology was used.

**Response Profile:** two users as well as two employees responsible for maintaining the system responded.

**Size:** the system has 1 600 users and consists of 460 online and 30 batch programs. The size of the database is 6 160 MB and there are 2 500 transactions a day.

**Growth:** high volume updates are a characteristic of the system.

**Integration with Other Systems:** the system is tightly integrated with both the GL and MIMS Systems as well as with one PC application.

**Cost:** PAMM has been in use in a production environment for eight to nine years. The annual total cost to run the system amounts to 5,7 million rand. The estimated total cost of development and maintenance to date amounts to ten million rand with a total expenditure on hardware of 30,5 million rand.

**Quality:** as far as requirements are concerned, the system is generally satisfactory and original specifications describe the existing functionality of the system fairly accurate. However, additional functionality is needed in the system. The underlying technology is satisfactory. System documentation is satisfactory to completely satisfactory and the quality of the data is satisfactory.

**Critical Factors:** availability is regarded as critical to extremely critical while performance, cost and security are critical. The critical factors are illustrated in Figure 2.4.

**Business Support:** the support provided by the system to organisational processes is generally satisfactory and the system is regarded as mission-critical to the business.

**Available Skills:** more than 75% of the employees involved in the original development and maintenance of the system is still employed by SASOL and the same percentage with the necessary skills to maintain the system is still employed by SASOL. It is fairly difficult to find people with the appropriate skills to maintain the system in the external market.

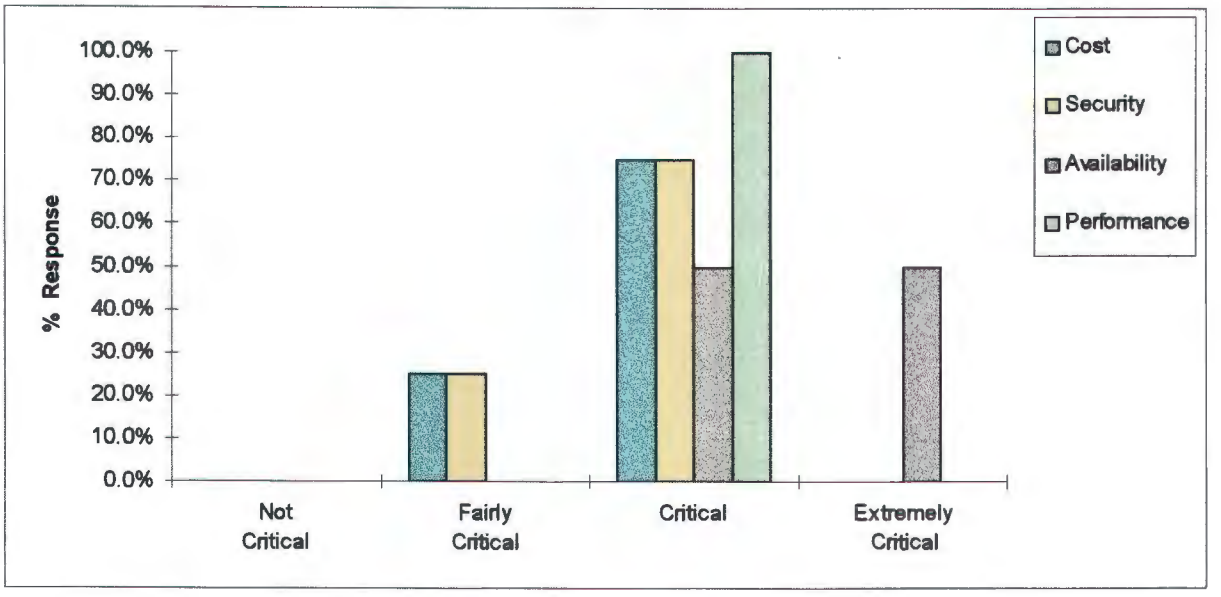


**Figure 2.4** Critical Factors of PAMM

**Uniqueness:** PAMM is of fair to medium complexity. The functional aspects of the system are fairly unique to SASOL and packaged software which satisfies all requirements does not exist.

**Maintenance History:** corrective maintenance is required fairly often and is limited to certain parts of the system.

### 2.5.5 The MIMS System

The MIMS System has a maintenance information nature and is used within SSF for the capturing of plant maintenance information as well as for providing management information concerning plant maintenance. All material required for plant maintenance is ordered and received through the PAMM System and paid through the GL System.

**Legacy System Details:** the MIMS System is packaged software which was customised for SASOL's business. It is running on an IBM mainframe computer, with operating system MVS and DBMS Datacom version 3.010, which is not the latest version available. The application language is COBOL.

**Response Profile:** five users and the project leader responsible for the maintenance of the system responded.

**Size:** the system has 1 200 users and consists of 527 online and 180 batch programs. The size of the database is 8 645 MB and there are 130 000 transactions a day.

**Growth:** the database grows at a rate of 5% a month and high volume updates are not a characteristic of the system.

**Integration with Other Systems:** the system is tightly integrated with other systems which include both the GL and PAMM Systems as well as two other, one which is Unix-based and one which is DOS-based.

**Cost:** MIMS has been in use in a production environment for about six years. The annual total cost to run the system amounts to ten million rand. The estimated total cost of development and maintenance to date amounts to 30 million rand with a total expenditure on hardware of at least 50 million rand.

**Quality:** as far as requirements are concerned, the system is generally satisfactory and original specifications describe the existing functionality of the system fairly accurate. However, additional functionality and ease of use are needed in the system. The underlying technology is unsatisfactory.

**Critical Factors:** system performance and availability are regarded as critical to extremely critical while security is critical. The cost of running the system is fairly critical. The critical factors are illustrated in Figure 2.5.

**Business Support:** the support provided by the system to organisational processes is generally unsatisfactory to completely satisfactory and the system is regarded as mission-critical to the business.

**Integration with Other Systems:** the system is tightly integrated with both the MIMS and PAMM Systems as well as with various other PC applications. A monthly data file supplied by the FS System is processed.

**Cost:** GL has been in use in a production environment for eight years. The annual total cost to run the system amounts to 3,4 million rand. The estimated total cost of development and maintenance to date amounts to 6,4 million rand with a total expenditure on hardware of at least 14,6 million rand.

**Quality:** as far as requirements are concerned, the system is generally satisfactory and original specifications describe the existing functionality of the system fairly accurate. However, additional functionality is needed in the system. Little additional ease of use is required. The underlying technology is satisfactory. The quality of the data as well as system documentation are completely satisfactory.

**Critical Factors:** system performance and availability are regarded as critical to extremely critical. Security is critical while the cost of running the system is fairly critical to critical. The critical factors are illustrated in Figure 2.3.

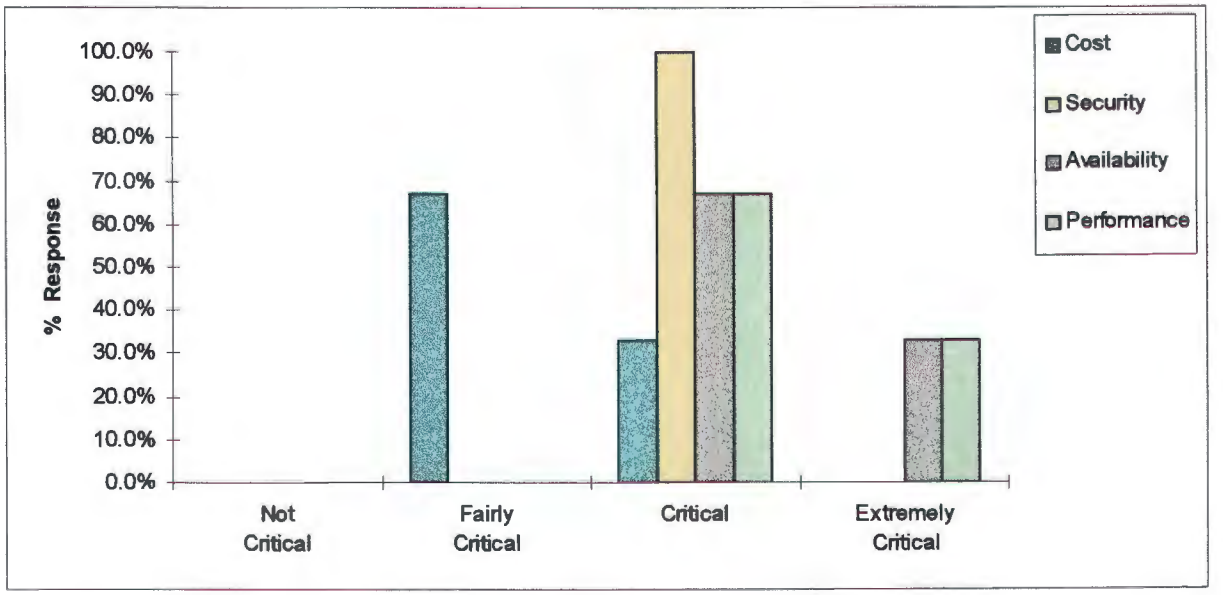


**Figure 2.3** Critical Factors of GL

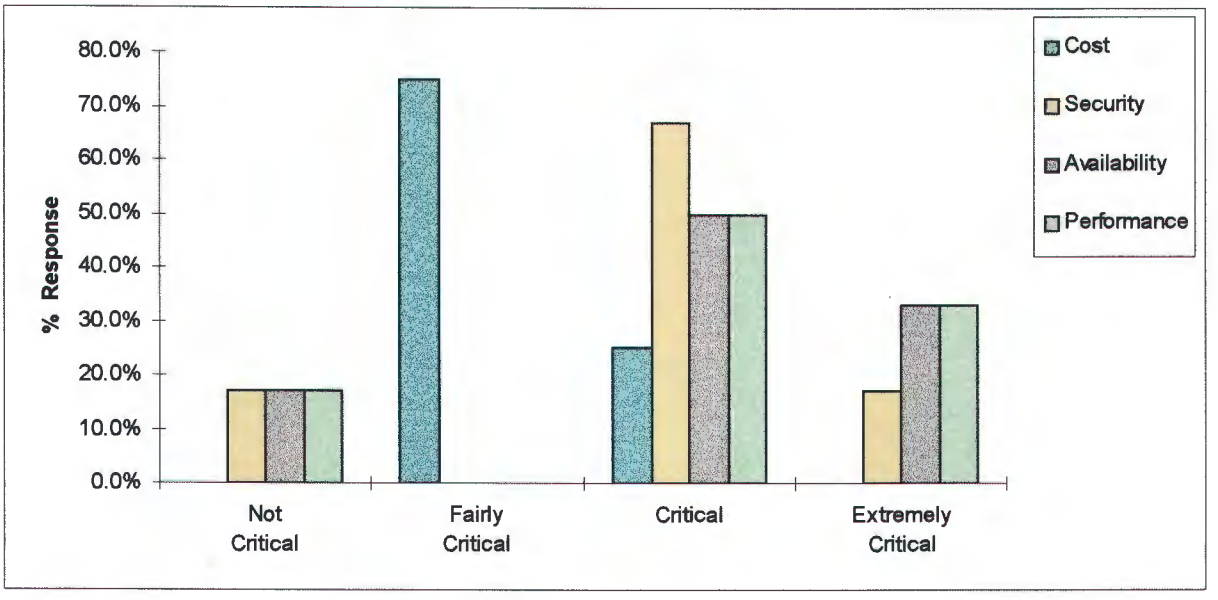


**Figure 2.5** Critical Factors of MIMS

**Available Skills:** less than 50% of the employees involved in the original development and maintenance of the system is still employed by SASOL and the same percentage with the necessary skills to maintain the system is still employed by SASOL. It is extremely difficult to find people with the appropriate skills to maintain the system in the external market.

**Uniqueness:** the functional aspects of MIMS are of medium complexity whereas the technical aspects are of high complexity. Responses received with respect to the functional aspects of the system varies from not unique to extremely unique to SASOL and packaged software exists with similar functionality.

**Maintenance History:** corrective maintenance is required fairly often but is not limited to certain parts of the system.

## 2.6 Summary and Interpretation of Results

Before the results are summarised and interpreted, the reason for the widely distributed answers obtained in certain areas should be given. The involved legacy systems support a variety of functions within an application domain and respondents could answer a question with regard to the

single function or set of functions that he[1] was primarily concerned with. Consider for example the widely distributed answers obtained regarding the criticality of availability for the MMS System (Table 2.1). Availability is critical for the receiving function of the MMS System as material can be delivered at any time of day or night and the system has to be available to allow for the receiving of the material. For other functions of the system (e.g. purchase ordering), however, availability is not that critical. A respondent primarily concerned with the receiving function might therefore regard availability as critical whereas a respondent concerned with purchase ordering might regard availability as not critical.

As indicated in Figure 2.6, the legacy systems which exist within SASOL are relatively large. They consist of an average of 574 programs each. On average 22% of these programs are batch. The average size of the databases is 4 351 MB with an average of 29 191 transactions per day. The average monthly growth percentage of the databases is 9,75%. The systems have an average of 617 users.

All legacy systems considered are at least integrated with one other legacy system as well as with various LAN or PC based systems. All the systems have been in use in a production environment for at least six years. The GL, PAMM and MIMS systems are used by SSF. If considered that according to SASOL's annual report for 1995, the total profit for SSF was R709,8 million rand, then the total annual cost to run only three of their systems amounts to 2,7% of the profit made in 1995. The annual cost to run the MMS and FS systems of SASTECH amounts to 10,7% of SASTECH's total profit for the year 1995. In the light of these facts it is clear that the costs involved to run these systems are relatively high. The costs involved with the systems are illustrated in Figure 2.7.

[1]The masculine form of the third person is used throughout to represent both genders.

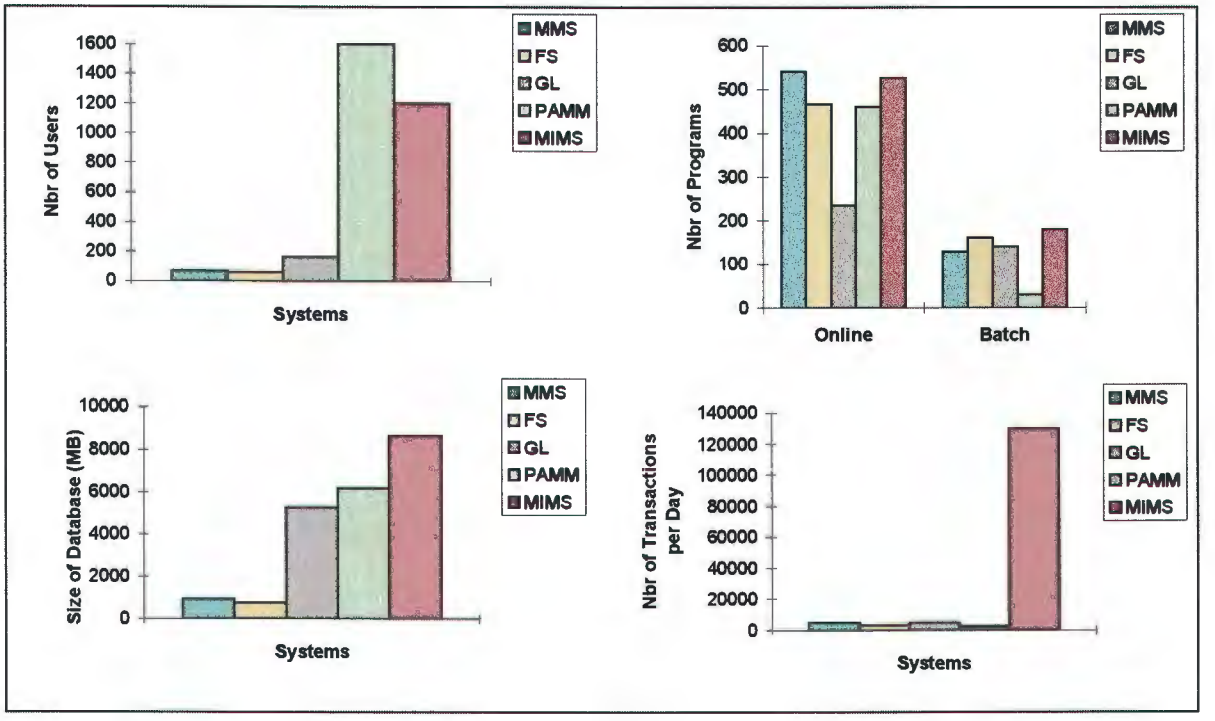


**Figure 2.6** Size of Systems

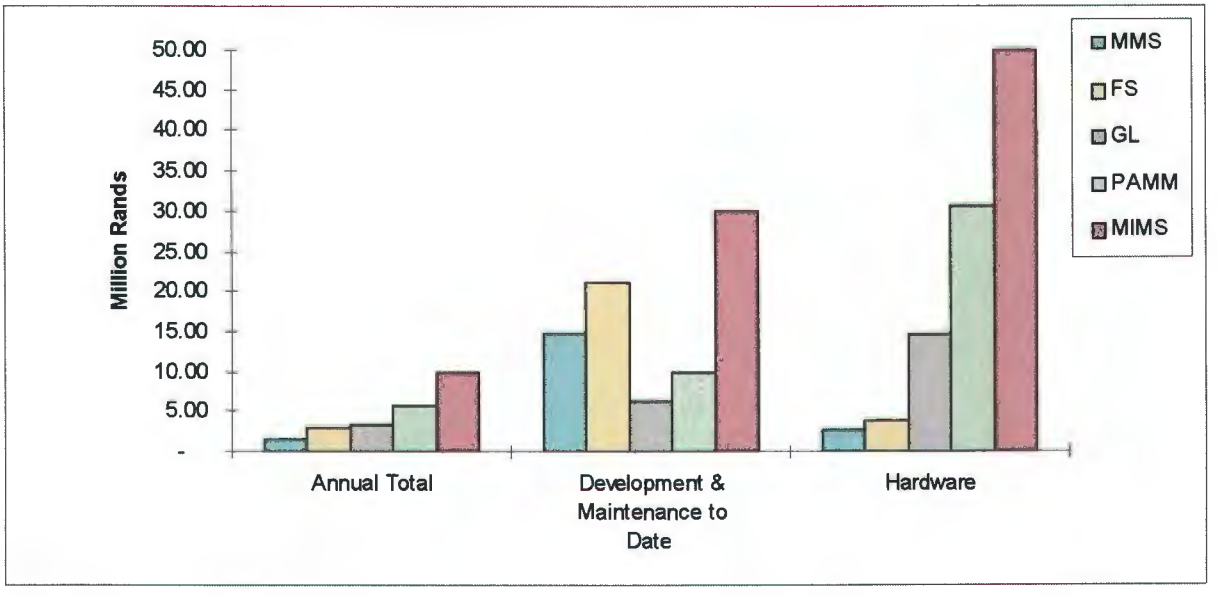


**Figure 2.7** Costs of Systems

As indicated in Figure 2.8, the quality of the systems are generally satisfactory, except for the MIMS System where most respondents indicated that the quality of the system was unsatisfactory.

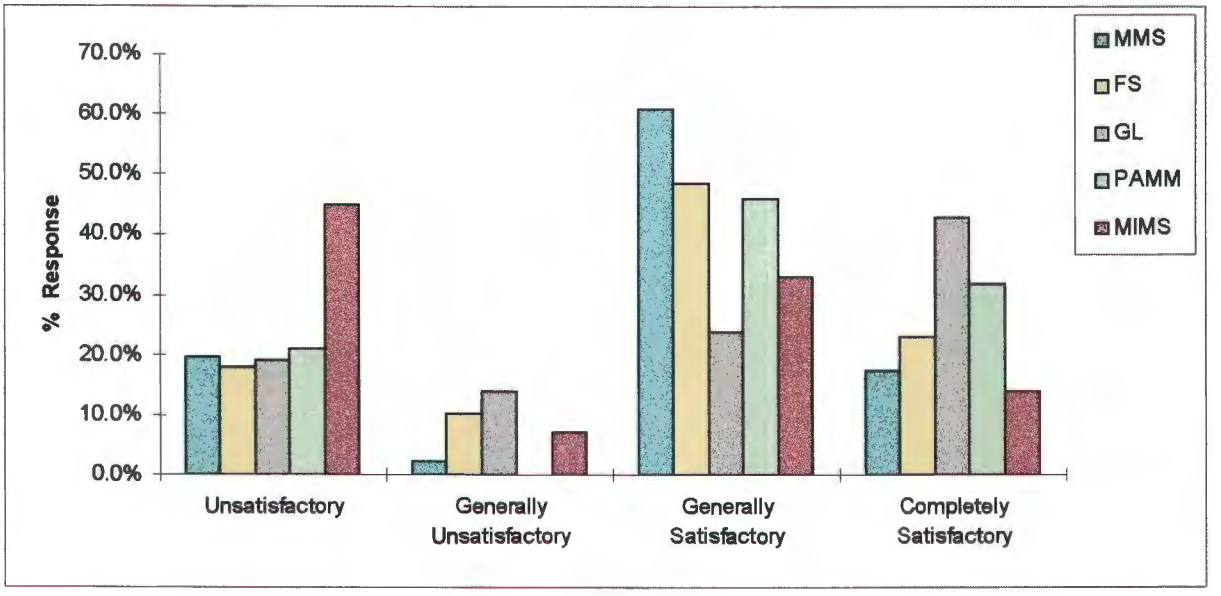


**Figure 2.8** Quality of Systems

Figure 2.9 summarises the critical factors of the five legacy systems. System performance and availability were considered most critical for all systems, whereas cost was the least critical.

All systems provide satisfactory support to SASOL's business and all systems, except the GL System, are unanimously considered to be mission-critical. Figure 2.10 illustrates the business support provided by the involved legacy systems. People with the appropriate skills to maintain the systems are usually fairly difficult to find. The PAMM System is the exception here as it is very easy to find people with the appropriate skills to maintain this system within SASOL. The ease with which the necessary skills for maintaining the involved legacy systems can be attained is indicated in Figure 2.11.

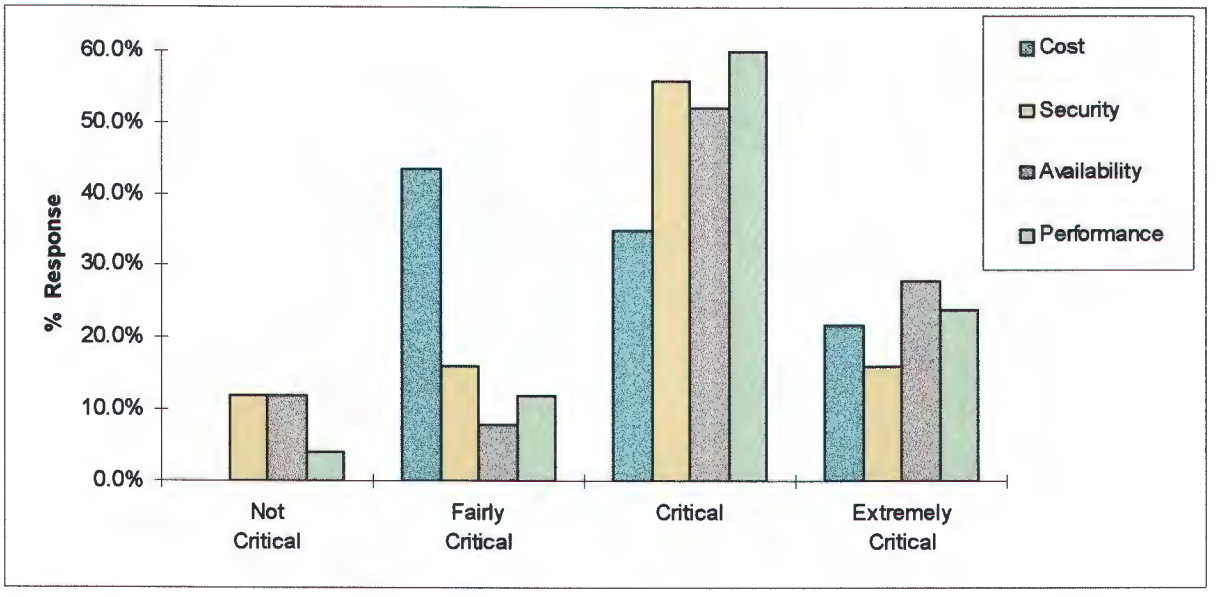


**Figure 2.9** Critical Factors of Systems

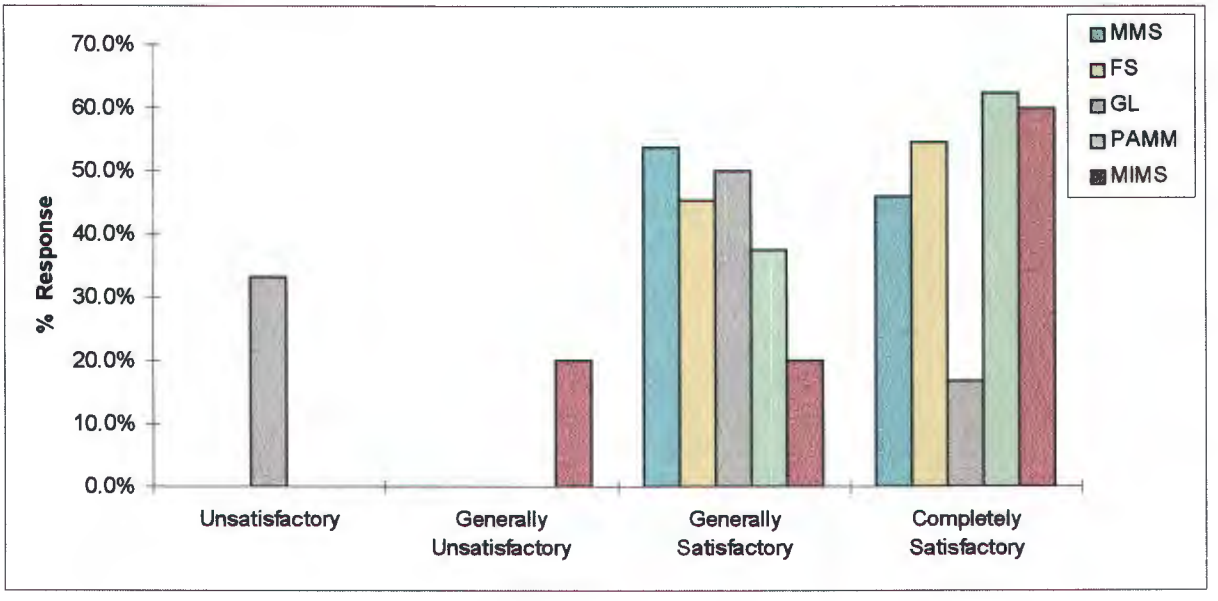


**Figure 2.10** Business Support of Systems

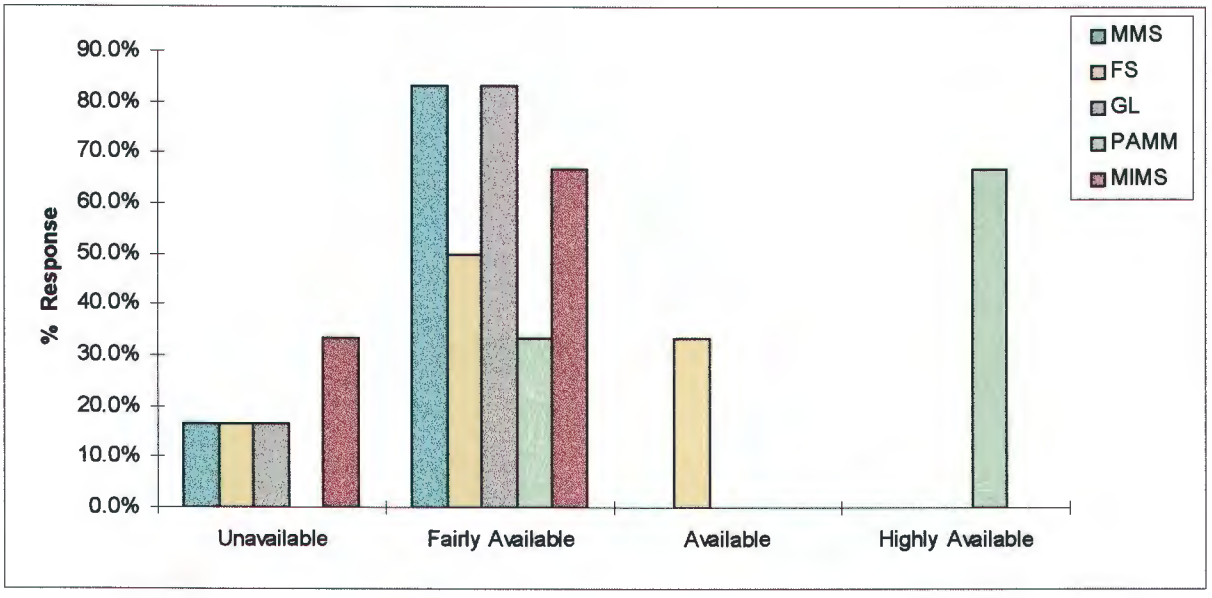


**Figure 2.11** Skill Availability of Systems

As indicated in Figure 2.12, the systems are of fair to medium complexity. The uniqueness of the legacy systems is illustrated in Figure 2.13. In some cases packaged software with similar functionality exists (the GL and MIMS systems), whereas in other cases the existing packaged software does not satisfy total requirements (the MMS, FS and PAMM systems). Some respondents commented that it was easier to adjust business processes to packaged software than vice versa. As indicated in Figure 2.14, corrective maintenance is not required very often on any of the systems. With some of the systems maintenance is limited to certain parts (the MMS and PAMM systems) while with the other systems it is not limited to certain parts.

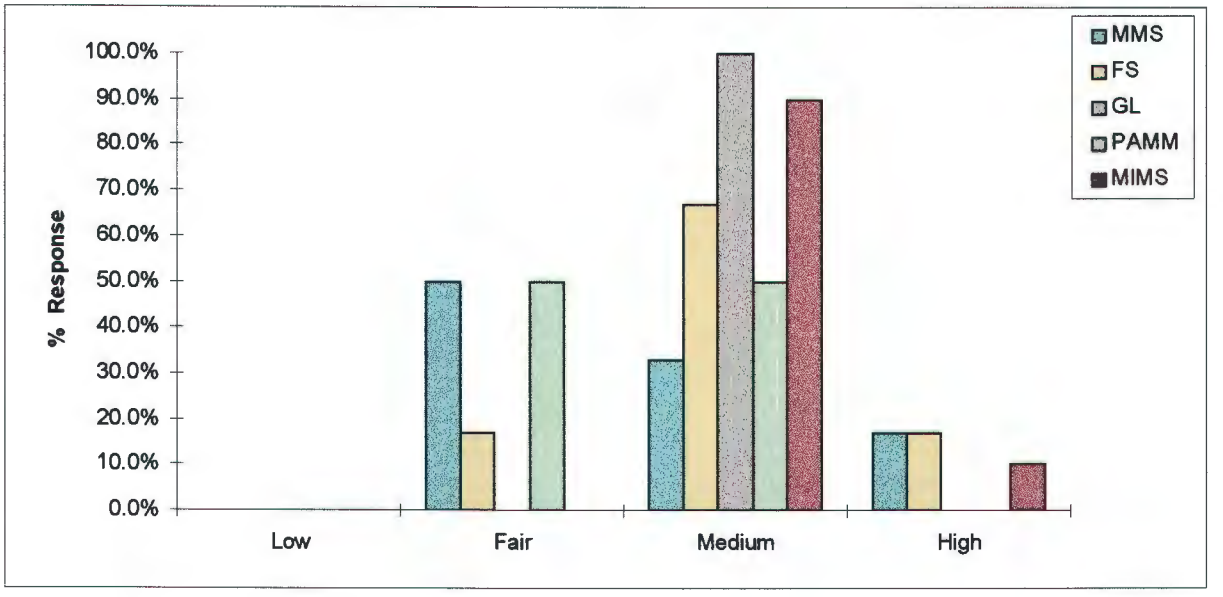


**Figure 2.12** Complexity of Systems

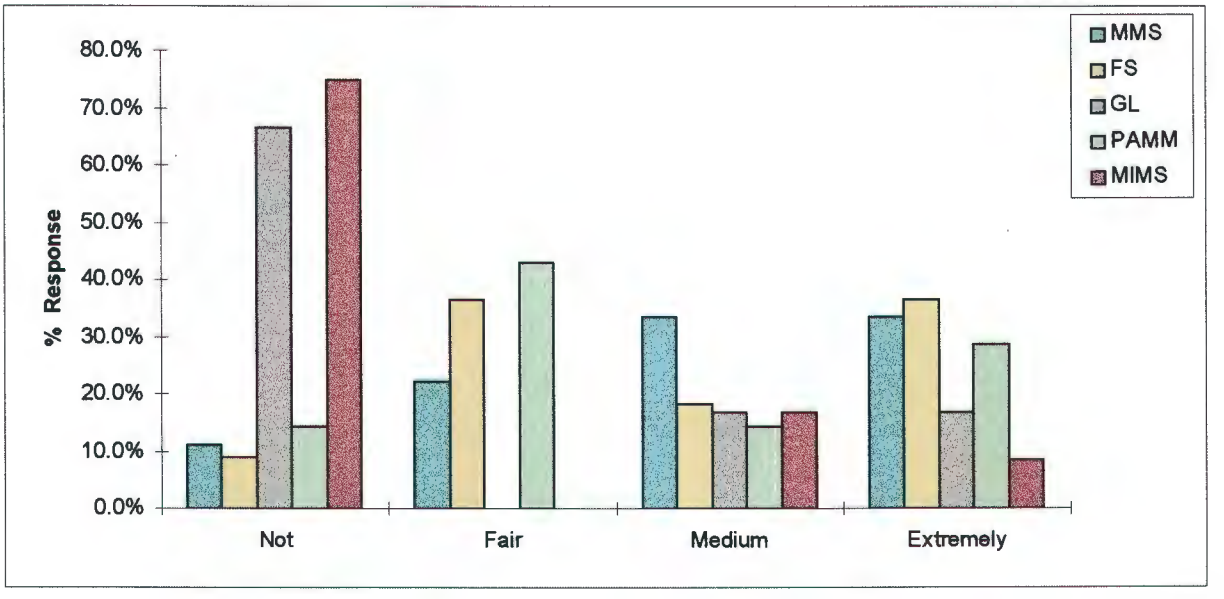


**Figure 2.13** Uniqueness of Systems

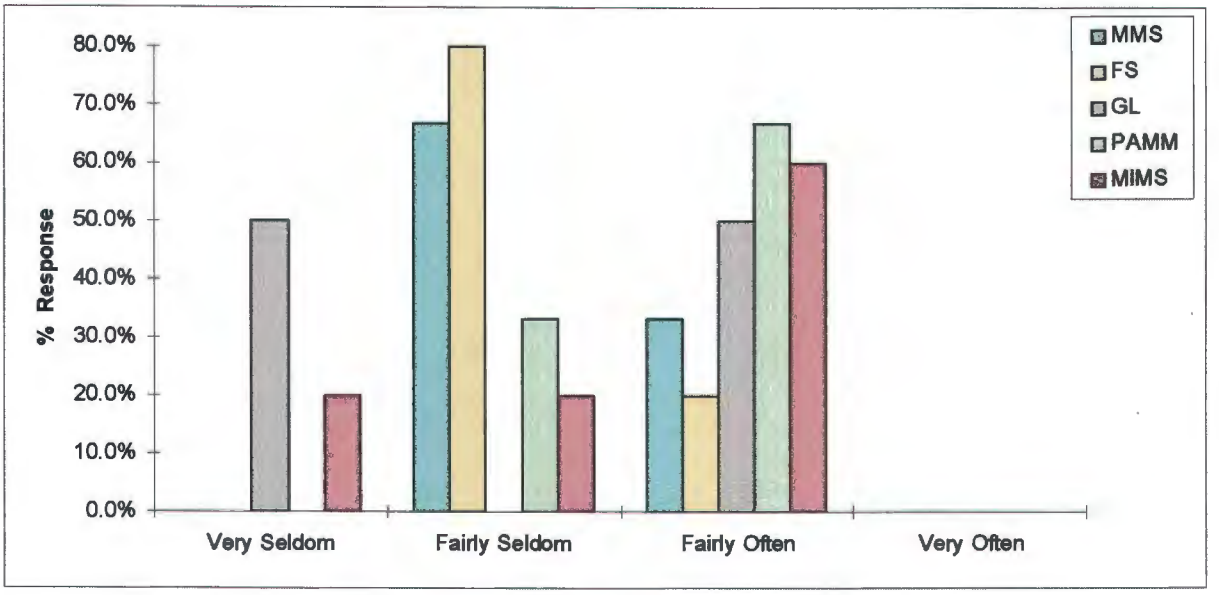


**Figure 2.14** Maintenance History of Systems

## 2.7 Trends in the IS/T Industry

Trends in the IS/T industry which are applicable to integrating legacy systems with client/server environments include downsizing, re-engineering, object-orientation (OO), middleware, standards, open systems and graphical user-interfaces (GUI's). Each of these trends are reviewed next, outlining the implications for dealing with legacy systems.

### 2.7.1 Downsizing

Downsizing involves the moving of mainframe-based applications to more cost-effective mid-range and minicomputer platforms without significant change. A driver for downsizing is the expensive proprietary mainframe million instructions per second (MIPS). In the integration of a legacy system with a client/server environment, there is usually some functionality or data that has to be off-loaded from the mainframe and which can be viewed as downsizing.

The greatest challenge of downsizing is the transition from the old to new approaches. The mainframe systems have to maintain the business support as usual, while skills migrate to

client/server. During the transitional period the investment in client/server environments has to increase while the investment in the mainframe environment has to be maintained. This has major cost implications, as the cost of a mainframe-only environment is less than that of a mainframe combined with client/server environment. Savings in costs will therefore only be perceivable in the longer term (Xephon,1993).

In the IS/T industry the choice of platforms, tools and applications adequate for downsizing and client/server environments is expanding rapidly. Primary mainframe software vendors, including Computer Associates, Dun & Bradstreet, Lawson Software Inc and Pilot Software, are moving their applications and tools to Unix. The mainframe vendors IBM and Amdahl have announced hardware and software to ease LAN-mainframe connectivity.

### 2.7.2 Re-engineering

Re-engineering is the process of reverse engineering followed by forward engineering. Reverse engineering is the process of recreating design documents or even specifications from source code. Forward engineering is the usual development process within the context of re-engineering that proceeds from specifications through design to code. Reverse engineering therefore takes the product from a lower level of abstraction to a higher level of abstraction whereas forward engineering takes the product from a higher level of abstraction to a lower level (Schach,1996).

Re-engineering can be used to provide legacy systems with a migration path to client/server environments. It aims at expressing existing legacy code in a clearer form - ideally at a higher level, e.g. in a 4GL-like environment which is easier to understand and therefore easier to maintain. The application is then taken forward again from the 4GL-like environment with the client/server platform in mind. Sneed (1995) highlights important aspects when planning the re-engineering of legacy systems.

Re-engineering tools to achieve this goal are available (Xephon,1993). One of the simplest is a pretty printer (or formatter) which can help to display code more clearly. Tools to construct diagrams (e.g. flowcharts or structure charts) directly from source code, also exist (Schach,1996).

### 2.7.3 Object-Orientation (OO)

OO allows the interpretation of an application as a collection of objects that interact with one another, change as a result of those interactions and yet maintain identity through change. This allows for the creation of a library of standard objects that can be reused to build many different applications. Reusability improves productivity, saves money, shortens development time and increases system reliability. Software reuse is discussed in Section 5.3.5. The principles of OO are discussed in Appendix C.

OO can be regarded as the inherent basis of a client/server environment. OO seeks to model the "real world" in terms of objects which may request certain services from other objects as well as provide certain services to other objects. In the client/server environment the object making a request can be considered to be the *client*, whereas the object that receives the request and provides the service can be regarded as the *server*.

Wrappering allows for the integration of legacy systems and their data with the client/server environment without discarding their existing platform. By making use of wrappering, non-OO systems and data can be integrated with OO applications. Rumbaugh (1991) defines a wrapper as:

> ***"a class or operation that encapsulates a call to library routines or some other code that is being reused."***

The wrapper specifies the valid operations on the non-OO system. The non-OO system becomes just another object that can both send and receive messages.

Forge (1995) gives an example of interfacing successfully to a certain airline reservation system by means of encapsulating required sections of the code as objects by "wrapping" them in an object layer. These sections of legacy code were linked to a new OO application program as objects. This exercise was considerably more cost-effective than it would have been, had the legacy system been rewritten. The Forte' application development environment, Easel, as well as the PARTS Wrapper from Digitalk allow for the encapsulation of non-OO systems by making use of wrappering.

### 2.7.4 Middleware

According to Watterson (1995), middleware refers to a variety of different software applications for establishing communication between client applications and host servers. It is a kind of protocol which connects different GUI's (front-ends) and DBMS's (back-ends). In the context of integrating legacy systems with the client/server environment, a mainframe computer can be regarded as just another host server if the appropriate middleware is available for establishing communication to the client application.

As a result, an application programmer does not have to be concerned with the details of making the physical and logical connection to the target data source as the middleware will take care of that. Three categories of middleware are distinguished (Watterson,1995):

1. network or messaging middleware shields programmers from networking details (but not from database details) and includes remote procedure call (RPC) and interprocess communication (IPC);

2. database middleware shields front-ends from the technical details of the database back-ends and includes application programming interfaces (API's) and gateways. The middleware translates incoming requests into database-specific commands.

   API's are an agreed-upon set of functions that perform common tasks such as opening a connection with the database and finding out the names of databases and tables. The most commonly used database API middleware is open database connectivity (ODBC) which was originally developed by Microsoft for Windows.

   SQL gateways also provide links between client applications and back-end databases, although they are proprietary and DBMS specific. They usually provide gateways to both SQL (relational) and non-SQL legacy data. Examples include Sybase's Open Server and Oracle's SQL Connect;

3. replication server middleware is used to copy databases or parts of databases. It can usually be programmed to replicate copies of the data either at fixed intervals, when an event occurs, or on demand.

### 2.7.5 Standards

A standard is a point of reference. It is an industry-accepted way of doing business that does not impede innovation or represent the narrow interests of any one single vendor or user. Reasons for the creation of standards are for variety control, usability, compatibility, interchangeability, portability and interoperability (Schroen & Meltz,1992).

**Usability** refers to the provision of the same look and feel to different applications across different operating system environments.

**Compatibility** is the capability to use applications and subsystems over time. Backward compatibility ensures that later releases of products are capable of accessing, interpreting and presenting information prepared with earlier versions.

**Interchangeability** allows a free choice of applications developed to the standards from the market, therefore giving the user a safer investment.

**Portability** refers to the capability of moving software applications from one operating environment or platform to another at minimal cost. When environments become more uniform, the cost of moving should decrease dramatically (Simonds,1992) as the need to rewrite applications that may run on multiple platforms will be eliminated.

**Interoperability** refers to the ability to interconnect computers that use different operating systems so that they can exchange information. At the most basic level, it involves the ability to transfer files, exchange electronic mail and conduct remote login sessions from one system to another. At the most sophisticated level, it may involve the ability to distribute an application between computing systems in such a way as to be transparent to the user (Simonds,1992).

De facto and de jure standards are distinguished. A de facto standard is the term, applied to a product or system from a vendor that has captured a large share of the market and which other vendors tend to emulate, copy or use in order to obtain market share.

A de jure standard is created by a formally recognised standards developing organisation. The International Organisation for Standardisation (ISO), Comite European de Normalisation (CEN), American National Standards Institute (ANSI) and the Open Software Foundation (OSF) are respectively examples of international, regional, national and consortia (associations between companies, vendors or users) standards developing organisations.

A de jure standard is developed under rules of consensus in an open forum, in which everyone has a chance to participate. De jure standards cannot be changed without going through the consensus process monitored by the standards developing organisation. The open systems interconnection (OSI) standard is an example of a de jure standard.

### 2.7.6 Open Systems

An open system is a set of standard relationships which enable different computers, subsystems, applications and system software to operate together (Wessels,1992). Open systems are designed to conform to standards which allow it to easily interface with products from other vendors. Wheeler (1992) defines open systems as

> ***"hardware and software implementations that conform to the body of standards that permit free and easy access to multiple vendor solutions."***

The objectives of open systems are to provide portability, scalability, interoperability and compatibility (Simonds,1992; Wheeler,1992). Portability, interoperability and compatibility are defined in Section 2.7.6. Scalability refers to the capability of changing the overall capacity of an application and utilising processing power as needed. This allows for expansion to meet the needs of a growing organisation as well as scaling down to fit independent business units in a decentralised organisation (Simonds,1992).

Wessels (1992) states that a system is not either "open" or "closed", but that the degree to which a system is open, depends on the degree to which the system conforms to industry standards. By analysing the openness of the interfaces that a system utilises, a system can be placed on a scale of openness.

The OSF is a major initiative in the IS/T industry to establish open systems. OSF aims at, among other things, establishing standards for an open operating system. Enterprise computing has the same objectives as open systems, but where the focus of open systems is on future systems, the focus of enterprise computing is on existing proprietary systems. Enterprise computing offers a means to overcome a legacy of incompatible systems whereas open systems have the same goals but accomplish this task by establishing standards for new systems (Marion,1994).

### 2.7.7 GUI's

A GUI refers to a user-friendly "point and click" type of user-interface usually present in a client/server environment. By simply selecting the application of choice from a set of graphical images (icons) on the screen, the program is executed. The user may select options or input data by using pop-up menus, buttons and scroll boxes.

OO user-interfaces such as Macintosh, are a form of GUI in which icons represent real-world business objects. The user instigates system actions by direct manipulation of the icons. For example, a product item can be added to a customer order by selecting the icon which represents the item and "dragging and dropping" it onto the icon representing the order. It is claimed that such interfaces have a closer correspondence to the end-user's mental model of the application than conventional GUI's (where icons represent application functions) and are hence easier to learn and therefore more efficient and accurate in use.

The development of GUI's has resulted in standards to be developed for application program interfacing. Standardisation of GUI's allows users to make a quick transition to different GUI applications. Microsoft's Windows is a very popular GUI which has captured a large share of the

GUI market. Other GUI's include OS/2 Presentation Manager from IBM as well as X-Windows for Unix platforms (Marion,1994).

## 2.8 Client/server Environments

McFadden and Hoffer (1991) define a client/server environment as:

> ***"a co-operative processing environment in which the logic of an application is divided between a front-end computer (the client) and a back-end computer (the server). The client generally manages the user-interface and other user-specific computations, while the server provides database management and related functions."***

Client/server computing refers to the relationship between two processes which are cooperating in the performance of some task. The client requests that some function be performed and the server must perform the function. A network connects the client to the server. In the world of computing clients and servers are executable programs (Loftus et al,1995).

A client/server architecture allows developers to take the best advantage of the different types of computers available. The user-interface can be placed on a PC which offers access to information in an efficient, intuitive manner. Client/server systems may also be developed without a GUI. Each workstation requires less memory since a complete copy of the DBMS is only required at the server. The central server containing the databases can have the appropriate security.

The network is one of the most important elements of a client/server architecture. Client/server applications use the network infrastructure to distribute processing appropriately and to provide users with user-friendly interfaces, quick response and access to data and applications. Network traffic is minimised since only qualified data is passed through the network. Network traffic can be further reduced by storing highly used and non-volatile data at individual workstations.

Systems can be built for a fraction of the cost by making use of off-the-shelf software components. Kavanagh (1995) identifies good candidates for the client/server architecture:

- data entry and editing systems e.g. order entry;
- interdepartmental work flow systems;
- sales and marketing information systems;
- information access and presentation systems in support of strategic and tactual decision making e.g. executive information systems (EIS's) and decision support systems (DSS's);
- compute-intensive systems e.g. scheduling;
- human resources systems;
- financial, mathematical statistical and pricing analysis systems;
- professional support systems e.g. Computer-Aided Design (CAD), engineering and medical.

Existing techniques and tools for client/server systems are not robust and mature enough for large transactional systems. As a result transactional systems are usually poor candidates for a client/server approach. There is a lack of client/server tools to control large system development projects (Redelinghuys & Nienaber,1994). Client/server tools, such as 4GL's with GUI's allow for a rapid prototyping approach to application design. The type of system being built is often suitable for evolutionary development. Many smaller client/server systems can be developed entirely with rapid prototyping, but large transactional systems development, such as the legacy systems involved, need a complete methodology due to immature client/server tools and inexperienced developers.

A client/server system can, however, be used as a front-end to a transactional system, e.g. for order entry. Kavanagh (1995) identifies poor candidates for total client/server architecture:

- very large or complex systems;
- systems with high-volume centralised I/O processing;
- systems that require centralised control and security;
- systems which are tightly integrated with other legacy systems.

Due to their transactional nature and tight integration with each other, the five legacy systems of SASOL may consequently be considered poor candidates for total client/server architecture. A discussion of this matter is outside the scope of this investigation and it is sufficient to say that an integration project involving one of these systems will have high inherent risk.

Organisations are considering client/server technology to improve productivity and efficiency as well as optimise the support provided by IS/T resources for business needs. A client/server solution provides the flexibility of PC application development with mainframe-level control over data. It provides a decentralised architecture that enables users to gain transparent access to information within a multi-vendor environment.

The mainframe still has a role to play in client/server environments. Database vendors, e.g. Sybase, provide integration facilities and products for the mainframe to become a super-server (Xephon,1993). Integrated CASE tools from Andersen Consulting and Texas Instruments are being announced which support both client and server development with the same CASE tool repository. Conversion tools for transporting custom CICS Cobol code to other platforms are already available and IBM is porting its CICS transaction processing environment to both IBM Unix and HP-Ux.

## 2.9 Strategies for Integrating with Client/server Environments

In order to decide on an integration strategy for a legacy system, the business processes supported by the legacy system as well as the technology supporting the legacy system need to be investigated. For the integration of a single legacy system, multiple approaches may need to be combined. Eight approaches were distinguished (Simonds,1992; Xephon,1993; Forge,1995; Kavanagh,1995) and are reviewed below.

### 2.9.1 Eliminate Legacy System

Eliminating the legacy system involves abandoning, outsourcing or manually performing a function. This approach may be followed if the purpose of a legacy system is part of the history of an

organisation. An inventory of a legacy system may reveal programs that are never used or have a few users who could use a different program to obtain the same results, as well as reports generated by the system which are never read.

### 2.9.2 Redesign Business Process

This approach involves analysing the business and designing new procedures and systems to support current needs. Many legacy systems support business processes which are neither necessary nor efficient. These business processes will need to be examined and streamlined. This will often result in either a change of scope or a reduction in the scope of the legacy system which supports the business process.

### 2.9.3 Replace Legacy System with Packaged Software

If a function is not unique to an organisation, similar solutions must exist (Xephon,1993). Packages have become more powerful, easily customised, accessible from other applications as well as available for client/server environments. As commented by some respondents of the questionnaire, it may be advantageous to redesign business processes around a state-of-the-art package solution. Customising the packaged software to support business needs can, however, be very difficult.

### 2.9.4 Redesign and Develop Legacy System

Redesigning and developing the legacy system with new client/server tools involves the building of a new client/server system to replace the legacy system (Xephon,1993). If the business processes have changed to such an extent that the existing legacy system is obsolete and no packages with adequate functionality exist, the system will have to be redesigned and developed.

### 2.9.5 Downsize Legacy System

Downsizing is regarded as a first step in integrating a legacy system with a client/server environment. However, the downsized system will not be a client/server system unless it is restructured, as all processing will still occur on the server. A "move off and then replace" downsizing integration strategy provides for a more gradual integration path. This will also allow for more flexibility in the transition of skills to client/server.

Downsizing the legacy system involves the moving of the current software to a new platform without significant change (Simonds,1992). Software products which can run the same application code on a variety of platforms do exist, e.g. CA's Datacom/Ideal and Oracle's Oracle. A very successful downsized project has been completed at SASOL. The project involved the downsizing of a legacy application in Datacom/Ideal on an IBM mainframe computer to a Unix HP9000 platform.

Complications with downsizing may include one or more upgrades of the legacy system to the current version before it can be downsized. The code needs to be analysed with respect to screen management, database and file management, add-on function libraries as well as access to system features in order to prepare an integration plan. Downsizing was discussed in Section 2.7.1.

### 2.9.6 Renovate Legacy System

Renovating the legacy system implies keeping the software on the current platform while improving its appearance or maintainability as appropriate (Simonds,1992). The user-interface, database as well as business rules may be renovated. As in the case of downsizing, the renovated system will not be a client/server system unless it is restructured into client and server components. The restructuring of legacy systems into client and server components is discussed in Section 2.9.8. Disadvantages of this approach include the continued mainframe costs and the fact that the legacy code still needs to be maintained. This approach can, however, be a good integration stage as a component of a larger strategy.

Without changing the original application code, a tool can be used to improve the appearance of the user-interface. For on-line systems, several tools use the IBM Ehllapi interface to access the system through a dumb terminal, allowing the development of graphical front-ends. This enables legacy applications to conform to the standards which users of newer systems expect, e.g. Micro Focus Application-to-Application Interface and Wall Data Ramba. Wall Data's Ramba can work with Powerbuilder and other client/server development tools to access mainframe screens. It acts as a DDE server and can therefore be integrated into many applications.

The database can be refurbished by cleaning up the data and making it accessible to other applications. Legacy data can be missing or corrupt, data items can be embedded in nonexplicit ways and the data can require business rules for interpretation. Making it accessible may involve integrating to a relational database or using timed replication to copy the data to such a repository.

Renovating the application code involves reverse engineering of the data and process models, design recovery in order to recover business rules as well as recovering screen design and transaction integrity. Typical items to be recovered for a mainframe system include COBOL items, JCL and database catalogs.

Tools to assist in understanding the system include Visual Reengineering Toolset and Visual Testing Toolset (McCabe,1990) which compile metrics and group non-OO code into classes as well as the Viasoft products (VIA/Insight and Renaissance) which analyse the code, i.e. flagging bad statements and unexecutable code. Tools to improve the system include Compuware XA and Knowledgeware LT1 which "beautify" and restructure the code. Tools which reverse engineer the system for replacement include those available from Cadre and ProCASE which recover the design of a system into structure charts or data models. This is an extremely complex task.

### 2.9.7 Restrict Legacy System

This approach involves keeping the legacy system, while all new development is done in the new environment. It implies an incremental integration to a client/server environment. Continuous or periodic data exchange between the new system and the legacy system is established while the use

of existing hardware, software and staff are continued and all new technology capabilities are added incrementally (Simonds,1992). The Evolutionary Technologies toolset or the Prism Warehouse Manager can be used to perform the extracts on the mainframe.

### 2.9.8 Restructure Legacy System

Restructuring a legacy system involves the modifying of the application code to reside on client and server components. Forge (1995) identifies the following steps in restructuring a legacy system:-

- Clearly identify aims and targets in business terms of new functionality, performance or reduced costs.
- Gather as much information as possible on the application, particularly its high-level application logic and the business of reasoning originally behind it.
- Analyse the application logic and identify key functions and the processes that execute each function. In order to restructure the code a strong differentiation of functions is necessary, e.g. an aircrew scheduling system identified the functions of local data access, data storage and update as the key functions, and restructuring was done accordingly.
- Compare the original application logic and verify that it maps to the current business logic. If not, this could be the appropriate time to change it.
- Determine which parts of the application could be restructured into client and server components.
- Ensure the dispersed parts are modular in coding, e.g. front-end user-interface functions for the client and back-end database application functions for the server. If the existing code is not modular, it will have to be rewritten.

Object wrappering can be very useful in the restructuring approach. Object wrappering is the encapsulation of an existing system and then restricting its interface to a set of inbound and outbound messages. Object wrappering was discussed in Section 2.7.3. The maintenance on the legacy system will need to be done in the traditional manner and it will not benefit from the characteristic advantages of OO.

## 2.10 Summary and Conclusion

The major problems of mainframe environments are high maintenance costs, inaccessible data and a lack of user-friendly graphical user-interfaces. These environments are stable and mature and their strengths include batch processing and automated job scheduling.

The questionnaire yielded the most dominant characteristics of the five identified legacy systems. They are:

- a relatively high number of daily transactions;
- tight integration of legacy systems with each other;
- large capital investments;
- high annual costs;
- high quality;
- critical availability and system performance;
- providing excellent business support;
- medium complexity.

Various integration trends were highlighted. They were downsizing, re-engineering, OO, middleware, standards, open systems and GUI's. **Downsizing** a legacy system involves the moving of the existing software to a new platform without significant change. It can be considered a first step in integrating a legacy system with the client/server environment. **Re-engineering** is the process of recreating design documents or specifications from source code, followed by the usual development process proceeding from specifications through design to code. Re-engineering aims at expressing existing legacy code in a form which is easier to understand and maintain and which can then be re-developed according to the requirements of the client/server environment.

**OO** allows the interpretation of an application as a collection of objects that interact with each other by means of messages, change as a result of those interactions and yet maintain identity through change. By making use of wrappering, a legacy system becomes an object which can both send and receive messages and which can be integrated with the client/server environment without

discarding its existing platform. The term **middleware** refers to a variety of software applications for establishing communication between different client applications and host servers. A mainframe computer can be regarded as just another host server if the appropriate middleware is available for establishing communication to the client application.

A **standard** is an industry-accepted means of doing business that does not impede innovation or represent the narrow interest of any one single vendor or user. Standards are essential for ensuring usability, compatibility, interchangeability, portability as well as interoperability and should be adhered to when integrating a legacy system with the client/server environment. **Open systems** are designed to conform to standards in order to allow easy interfacing with products from other ıdors. As a result, open systems have a high degree of portability, scalability, interoperability as ll as compatibility.

GUI is the term used to refer to a user-friendly "point and click" type of user-interface usually sent in a client/server environment. The major weaknesses of client/server environments include lack of both mature client/server techniques and tools as well as experienced professionals. engths include increased cost-effectiveness, improved productivity, excellent graphical user- :rfaces as well as reduced network traffic.

ferent approaches (which may be combined into a single strategy) for integrating legacy systems l client/server environments were reviewed. They included eliminating the legacy system, esigning the business processes which the legacy system supports, replacing the legacy system h packaged software, redeveloping the legacy system, downsizing the legacy system to a new tform, renovating the legacy system, restricting the legacy system as well as restructuring the acy system into client and server components.

a last comment it should be emphasised that decisions regarding integrating legacy systems with client/server environment should aim at solving business problems and should not only be done the sake of technology.

# CHAPTER 3

# METHODOLOGIES FOR SOFTWARE DEVELOPMENT

## 3.1 Introduction

A methodology for software development provides an overall systematic approach to the production and improvement of software, using a collection of methods and techniques, each with their predefined notational conventions and representation schemes. A life-cycle methodology consists of methods to support each of the phases of the life-cycle of software development. It is usually presented as a series of steps, with techniques and representation schemes associated with each step.

This chapter is devoted to methodologies for software development. The basic concepts of a generic methodology are identified. They are the paradigm, representation schema, methods, techniques, procedures and deliverables. The various perspectives supported by existing methodologies are described after which the roles of the software process model and software development life-cycle (SDLC) are reviewed. The different levels of the Capability Maturity Model (CMM) (Humphrey,1989) are explained in the final section of this chapter.

## 3.2 The Basic Concepts of Methodologies

A generic methodology is conceptualised by means of a meta model in Figure 3.1. The building blocks of software methodologies are now explained in general terms with examples to illustrate their roles.

A **paradigm** is an abstract or intellectual model or pattern on which the methodology is based, e.g. the structured paradigm and the object-oriented (OO) paradigm.

The **representation schema** based on some paradigm, should provide a precise, consistent, unambiguous and semantically complete notation for the graphical representation of the system under development. It provides a means of effective communication between the user and the system developer as well as between system developers. The representation schema contributes to the structuring of the knowledge about the system under development, it is usually easy to

understand and helps in correctness and consistency checking. Examples include activity graphs, data structure diagrams and data flow diagrams (DFD's).

**Methods** are explicit, orderly, definitive prescriptions for achieving an activity or set of activities and are usually based on a paradigm. A way of carrying out the work steps of a complete phase of software development, e.g. design or integration, is often termed a method. It is implemented by utilising procedures, techniques and tools. Examples include structured and OO design (OOD).

**Techniques** are used by methods. The term technique refers to a part of a methodology which may employ a well-defined set of concepts and a way of handling them in a step of the work. A chnique is often used for a portion of a phase of software development. Examples of techniques clude entity relationship modelling, functional decomposition, data flow analysis and transition ets.

**rocedures** are manners of carrying out processes. It refers to the manual activity or set of ctivities and steps required to implement methods. An example is the procedure to perform quirements and specification, followed by design, implementation and integration usually commodated in any methodology.

**eliverables** are produced as a result of development tasks and activities and should be well-efined for each phase of development. Two types of deliverables exist, i.e. the software models id the documentation describing the software. Examples of deliverables include the software roject management plan (SPMP), the system design and the test plan.

utomated tools provide computer assistance to the methods and techniques of a methodology. xamples include online editors as well as a planning tool such as Microsoft Projects.

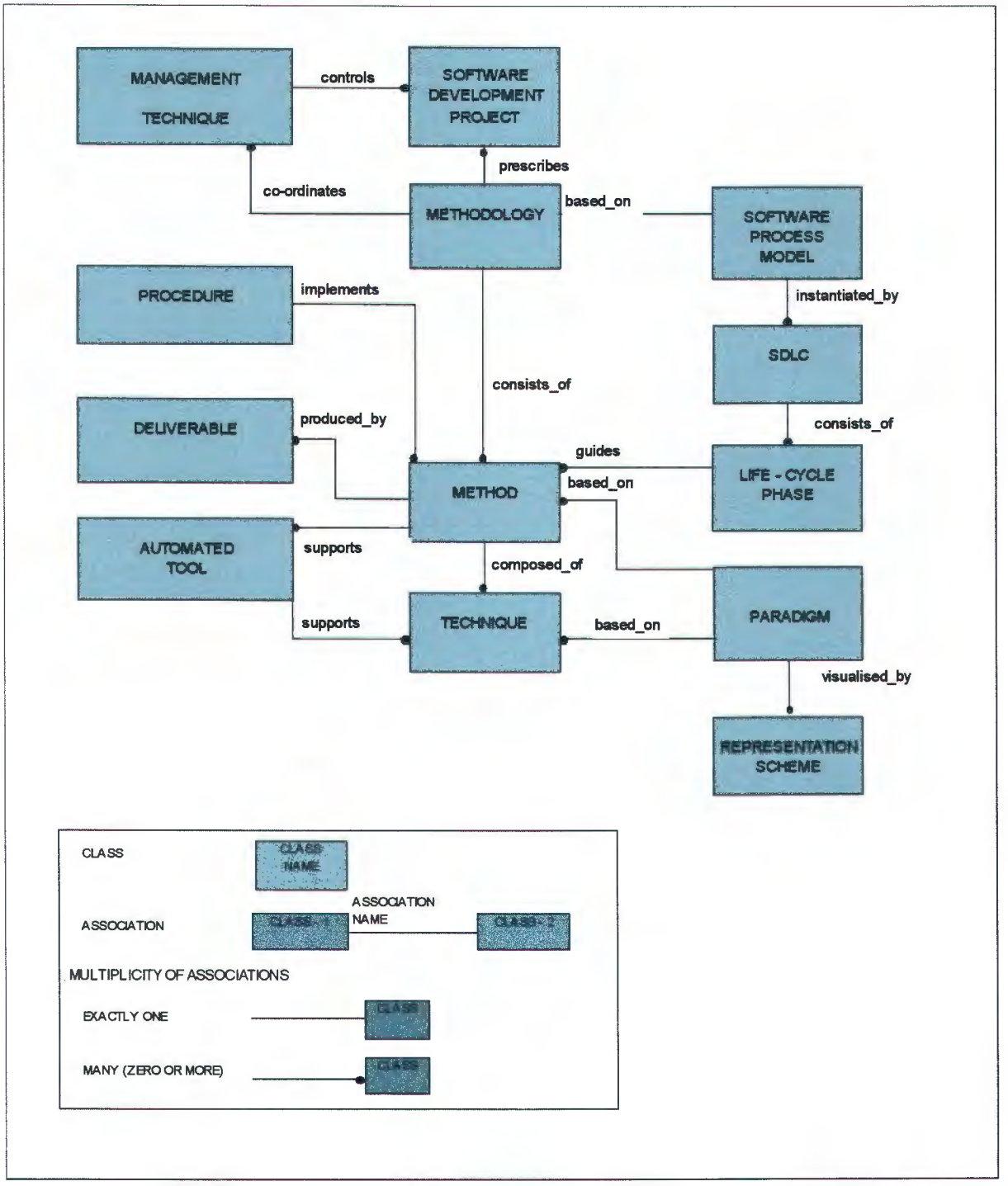


**Figure 3.1** A Meta Model for a Generic Methodology

## 3.3 Methodology Classification

Methodologies are not appropriate for all application domains, e.g. for some small, highly mathematical applications. A methodology which is appropriate for one situation may be inadequate for another as application domains differ and every project has unique characteristics. A major aspect of any methodology is the degree to which the information system's life-cycle is supported (Olle,1988, Conger,1994). If it is felt to be important to carry out information systems planning, then a methodology covering this phase will be preferable to one which does not. After experience with several methodologies, it is possible to think in terms of the underlying concepts and techniques. The approach to analytic data modelling promoted in one methodology may be ombined with the approach to data flow analysis in another methodology, and cross-referencing e results of the two analyses in a way not suppported in either.

Methodologies which are supported by the techniques of simulation and prototyping exist (Kavi,1984; Lantz,1985; Conell & Shafer,1989). Computer-aided simulation techniques are important for the verification of the feasibility and evaluation of the performance of the target system. Prototyping can complement the functions of a simulation facility and assist in evaluating if e requirements specification conforms to the user needs.

Prototyping allows for the rapid creation of a working model that is functionally equivalent to a subset of the system under development. It is useful for gathering requirements but is usually not intended to evolve into a final product. An evolutionary prototype is one with incomplete scope which is, however, intended for eventual production. Advanced automated tools exist to build a quick version of all or part of the target system. The speed and flexibility of developing prototypes offer the advantage that users can experience and test the application (in particular, the user-interface) early and allow changes to be made to a system based on their feedback. Prototyping gives executives a concrete model to look at, and it is efficient to access each version as often as needed to satisfy requirements. The interaction between the user and the software developer when prototype is involved, is often referred to as iterative prototyping.

A major strength of the rapid prototyping SDLC is that the development of the system is essentially linear, proceeding from the rapid prototype to the delivered system. (The role of the SDLC is discussed in Section 3.4.) Feedback loops (as in the Waterfall Model, Section 3.4) are less likely to be needed because it is reasonable to expect that the specification document will be correct as the working rapid prototype, which has been used to construct the specification document, has been validated through interaction with the user. Rapid prototyping is discussed in Section 5.3.3 within the context of the Enhanced Spiral Model for Integration (ESMI).

Some methodologies provide support for incremental and iterative development (Currit et al,1986; Selby et al,1987; Gilb,1988). Iterative development allows for the repetition of the development steps at progressively finer levels of detail. Each iteration adds or clarifies features rather than modifies work that has already been done. There is therefore less chance of introducing inconsistencies and errors.

Incremental development allows the target system to be developed in small, manageable increments, where each new increment is the previous increment with additional functionality. The target system is not viewed as a monolithic entity with a single delivery date, but as an integration of the outputs of successive steps of each iteration. Large systems are usually developed in increments because of their size.

When an application is developed without making use of a methodology there are no prescribed activities and reliance on the individual's experience and problem-solving ability is inevitable. A lack of methodology results in a lack of rigour and a trial-and-error problem-solving strategy.
The various perspectives which are taken in established methodologies for software development are now reviewed. Six classes of methodologies are distinguished, i.e. process-orientation, data-orientation, behaviour-orientation, cross-references between perspectives, meta and object-orientation (OO).

### 3.3.1 Process-oriented Methodologies

The earliest perspective to be recognised during the evolution of information systems methodologies was the process-oriented perspective (Gane & Sarson,1979; Yourdon & Constantine,1979). In the early days of data processing, the computer was regarded as a convenient and quantifiably cost-effective tool for performing specific processes, such as generating a payroll or producing a set of invoices. The need to analyse the process which the computer was required to perform was dominant, and the programming languages available for constructing the application programs reflected this emphasis. Many methodologies moved away from any emphasis on the computerisable process towards an analysis of the "real world" activity as performed in the business, the understanding being that this activity could conveniently be computerised.

Process methodologies (e.g. Method/1) require a structured, top-down approach to evaluating processes and the data flows with which the processes are connected. Documentation includes DFD's and data store definitions.

The process-oriented perspective can be broken down into four distinct aspects, the first two respectively relating to the business analysis and system design phases, and the remaining two aspects relating to the construction design phase:

1. business activity;
2. user perceivable task;
3. computerisable process and
4. a compilable unit of programming.

An activity is performed in a business area. Recognition of each activity is quite independent of the presence or absence of a computerised system. However, how it is performed may depend heavily on an information system.

A user perceivable task is a piece of work which is supported by a computerised information system. A list of tasks is typically to be found on a menu of such tasks, as presented to a user. The user understands these tasks and is able to select from the menu any of the tasks which he is authorised to initiate.

A computerisable process is an executable piece of software built by the developer, using whatever tools are appropriate. The computerisable process may support one or more user perceivable tasks, but the user does not have and does not need knowledge or understanding of the nature of the process. A typical computerisable process may consist of one or more compilable units of programming.

A compilable unit of programming is several lines of source code, prepared by a programmer using some programming language, which is prepared as part of the construction phase.

Most legacy systems (including the five which were investigated) were developed using either no methodology or a process-oriented methodology. The lack of a methodology can be regarded as the major reason for the introduction of errors caused by unrelated changes to legacy applications as well as the corruptness of legacy data.

### 3.3.2 Data-oriented Methodologies

The advent of database technology in the mid-1960's was responsible for the data-oriented perspective (Orr,1981; Warnier,1981; Jackson,1983). The radical data-oriented perspective seemed, in some cases, to ignore the significance of processes to be performed on the data.

The data-oriented perspective places emphasis on a complete and thorough analysis of the data and its relationships. After the evaluation of the data and their relationships, outputs are mapped onto inputs in order to determine processing requirements. There was an early emphasis on the provision of access paths to optimise performance of selected programs, but the acceptance of relational theory during the 1970's moved the consideration towards retrievability of all

information, independently of storage representation, and subsequently towards the expressibility of the integrity constraints which the data must satisfy.

The data-oriented perspective is seen to concentrate on the following aspects:

1. business data;
2. prescriptive, but construction and performance independent, data (sometimes called logical database design);
3. prescriptive data, taking into account the construction tool to be used;
4. stored representation of data, including indexes and access paths. This may be either the only option available with the construction tool, selected by the construction tool or, alternatively, selected by the construction designer.

These four aspects correspond to three phases in the information system life-cycle - the first to business analysis, the second to system design, and the third and fourth to construction design (Olle,1988). Documentation of data methodologies include entity relationship diagrams (ERD) and entity hierarchy diagrams. Data Oriented Design (DADES) is an example of a methodology with a data-oriented perspective.

### 3.3.3 Behaviour-oriented Methodologies

More recent trends in information systems methodologies focus on the need to analyse and understand events in the real world which impact data recorded in the information system e.g. the second part of Activity Modeling / Behavior Modeling (Gomaa,1986; Hatley & Pirbhai,1987).

It is often difficult to combine the behaviour and process-oriented perspectives in a single methodology. Application areas which are "event intensive" require careful analysis to determine which sequences of events are permissible and which are inadmissible. The behavioural perspective is important for the design of systems in certain computerised application areas, e.g. process

control. Events happening in the "real world", e.g. "Receipt of a sales order", can trigger a requirement for appropriate action in the computerised system.

Some existing information systems methodologies recognize the importance of the behavioural perspective and the problems it tackles. There is a preference for analysing business events as part of data analysis and for prescribing the events which can happen while the information system is running as part of the design of the computerisable processes. Only limited experience with the behavioural perspective exists.

### 3.3.4 Methodologies with Cross-references between Perspectives

These methodologies integrate two or more of the above-mentioned perspectives (Olle,1988), e.g. REMORA and MERISE. The balance of importance among the perspectives varies according to the specific methodology. REMORA supports the business analysis and system design phases. Basic concepts are those of object, operation and event. REMORA integrates the three perspectives, with some emphasis on the behavioural perspective. MERISE uses all three the above-mentioned perspectives. It combines an entity-relationship (ER) approach for data and a Petri net based approach for processes.

### 3.3.5 Meta-methodologies

A meta-methodology is not bound to any perspective and offers the possibility of customising a methodology for a specific application to be developed (Davis et al,1983; Du Plessis et al,1986; Wasserman et al,1986). Meta-methodologies are methodologies used to develop methodologies, e.g. System Descriptor and Logical Analyser (SDLA). Automated environments such as Problem Statement Language / Problem Statement Analyser (PSL/PSA), a Requirements Engineering Environment, is methodology independent and the environment has been customised for various methodologies. It is based on an Entity Relationship Analysis (ERA) type modelling schema.

### 3.3.6 Object-oriented Methodologies

Since 1967 a variety of techniques have been suggested to help solve the software crisis. Since that time there has been the awareness that the quality of software was generally unacceptably low and that deadlines and cost limits were not being met. The development of the structured paradigm led to significant improvements in the software industry with methods such as structured systems analysis, structured design, structured programming and structured testing. However, these methods were unable to manage software systems of increasing size and could not solve the problem that two-thirds of the software budget was being devoted to maintenance. This realisation lead to a continued search for alternative paradigms which could solve the essential problems of software development.

According to Coad and Yourdon (1991) OO concepts were first used by the development group of the Simula language in the late 1960's. In the 1970's these concepts were adopted by the Xerox PARC development team who developed the Smalltalk programming language. Although OO programming was used in these developments, OO analysis and design as known today were not used because the functional approach to system design was the norm and the OO approach was not established. Korson and McGregor (1990) as well as Schach (1993) describe the OO design paradigm as the next logical step in a progression which has led from a purely procedural approach to an object-based approach and to the OO approach. This progression is the result of a gradual shift in the point of view regarding the development process.

The reason for the limited success of the structured paradigm is that structured techniques are either process-oriented or data-oriented, but not both. The basic components of a software system are the processes and the data on which the processes operate. Some structured techniques, e.g. Data Flow Analysis (DFA), concentrate on the processes of the system, whereas the data is of secondary importance. Other structured methods, e.g. Jackson System Development (Jackson,1983), concentrate on the data of the system whereas the processes that operate on the data are of secondary importance. In contrast, the OO paradigm (Wirfs-Brock et al,1990; Rumbaugh,1991; Booch,1994; Capper et al,1994), consider both data and processes to be of equal

importance as an object is a unified software component that encapsulates both the data and the processes that operate on that data.

The essential concepts of OO are objects, classes, links, associations and messages. Central to the OO paradigm is the notion of an object. An **object** is a concept, abstraction or entity with crisp boundaries and meanings which is denoted by a name and consists of attributes and operations. The object *Person* may have attributes *name* and *age* and operations *change-job* and *change-address*. Objects communicate by means of **messages**. An object requests a service from another object by sending a message to the server object. A **class** is an abstract data type that describes a group of objects with similar attributes, common operations, common relationships to other objects as well as common semantics. An object is an instance of a class. A **link** is a physical or conceptual connection between object instances. An **association** describes a group of links with common structure and common semantics. A link is an instance of an association. These concepts are not described in detail in this dissertation but may be referenced in an acknowledged source such as Rumbaugh et al (1991). The principles of OO are summarised in Appendix C.

The OO paradigm facilitates a reduced level of complexity of a software system and simplifies both development and maintenance. OO designs are more likely to result in applications with desirable properties, e.g. modularity, information hiding, functional cohesion and minimal coupling (Appendix C). In addition, it promotes the reuse of objects. As a result, programmer productivity is increased, which results in a reduction of both development and maintenance costs as well as the implementation of reliable software systems.

Various OO methodologies were developed during the 1980's and 1990's (Booch,1986; Rumbaugh et al,1991; Monarchi & Puhr,1992; Berard,1993; Wilkie,1993; Booch,1994; Schach,1996). These methodologies require a top-down view of data objects, their allowable actions as well as the underlying communication requirement in order to define a system architecture. Well-defined graphical notation in the form of representation schema, based on the OO paradigm, allow for precise and complete descriptions of the perspectives taken of the underlying problem area.

Henderson-Sellers and Edwards (1990) proposed a seven-point methodological framework for OO systems development in which entity-data flow diagrams (EDFD) or information flow diagrams (IFD) are used as representation schema. The life-cycle is graphically represented by the fountain model. Entity-relationship diagrams, state transition diagrams and DFD's are respectively used in the Object Modeling Technique (OMT) of Rumbaugh et al (1991) for the representation of the object, dynamic and functional models. Booch (1994) describes a methodology for OO analysis and makes use of round-trip gestalt design, i.e. a style of design that emphasises the incremental and iterative development of a system through the refinement of different yet consistent logical and physical views of the system as a whole. Class diagrams, object diagrams, module diagrams, process diagrams, state transition diagrams as well as interaction diagrams are used to represent the underlying problem area.

The OO software development process is iterative and incremental because it is neither a strictly top-down nor a strictly bottom-up process. Each iteration allows the adding or clarifying of features. Products of high quality are therefore possible. This process is seamless as there are no discontinuities in which a notation in one phase is replaced by a different notation in another phase. A complete OO methodology includes methods for OO analysis (OOA), e.g. such as proposed by Jacobson et al (1992), OO design (OOD), e.g. such as proposed by Wirfs-Brock et al (1990) as well as OO programming (OOP), e.g. such as proposed by Cox (1986).

## 3.4 The Software Process Model and SDLC

In order to manage a software development project it is necessary to adopt an appropriate **software process model**. The primary functions of a software process model are to determine the order of the phases involved in software development and evolution, and to establish the transition criteria for progressing from one phase to the next. These include completion criteria for the current phase as well as choice criteria and entrance criteria for the next phase. Software process models therefore provide guidance regarding the order in which a project should carry out its major tasks.

Various interpretations of the software process model exist, such as Stagewise Models, the Waterfall Model, Evolutionary Development or Incremental Models, Transform or Prototyping Models and more recently the Spiral Model as well as OO models. Each of these models are briefly reviewed below.

As early as 1956, experience on large software systems had led to the development of a **Stagewise Model** (Benington,1956). This model stipulated that software be developed in successive phases.

The **Waterfall Model** (Royce,1970) was a refinement of the Stagewise Model. It provided two primary enhancements to the Stagewise Model, i.e. the recognition of the feedback loops between phases as well as an initial incorporation of prototyping in the software life cycle.

The phases of the **Evolutionary Development Model** (McCracken & Jackson,1982) consist of expanding increments of an operational software product, with the directions of evolution being determined by operational experience. It gives the user a rapid initial operational capability and provides a realistic operational basis for determining subsequent product improvements.

The **Transform Model** (Balzer et al,1983) provides for the automatic convertion of a formal specification of a software product into a program satisfying the specification and then, improving and exercising the software product in an iterative loop.

The **Spiral Model** (Boehm,1986) of the software process can accommodate most previous models as special cases and further provides guidance as to which combination of previous models best fits a given software situation. The model reflects the underlying concept that each cycle involves a progression that addresses the same sequence of steps, for each portion of the product and for each of its levels of detail, from an overall concept down to the coding of each individual program. The spiral model applies equally well to development or maintenance.

The development spiral is initiated by a hypothesis that a specific operational mission could be improved by a software effort. At specific times during the spiral process, if the hypothesis fails

certain prescribed tests, the spiral is terminated. If not, it terminates with the installation of new or modified software and the hypothesis is tested by observing the effect on the operational mission.

Each cycle of the spiral begins with the identification of

- the objectives of the portion of the product being elaborated (e.g. functionality, performance);
- the alternative means of implementing this portion of the product (e.g. buy, reuse, design), and
- the constraints imposed on the application of the alternatives (e.g. cost, schedule).

The alternatives relative to the objectives and constraints are then evaluated. This process will identify areas of uncertainty that are significant sources of project risk.

If necessary, the next step should involve the formulation of a cost-effective strategy for resolving the sources of risk. This may involve the use of risk-resolution techniques such as prototyping, simulation, benchmarking or combinations of these. Risk considerations can lead to a project implementing only a subset of all the potential steps in the model, e.g. if performance or user-interface risks strongly dominate program development, an evolutionary development approach could be followed: a minimal effort to specify the overall nature of the product, a plan for the next level of prototyping as well as the development of a more detailed prototype to continue to resolve the major risk issues. In this case, specification writing might be addressed but not exercised.

Each cycle is completed by a review involving the role players concerned with the product. The review covers all deliverables developed during the previous cycle, including the plans for the next cycle and the resources required to carry them out. The major objective of the review is to ensure that all concerned parties are mutually committed to the approach for the next cycle.

**OO models** include the fountain model (Henderson-Sellers & Edwards,1990), the recursive/parallel life-cycle (Berard,1993), the revised spiral model (Van der Walt,1993) as well as

the round-trip gestalt design (Booch,1994). The revised spiral model is based upon the spiral model (Boehm,1986) and was revised for OO development.

In this dissertation a **software development life-cycle** (SDLC) model is interpreted as an instance of a software process model. It provides a framework according to which software development, and in this case the integration of legacy systems with the client/server environment, should be performed. A complete software life-cycle spans from initial formulation of the problem, through analysis, design, implementation and testing of the software, followed by an operational phase during which maintenance and enhancement are performed.

## 3.5 The CMM

The CMM was first proposed by Humphrey (1989) of the Software Engineering Institute at the Carnegie-Mellon University. It is a strategy for improving the software process which is independent of the actual SDLC model used. It is based on the belief that problems regarding software development are caused by improper management of the software process and therefore the use of new software techniques cannot be solely responsible for increased productivity and profitability (Paulk,1995).

The CMM assists organisations in the incremental improvement of the management of the software process and it is believed that as a result, improvements in techniques will be a natural consequence. Five different levels of process maturity (level one being the lowest and level five the highest) are defined. An organisation advances in a series of small, evolutionary steps towards the higher levels of maturity. The five levels of process maturity are:-

1. The **Initial Level**, i.e. there are no sound software engineering management practices in place within the organisation. Time and cost overruns are at the order of the day and are caused by a lack of planning in particular. The software process is unpredictable as it depends totally on the current staff, i.e. the process changes as

the staff changes. It is therefore impossible to predict the time and cost associated with a product with accuracy. SASOL is without doubt a level one organisation.

2. The **Repeatable Level** where the basic software project management practices are in place. Management and planning techniques are based on previous experience with similar products. Measurements (e.g. of costs and schedules) are taken. These measurements can be used to draw up realistic duration and cost schedules for future projects. Problems are identified by managers as they arise and immediate corrective action is taken to prevent crises.

3. The **Defined Level** where the process for software development is fully documented, i.e. both the managerial and technical aspects of the process are defined. Wherever possible, efforts (e.g. reviews) are continuously made to improve the process. Schach (1996) is of the opinion that new technology such as a software engineering environment (SEE) (discussed in Section 5.5), can be introduced at this level in order to increase quality and productivity.

4. The **Managed Level** where quality and productivity goals are set for each project. These two quantities are continually measured and corrective action is taken whenever unacceptable deviations from the goal occurs. According to Schach (1996) individual projects have reached this level, but no organisation has yet reached it. It can be regarded as a target for the future.

5. The **Optimising Level** where an organisation is continually improving the processes by making use of statistical quality and process control techniques. The knowledge gained from projects is utilised in future projects. This will result in a consistent improvement in productivity and quality. In order to improve its software process, an organisation should start off by attempting to gain an understanding of its existing process. The intended process should then be formulated after which the actions that will result in achieving the intended process

are determined and prioritised. A plan to achieve the improvement is drawn up and executed. By repeating this series of steps an organisation will be successively improving its software process. No organisation has yet reached level five.

## 3.6 Summary

A methodology for software development is an overall systematic approach to the production of software, which is based upon a paradigm, representation schema, methods, techniques, procedures as well as deliverables. A **paradigm** is an abstract model on which the methodology is based such as the structured and OO paradigms. The **representation schema** is based on the paradigm and provides a notation for the graphical representation of the system to be developed. A **method** provides a manner of carrying out the work steps of a complete phase of software development. **Techniques** are used by methods and are often utilised for a portion of a phase of software development. **Procedures** refers to the set of activities and steps required to implement methods. **Deliverables** are produced as a result of development tasks and activities.

Six classes of methodologies were distinguished according to the various perspectives taken by established methodologies. These classes are process-oriented, data-oriented, behaviour-oriented, cross-references between perspectives, meta as well as OO. The shortcomings of the process- and data-oriented perspectives are that they are either action-oriented or data-oriented, but not both. Process-oriented methodologies concentrate on the actions of the system under development while the data is of secondary importance. With data-oriented methodologies the emphasis is on the data while the actions that operate on the data are of less significance. In contrast, OO methodologies consider both data and actions to be of equal importance as an object is a unified software component that incorporates both the data and the actions that operate on that data.

OO methodologies are based on the OO paradigm which promises reduced complexity of software systems, reusable objects, increased programmer productivity as well as the ability to manage the increasing size of software systems. This results in reduced development and maintenance costs as well as the implementation of reliable software systems. OO methodologies will therefore make a

significant contribution to solving the software crisis, namely, that the quality of software is generally unacceptably low and that deadlines and cost limits are not met.

Legacy systems were developed by making use of either a process-oriented methodology or no methodology at all. The use of no methodology explains the presence of corrupt data and the difficulty of maintenance of legacy systems.

A software process model should be adopted to prescribe the order of the different phases involved in software development as well as to establish the transition criteria for progressing from one phase to the next. In the context of this dissertation, a SDLC model is an instance of a software process model. In Figure 3.2 the conceptual model for integration (Chapter 1) is refined. As illustrated in this figure, the techniques of the integration methodology are based on a paradigm. These techniques are used by methods which are in turn, implemented by procedures. The integration methodology is based on a software process model and the paradigm is visualised by means of the representation schema. Automated tools provide for computerised support for the integration methodology. Deliverables are produced as a result of the integration process.

The CMM is based on the belief that problems regarding software development are caused by improper management of the software process and it was therefore developed to assist organisations in the incremental improvement of software process management. Five different levels of process maturity are defined. They are from the lowest to the highest the initial, repeatable, defined, managed and optimising levels. An organisation advances in a series of small, evolutionary steps towards the higher levels of maturity. SASOL can be considered to be a level one organisation as no sound software engineering management practices are in place within the organisation. This situation partly motivated the context of this investigation as outlined in Section 1.3 of Chapter 1.

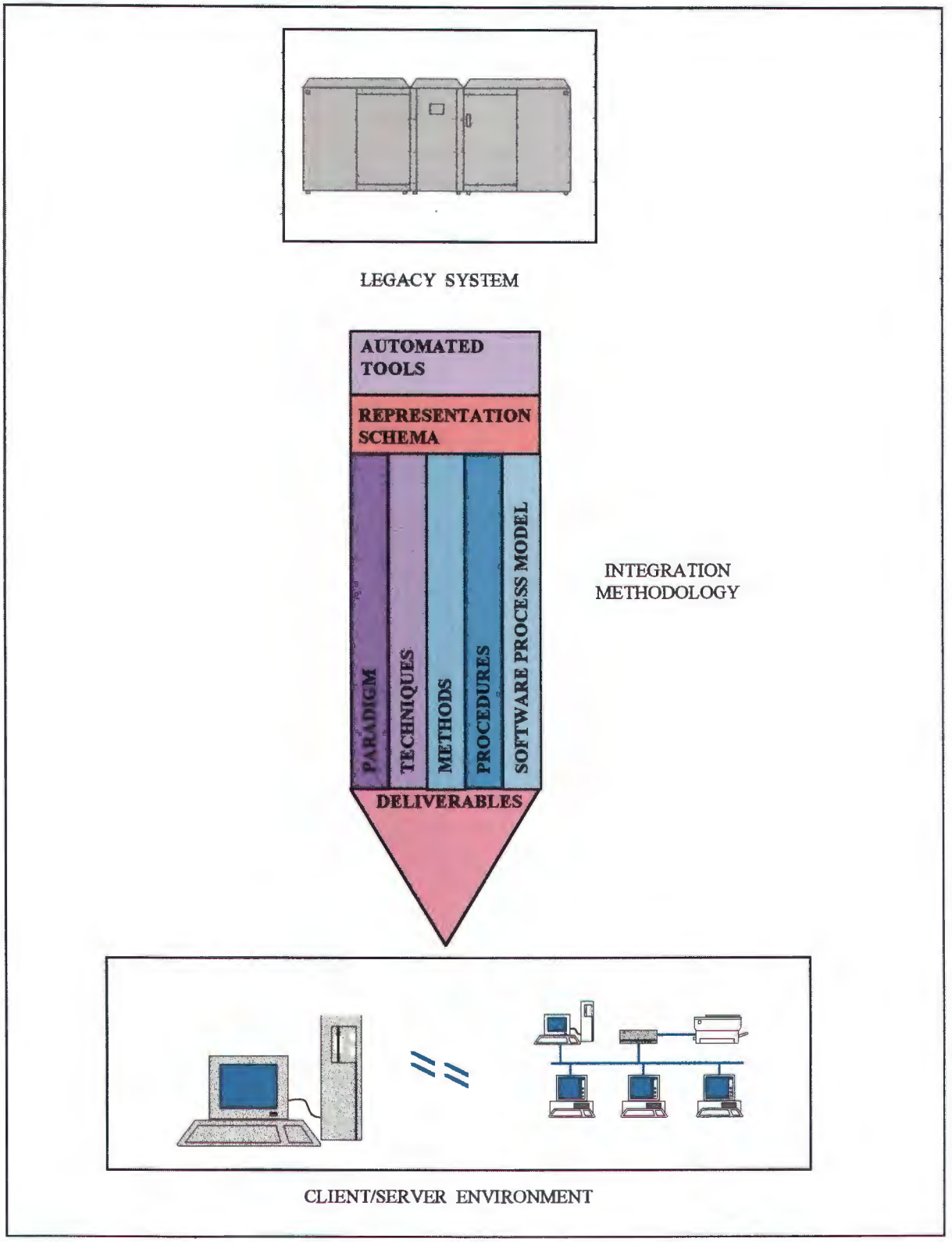


**Figure 3.2** The Refined Conceptual Model

# CHAPTER 4

# THE INTEGRATION METHODOLOGY

## 4.1 Introduction

The existence of legacy systems, which usually represent large capital investments, complicates the integration with client/server environments. A methodology for integrating legacy systems with the client/server environment should include the methods, procedures and techniques used to direct the activities of each phase of integration, the deliverables for each phase as well as a representation schema.

The methodology should ensure that project objectives are well-defined and understood by all project staff, tasks are fully defined and assigned to staff with the appropriate skills and quality reviews are conducted throughout each phase of the project. The methodology should be well-manageable so that the integration process can be controlled and measured but not so rigid that it fails to provide sufficient degrees of freedom to encourage creativity and innovation.

In this chapter, the features and technical requirements of an integration methodology are identified in the context of the most dominant characteristics of the legacy systems investigated in Chapter 2. The generic tasks to be undertaken during the integration life-cycle are presented. The Enhanced Spiral Model for Integration (ESMI) is proposed as a software process model for integrating legacy systems with the client/server environment. It embodies many of the features of other software process models and adds risk-driven and prototyping strategies. The representation schema, methods, techniques, procedures as well as cycle deliverables of the ESMI are summarised. Project management is covered in the final section of the chapter. Project management should ensure that the integration project is completed on time, within budget and that the resultant integrated system satisfies user requirements.

## 4.2 Requirements of the Integration Methodology

Kavanagh (1995) expects more risks with the integration of legacy systems to client/server environments when there are more than a thousand users, more than 100 000 transactions per day and a monolithic database of more than 20 gigabyte (GB). Risks are also higher for centrally

managed services such as security, when there is tight integration with other legacy systems, large batch components, the availability requirement is 24-hours, 7-days, and high investment in equipment. All the legacy systems within SASOL identified in Chapter 2 have at least two of these characteristics.

According to Boehm (1989), important dimensions which influence the risk inherent in a project include:-

1. **Project size.** The larger the project with regard to cost, staffing levels, elapsed time and number of departments affected, the greater the risk.
2. **Experience with the technology.** As the technical knowledge (regarding the project's hardware, operating system, DBMS and application language) of the project team and the IS organisation decreases, project risk increases. By making use of consultants with the necessary skills, risks can, however, be reduced.
3. **Project structure.** Structured projects pose lesser risk than non-structured projects. In the case of structured projects the outputs are defined completely by the very nature of the task, and are fixed without being subject to change during the life of the project. The outputs of non-structured projects are more dependent on the manager's judgement and hence vulnerable to change.

The five legacy systems which were identified within SASOL range from 376 to 707 programs with the number of users ranging from 58 to 1 200. A project to integrate one of these systems with the client/server environment will take a significant amount of time and also cost a significant amount of money. Moreover, limited experience of client/server technology exists within SASOL. According to Boehm's (1989) criteria, a project to integrate one of these legacy systems will have high inherent risk.

McFarlan (1989) identifies the degree of structure and level of technology of a project to influence the project's risk. High structure implies that the outputs of the project are very well-defined by the nature of the task and that the possibility of the users changing the output requirements is

essentially non-existent. Projects which are highly structured and which present technology familiar to the organisation (low level of technology), have the lowest risk. Highly structured projects which present unfamiliar technology (high level of technology) are vastly more complex, e.g. the moving of a system to a different computer platform without significant enhancements. Projects which have low structure and a low level of technology, have a reduced risk if managed effectively. An explicit effort to involve the users is critical to the success of these projects. Projects with low structure and high technology involve the highest risks. These projects are complex and their managers need both technical experience as well as the ability to communicate with users.

Integration projects for the identified legacy systems will involve high technology as little experience in client/server technology is available within SASOL. It is therefore evident that an integration project is one of extremely high risk and the integration methodology should therefore have a risk-driven approach.

One of the most prominent characteristics of the legacy systems of SASOL was their tight integration with both other legacy systems as well as various non-legacy systems. Client/server technology is furthermore relatively new and immature and experienced professionals are difficult to find. The US Air Force handbook on software risk abatement (1988) assesses the probability that a system with strong interdependencies, new technology or a lack of skilled staff will overrun its budget at between 0,7 and 1,0, which indicates that such an outcome occurs frequently. An integration project is therefore likely to overrun its budget. The OO paradigm is considered to solve the problems of the structured paradigm by contributing to a solution of the software crisis, discussed in Chapter 3.

The OO principles, summarised in Appendix C, have the potential for significant support for the integration of legacy systems with the client/server environment. By isolating important aspects and suppressing irrelevant aspects until a later stage, data abstraction can be used to master complexity during the integration process. Data encapsulation allows for the hiding of the detail of complex classes making them easy to use. Information hiding prevents small changes from having

large ripple effects and as a result reduces maintenance time and cost of the integrated system. It ensures data integrity as data structures can only be accessed via its own methods.

Classification also contributes to a reduction in complexity during the integration process. The creation of class libraries containing reusable classes will result in integrated systems which are reliable and easy to maintain.

Inheritance provides conceptual simplification and reduces the amount of redundant code in the integrated system. Maintenance of a system developed by making use of inheritance, will only affect an isolated area instead of being carried through the whole system. This results in reduced maintenance costs.

Polymorphism provides a type of operation reuse which increases the reliability of the integrated system because software which makes use of well-proven and stable operations is likely to have less errors. Modularity improves ease of maintenance on the integrated system.

The proposed integration methodology should therefore be based on the OO paradigm. OO software development requires iteration among life-cycle phases, a prototyping strategy as well as the incremental building of a product (Du Plessis & Van der Walt,1992). As a result of the inherent risk involved in an integration project, it is, however, not recommended to start with such a project as a pilot OO project. An integration project's team should be well-trained in OO and client/server technology and, if possible, be given support by appropriate outside help from experienced consultants.

### 4.3 The Integration Methodology

The reasons for having originally adopted the Spiral Model (Boehm,1986) for the OOISEE project at UNISA are the OO requirements of iteration among life-cycle phases, a prototyping strategy as well as the incremental building of a software product. The revised spiral model for OO development (Du Plessis & Van der Walt,1992; Van der Walt,1993) was customised during this

investigation to provide a framework for integrating legacy systems with the client/server environment. This software process model was chosen for its OO and risk-driven strategies. It consists of five cycles: the Feasibility, Architecture, Analysis, Design and Implementation Cycles. The integration project starts with the first cycle from the centre of the spiral. As integration progresses, consecutive cycles are completed until the final acceptance and completion of the project. Generic tasks to be undertaken during each of the quadrants have been identified following Sage and Palmer's system approach (Sage & Palmer,1990), i.e. tasks concerned with Issue Formulation and Assessment, Risk Analysis and Evaluation of Alternatives, Development as well as Review / Planning.

In order to reduce the chance of project failure, strong emphasis is put on risk analysis before the actual beginning of the integration process during the Implementation Cycle. Various reviews are held throughout the integration life-cycle at checkpoints to ensure control of the integration process. The integration process may move to a previous cycle when needed.

The generic tasks relevant for the integration of a legacy system with the client/server environment are now explained in the context of the five cycles of the Enhanced Spiral Model for Integration (ESMI).

### 4.3.1 Generic Tasks of the Integration Life-cycle

The identified generic tasks to be undertaken during each of the quadrants of a systems integration life-cycle are illustrated in Figure 4.1. These tasks are summarised here.

**Quadrant 1 - Issue Formulation and Assessment** during which the following tasks are performed:-

1. Identify the objectives of the cycle.
2. Formulate a strategy to reach the identified objectives according to plan.
3. Assess existing assets.

4. Identify alternative strategies to reach the identified objectives.
5. Identify the applicable constraints.
6. Establish quality metrics.

**Quadrant 2 - Analysis and Evaluation of Alternatives** imply the following:-

1. Analyse results of assessment.
2. Risk analysis, i.e. evaluate the identified alternative strategies according to the risks involved in terms of technical aspects, costs, schedule and support. The strategy with the lowest risk should be chosen. The lowest risk is defined according to priorities set on the evaluated risk areas.
3. Risk avoidance planning, i.e. planning should be done to lower the identified risks.
4. Prototyping, i.e. test areas of risk by making use of modelling techniques such as prototyping, simulation and analytic modelling. In this way areas of risk are reduced.
5. Commit to a specific strategy.

**Quadrant 3 - Development** comprises:-

1. Development activities and tasks. Provision is made for evolutionary development and this cycle can be entered for either a module of code or a subsystem under development.

**Quadrant 4 - Review and Planning** imply the following:-

1. Evaluation, i.e. the deliverables produced during the third quadrant are verified against the specifications and quality metrics. A decision whether to update the project baseline is made. Quality related activities (e.g. code walkthroughs and reviews) are performed. The evaluation ends with the

product development review. The result of this evaluation is the acceptance of the developed product.

2. Planning for the next cycle of development is done. A review of the work breakdown structure (WBS) and cost breakdown structure (CBS) is done. The schedule for the next cycle is drawn up. This quadrant ends when the user is presented with a software development plan for the next cycle and the commitment to continue with the next cycle is obtained from the user.

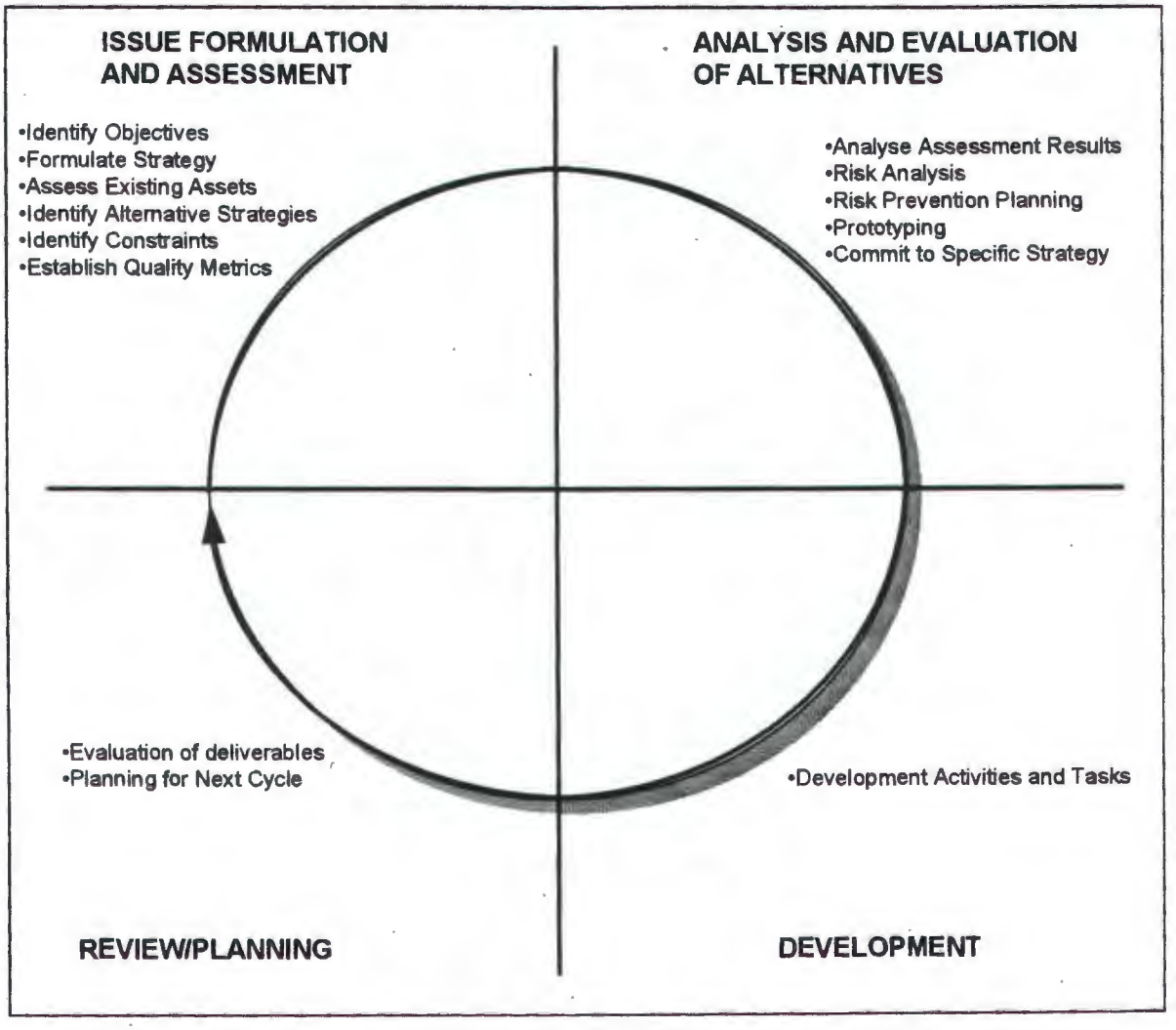


**Figure 4.1** The Generic Tasks of the Integration Life-cycle

### 4.3.2 The Enhanced Spiral Model for Integration (ESMI)

As illustrated in Figure 4.2, the ESMI consists of five cycles, namely the Feasibility, Architecture, Analysis, Design and Implementation Cycles. Work in each quadrant of the cycles follows the generic format outlined in Section 4.3.1 and is now presented.

#### (i) Feasibility Cycle

The Feasibility Cycle commences with the need to integrate a legacy system with a client/server environment. The Feasibility Cycle is traversed only once. During issue formulation, a business case for integration is determined. Strategic, business and technical objectives of the project are determined and quantified as accurately as possible. This enables the scope to be defined. A problem statement is formulated which includes the constraints relevant to the project. A strategy is formulated for reaching the objectives according to plan.

The legacy system is assessed in terms of the application, infrastructure, skills of maintenance staff, maintenance history as well as costs and problems. The application is assessed in terms of size, performance, complexity, condition, subsystem integration, the functions it performs as well as the data it maintains. The maintenance history is assessed in order to calculate the staff complement required to do basic maintenance as well as to identify potential components of the application for termination or redesign. System problems which require maintenance are assessed from both IS/T as well as user's perspectives. The results of the assessment are documented in the Legacy System Description Document.

Alternative strategies for integration are identified. Quality metrics, i.e. of technical performance, cost/benefit performance as well as operating performance, for the integrated system are determined.

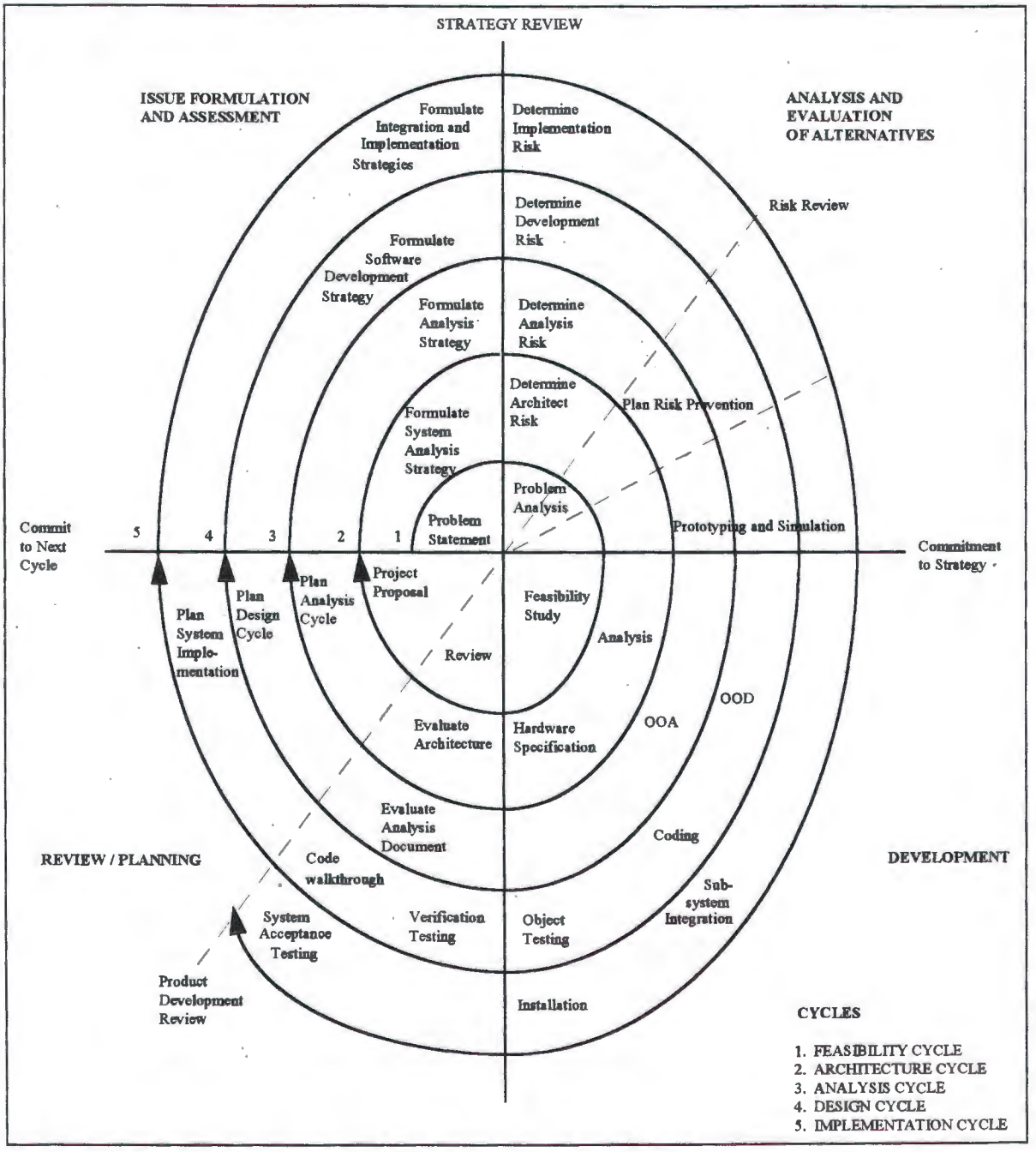


**Figure 4.2** The Enhanced Spiral Model for Integration

During analysis and evaluation, the alternative strategies for integration are evaluated. The legacy system assessment results are analysed. A feasibility study, involving the first cost-benefit analysis

(discussed in Section 5.3.2) of a chosen alternative strategy is conducted in the development quadrant. Feasibility, in respect of integration comprises (Kavanagh,1995):

- technical feasibility, i.e. is it possible to do;
- economic feasibility, i.e. is it worth doing;
- operational feasibility, i.e. will it work in the organisation concerned;
- schedule feasibility, i.e. is it possible to do it in the required time;
- political feasibility, i.e. will it be acceptable in practice in the organisation concerned.

At the end of the Feasibility Cycle the chosen strategy is reviewed. A commitment to a specific strategy is made. The feasibility study culminates in a Preliminary Project Proposal.

**(ii) Architecture Cycle**

Once the Preliminary Project Proposal is accepted and authorised by management, the Architecture Cycle is entered. The Software Quality Assurance Plan (SQAP), the Project Risk Management Plan (PRMP) as well as the Software Project Management Plan (SPMP) are deliverables of this cycle.

During the Architecture Cycle the top level hardware and system software architectures are defined. This will allow for a more accurate cost estimation. Hardware architecture includes network hardware, server hardware and workstations. System software architecture includes communications protocols, network operating systems, server operating systems, DBMS's, client/server tools, development tools and personal workstation applications.

Server hardware includes the file servers, resource servers, database servers and application servers. File servers make files available to client systems, so that users can share their data and software as well as rely on scheduled maintenance. A resource server manages specific resources which are too expensive to dedicate to a single client, e.g. a print server. A server is needed to run a database management system (DBMS). The DBMS should be manageable, i.e. have high availability, data

integrity and security. An application server runs application software and may be the same server as the database server. For advanced uses, e.g. graphically intensive work, a dedicated application server may be necessary. The types of possible workstations include Microsoft Windows, Macintosh, OS/2, Unix, Windows NT as well as X-Windows or dumb terminals.

Development tools can be classified into query and report tools (e.g. Crystal Reports), EIS tools (e.g. Lightship), client/server development tools with flexible GUI's for general application development (e.g. Visual Basic) as well as PC business tools (e.g. Excel).

In order to select the most suitable client/server tools, the application systems for which the tools will be employed, the environment where the tools will be used as well as the technological direction of the organisation need to be considered. When defining the top level hardware and system software architectures, it is important to conform to standards where possible. Standards were discussed in Section 2.7.5 of Chapter 2.

The existing infrastructure is assessed in terms of all processor and network hardware as well as system software (operating system, database, security and management packages). Alternative architecture strategies are identified which include issues such as the type of network and hardware platform. The applicable constraints as well as quality metrics are identified. Metrics include those of technical performance, cost/benefit performance and operating performance. Results of the assessment are documented in the Legacy Infrastructure Description Document.

In the second quadrant, the results of the assessment of existing hardware and system software architectures, are analysed. Risk analysis is performed on all the identified strategies and results are documented in the Architecture Strategy Evaluation Report. During the risk review, the strategy with the smallest risk is selected for further evaluation. Risk avoidance planning is done and risk areas identified in the risk analysis may be simulated or prototyped to reduce risk and to enhance the understanding of the proposed system. Risk analysis and risk avoidance planning are discussed as steps of risk management in Section 5.3.2 of the next chapter. Prototyping is discussed in

Section 5.3.3. At the end of the second quadrant a commitment to a specific hardware and system software strategy is made.

In the third quadrant top level analysis is done. The Architecture Document, which forms the basis of the contract between the user and the software developer and which includes the user requirements and acceptance criteria, is compiled. Top level class diagrams, data flow diagrams (DFD's) and state diagrams are produced. Subsystems are identified which models the top level object, functional and dynamic perspectives respectively. Subsystems are seen as very high level objects with a specific function and with low coupling to other high level objects. The hardware configuration, which involves the definition of the required hardware, is defined.

In the fourth quadrant, the chosen strategy is reviewed. The hardware architecture is evaluated in order to make sure that all issues are covered and that the hardware architecture ties up with the software architecture. The Architecture Cycle ends after the planning of both the Analysis Cycle as well as the rest of the integration project. The WBS, CBS and schedules are updated and the Project Proposal for the rest of the system is compiled. The Software Project Management Plan (SPMP) is defined for the rest of the project. The Architecture Cycle ends when the Project Proposal is accepted by the user. This cycle may be traversed more than once, if it is realised during the Analysis or Design Cycles that the hardware specification will not meet system demands.

**(iii) Analysis Cycle**

The analysis strategy is defined in detail during the Architecture Cycle. During the first quadrant of the cycle, objectives, strategies and constraints for the analysis of the subsystems are considered. If a subsystem strategy includes object wrappering or the improvement of the user-interface without changing the original application code, existing subsystems must be assessed and documented in a Legacy Subsystem Description Document. Object wrappering is the encapsulation of an existing system and the restriction of its interface to a set of inbound and outbound messages. Wrappering of non-OO systems was discussed in Section 2.7.3. The first quadrant yields a strategy definition review.

During the second quadrant risk analysis is performed for the alternative subsystem strategies. The results are documented in a Subsystem Strategy Evaluation Report. The subsystem strategies with the lowest risk are identified and risk avoidance planning is done for the identified high risk areas of these strategies. Risk analysis and risk avoidance planning are steps of risk management discussed in Section 5.3.2 of Chapter 5. Prototyping (discussed in Section 5.3.3) may be done to resolve uncertainties of risk areas. This quadrant ends with a commitment to a strategy for every subsystem.

The third quadrant involves detailed OOA. OOA is discussed in Section 5.2.1 of the next chapter. The top level class diagrams, DFD's and state diagrams of the Architecture Document are used as the starting point in the definition of the Object, Dynamic and Functional Models. This cycle results in the Object, Dynamic and Functional Models for the system which are documented in the Analysis Document. The procedures for constructing these models are reviewed in Section 5.4.1.

During the fourth quadrant the Object, Dynamic and Functional Models are evaluated to ensure that they are complete and consistent. The SPMP is updated. OOA is completed after the approval of the Analysis Document. The Design Cycle is planned, the WBS, CBS as well as the SPMP are updated. The Analysis Cycle ends when the user agrees to the continuation of the Design Cycle.

**(iv) Design Cycle**

During the Design Cycle the system is designed and organised into subsystems which are to be developed. In the first quadrant the objectives for the Design Cycle are defined. The design strategy is formulated. If a subsystem strategy involves object wrappering or the improvement of the user-interface without changing the original application code, the existing application code must be assessed. The physical views of the system are modelled and the design alternatives as well as constraints are identified. Design quality metrics are established. The order in which the subsystems and parts within subsystems are designed, will be influenced by the integration and test plan. The classes which form the kernel of the system must be designed and developed first as they

will be used in subsequent development. Testing should be done throughout the Design and Implementation Cycles and is regarded the primary tool of Software Quality Assurance (SQA). SQA is discussed in Section 5.3.4. The first quadrant ends with a software development strategy review.

During the second quadrant the design alternatives are analysed in order to determine the risks of each alternative. Risk analysis (discussed in Section 5.3.2) is performed on the alternative design strategies. The solution strategy with the smallest risk and which satisfies the user constraints are chosen. Prototyping of high risk areas reduces uncertainty and enhances the understanding of the system. Incremental prototyping may be used as part of the iterative and evolutionary strategy of OOD, referred to by Booch (1994) as "round-trip gestalt design". Prototyping is covered in Section 5.3.3 of the next chapter. The quadrant ends with the acceptance of a design strategy which will be documented in the System Design Document.

OO design (OOD) is performed during the third quadrant and is discussed in Section 5.2.2. Rumbaugh (1991) identifies the following steps for OOD:-

1. Combine the three models obtained in the Analysis Cycle to obtain operations on classes.
2. Design algorithms to implement operations.
3. Optimise access paths to data.
4. Implement control for external interactions.
5. Adjust class structure to increase inheritance.
6. Design associations.
7. Determine object representation.
8. Package classes and associations into modules.

OO programming (OOP) is an incremental process and the steps of this quadrant may therefore be re-iterated before a subsystem is completed. OOP is covered in Section 5.2.3 of Chapter 5. The detailed physical design is documented in the Detailed Design Document. Incremental prototyping may be used as part of the iterative and evolutionary strategy of OOD. Throughout the Design

Cycle the completed prototypes are evaluated in order to determine whether user requirements are met and whether they may be integrated into the evolving product.

Regular design reviews are held to measure progress towards the final product. Once a subsystem is completed, verification testing (discussed as part of SQA in Section 5.3.4) is done on the subsystem as a whole and a test report is drawn up. The code is accepted after the product development review. In case of a need to re-iterate the Analysis Cycle, the development strategy is revised.

As subsystems are completed, planning for subsystem integration and system implementation is done. The SPMP is updated. WBS, CBS and cost estimations are reviewed. The Design Cycle ends when the user agrees to the continuation of the Implementation Cycle.

**(v) Implementation Cycle**

The Implementation Cycle is usually traversed only once. During the first quadrant, various strategies for subsystem integration, integration testing and conversion are identified and documented in the Integration Strategy Evaluation Report. Conversion may imply the transferring of data from the legacy system to the client/server environment. Different hardware installation plans are identified. The training schedule, training programme and user manual for new system users are drawn up.

During analysis and evaluation of alternatives, the risks involved in the different subsystem integration strategies are identified. Risk avoidance planning is done to lower the identified risks. Risk avoidance planning is considered a step of risk management and is covered in Section 5.3.2. The quadrant ends with the review of the hardware installation layout, software packaging as well as contents of the training programme.

During the development quadrant, subsystem integration is performed and tested. Incompatibilities between subsystem interfaces may require the Design Cycle to be repeated. Volume testing is done

for those facets of the system which are peculiar to large volumes, including the storage allocations for the database and the service constraints. Volume testing of the storage allocations ensures that the system will handle its expected activity. Volume testing to verify that the system meets the service constraints ensures that the system does not exceed the allowable response times for updating the database, responding to enquiries or producing reports.

A successful parallel test is often a prerequisite for system acceptance. System acceptance testing is the verification that the system meets the user requirements and results are documented in the Acceptance Test Reports. Parallel testing compares the processing of the same data through both the new system and the legacy system. Testing is the primary tool of SQA discussed in Section 5.3.4. Once the user is satisfied that the integrated system can take over the functions of the legacy system and that the system runs according to specifications, the system is considered completed.

### 4.3.3 The Representation Schema

The Object Modeling Notation (Rumbaugh et al,1991) is used to represent the system to be integrated from three related, but different viewpoints, each capturing important aspects of the system, but all required for a complete description.

The Object Model which represents the static, structural, "data" aspects of the system is represented graphically with object diagrams containing object classes. The classes are arranged into hierarchies sharing common structure and behaviour and are associated with other classes.
The Dynamic Model describes the behavioural control aspects of the system to be integrated. These aspects are concerned with time and the sequencing of operations. The Dynamic Model is represented graphically with state diagrams which shows the state and event sequences permitted for classes of objects. Actions in the state diagram correspond to functions in the Functional Model while events in the state diagram become operations on objects in the Object Model.

The Functional Model describes the functional aspects of a system which are concerned with the transformations of values. It is represented with DFD's. DFD's show the dependencies between

values and the computational structure of a system without regard for when or if the functions are executed.

### 4.3.4 The Methods

The methods which form part of the ESMI are:

- OOA which is a method for analysis used during the Analysis Cycle;
- OOD which is a method for design used during the Design Cycle, as well as
- OOP which is a method for programming used during the Design Cycle.

These methods are discussed in the next Chapter in Sections 5.2.1, 5.2.2 and 5.2.3 respectively.

### 4.3.5 The Techniques

The following techniques for the ESMI were identified:

- Software cost estimation techniques are used when performing a cost-benefit analysis to project future costs;
- risk management techniques to identify, address and eliminate risk items;
- Rapid prototyping is used for the gathering of requirements as well as for reducing the risk areas during risk analysis;
- Software Quality Assurance (SQA) techniques are essential to ensure an integrated system of high quality;
- Software reuse allows for the creation of libraries with reusable components, ensures software of high quality as well as shortens the integration process.

These techniques are discussed in the next Chapter in Sections 5.3.1, 5.3.2, 5.3.3, 5.3.4 and 5.3.5 respectively.

### 4.3.6 The Procedures

The ESMI starts with a Feasibility Cycle during which the feasibility of integration is addressed, followed by an Architecture Cycle during which the top level hardware and system software architectures are defined. The Architecture Cycle may be entered more than once, if during the next two cycles it is realised that the hardware specification will not meet system demands.

The Analysis Cycle deals with the analysis of the problem domain and can also be re-iterated. The Design Cycle which follows the Analysis Cycle is re-iterated numerous times to complete a subsystem and deals with constructing both the logical and physical designs.

The last cycle to be completed is the Implementation Cycle which is usually traversed only once and which deals with the integration of subsystems and the implementation of the integrated system. Work in each quadrant of all cycles follows the generic format outlined in Section 4.3.1.

Apart from the overall procedure, procedures for constructing the Object, Dynamic and Functional Models as well as for cost estimation were also distinguished. These procedures are discussed in the next chapter in Sections 5.4.1 and 5.4.2 respectively.

### 4.3.7 The Deliverables

The deliverables for the five cycles of the ESMI are now summarised.

#### (i) Feasibility Cycle

During the first quadrant a **Problem Statement** containing the constraints relevant to the integration project, is formulated. The results of the legacy system assessment should be documented by means of schematic diagrams, document description worksheets, DFD's and notes to form the **Legacy System Description Document**. Schematic diagrams depicts the information flow, timing and volumes in chart form. Document description worksheets describe the documents

of the legacy system. DFD's depicts the functions and the files used by the legacy system. Notes should be organised by information element, function and document.

During Analysis and Evaluation, a **Feasibility Report** is compiled. A typical Feasibility Report will include:

- a summary of the scope of the integration and the recommendations;
- a problem statement containing the objectives and requirements of the integration project;
- a cost-benefit analysis;
- a risk analysis;
- an analysis on the probable effects on the organisation, internally as well as externally;
- specification of departmental and individual responsibilities for the implementation of the integrated system;
- the criteria by which successful implementation will be measured, e.g. cost/saving, as well as
- a detailed action plan showing how, when and by whom each measurable unit will be completed.

The feasibility study culminates in the **Preliminary Project Proposal** which contains:

- a detailed problem definition within the business context;
- a high level strategy which describes the planned approach to be followed in order to address the problem definition;
- an overall estimate of software and hardware resources needed;
- an analysis of the cost-effectiveness of the proposed system, as well as
- an indication whether the existing time and budget constraints will be acceptable.

**(ii) Architecture Cycle**

The results of the assessment of existing infrastructure are documented in the **Legacy Infrastructure Description Document**. During the second quadrant the results of analysis are documented in the **Architecture Strategy Evaluation Report**.

The **Architecture Document** forms the basis of the contract between the user and software developer and includes:

- the user requirements;
- acceptance criteria;
- class diagrams;
- DFD's;
- state diagrams;
- subsystem specification, and
- hardware definition specification.

The **SPMP** which describes the integration process in fullest detail and should conform to IEEE standards, is compiled. It is successively refined as the project progresses. The SPMP is updated at each cycle review. In addition to defining the work to be done, the SPMP provides management with the basis against which progress can be periodically reviewed and tracked. It includes aspects such as the life-cycle model to be used, project responsibilities, managerial objectives and priorities, the techniques and tools to be used as well as resource allocations. The SPMP reflects the separate phases of the integration project, the people involved in each task as well as the deadlines for completing that task.

It contains the technical and managerial work breakdown structure (WBS) to be performed, estimations of resource and budget requirements, the cost breakdown structure (CBS) as well as the development schedule based on these estimates. Major components of the SPMP are therefore the deliverables (what the user is going to get), the milestones (when the user gets them) and the

budget (how much it is going to cost). In addition it should contain the test plan and system maintenance plan. The test plan describes the procedure for testing the deliverables of the integration project. The system maintenance plan includes the backup, corrective maintenance and disaster recovery procedures. According to Humphrey (1989) the elements of a SPMP are the goals and objectives, a sound conceptual design, the WBS, product size estimates, resource estimates as well as the project schedule.

The IEEE Standard 1058.1 (IEEE 1058.1,1987) prescribes an outline of a SPMP (Figure 4.3). There are distinct advantages to following this standard. These advantages include the fact that the standard incorporates the experience of representatives of major organisations involved in software as well as input from both industry and universities. Another advantage is that the IEEE SPMP is designed for use with all types of software products, irrespective of size and functionality. The plan framework is, however, not described in detail here.

A **PRMP** consists of each of the individual risk management plans for each risk item as well as an overview of how the individual plans fit together with each other and with the overall integration project plan. It ensures that each integration project makes an early identification of its top risk items, develops a strategy for resolving the risk items, identifies and sets down an agenda to resolve new risk items as they surface as well as highlights progress versus plans in periodical reviews. It establishes the necessary budgets and schedules for risk reduction activities and ensures that they are compatible with those in the integration project's SPMP.

| | |
|---|---|
| 1. | Introduction |
| 1.1 | Project Overview |
| 1.2 | Project Deliverables |
| 1.3 | Evolution of the SPMP |
| 1.4 | Reference Materials |
| 1.5 | Definitions and Acronyms |
| 2. | Project Organisation |
| 2.1 | Process Model |
| 2.2 | Organisational Structure |
| 2.3 | Organisational Boundaries and Interfaces |
| 2.4 | Project Responsibilities |
| 3. | Managerial Process |
| 3.1 | Managerial Objectives and Priorities |
| 3.2 | Assumptions, Dependencies and Constraints |
| 3.3 | Risk Management |
| 3.4 | Monitoring and Controlling Mechanisms |
| 3.5 | Staffing Plan |
| 4. | Technical Process |
| 4.1 | Methods, Tools and Techniques |
| 4.2 | Software Documentation |
| 4.3 | Project Support Functions |
| 5. | Work Packages, Schedule and Budget |
| 5.1 | Work Packages |
| 5.2 | Dependencies |
| 5.3 | Resource Requirements |
| 5.4 | Budget and Resource Allocation |
| 5.5 | Schedule |
| Additional Components | |

**Figure 4.3** Outline of a SPMP (IEEE 1058.1,1987)

The **SQAP** comprises SQA monitoring procedures, practices and policies for assuring compliance with the various SQA standards. Sage and Palmer (1990) identify the essential components of the SQAP:

- identification and implementation of the scope and purpose of the plan;
- identification and implementation of the organisational structure for implementing the plan, including specific tasks to be performed by members of the SQA group;
- identification and implementation of documents that need to be prepared as well as methods to determine quality and adequacy of documentation;
- identification and implementation of metrics, standards, procedures and practices that will be used in implementing the plan, and
- identification and implementation of methods to be used in collecting, maintaining and recording SQA information.

At the end of the fourth quadrant the **Project Proposal**, which must be accepted by the user in order to proceed with the integration project, is compiled for the rest of the system.

**(iii) Analysis Cycle**

The **Legacy Subsystem Description Document** contains a description of existing legacy subsystems. The **Subsystem Strategy Evaluation Report** is compiled after risk analysis is performed on the alternative subsystem strategies. The **Analysis Document** containing the logical design of the subsystems, is compiled in the third quadrant and consists of the Object, Dynamic and Functional models for the system. The **SPMP** is reviewed and updated with more accurate information.

### (iv) Design Cycle

The **System Design Document** should contain a description of the design strategy which has the smallest risk and which satisfies the constraints of the user. The **Detailed Design Document** contains the detailed physical design. OOP results in both **Source and Object Code Listings**. The **completed Subsystems** are verified to satisfy requirements and **Test Reports** containing the results of the verification testing, are compiled. The **SPMP** is updated with the latest available information.

### (v) Implementation Cycle

During the first quadrant the alternative strategies for subsystem integration, integration testing and conversion are documented in the **Integration Strategy Evaluation Report**. The **Training Schedule**, **Training Programme** and **User Manual** for the new system users are compiled. The Training Schedule contains the the names of the users to be trained, the locations where training will be given as well as the dates and time when training will be given. The Training Programme contains the contents of the training. The User Manual contains a description of the procedures necessary to operate the system from a user's perspective. The **System Maintenance Plan** includes the backup, corrective maintenance and disaster recovery procedures. The results of acceptance testing are documented in the **Acceptance Test Reports**. The final deliverable of the integration project is an **Integrated System**.

## 4.4 Project Management

Managers need different styles and approaches to manage different types of projects effectively. The right approach flows from the specific project rather than the other way around. The corporate culture in which the project is performed should influence the management approach followed, e.g. the use of formal project planning and control tools is more likely to produce successful results in a highly formal environment than in an environment where the prevailing culture is more personal and informal. Skills needed with high risk projects, such as an integration

project, include technical and administrative experience and knowledge, the ability to establish and maintain teamwork as well as the ability to communicate with clients.

The major categories of project management tasks are project planning, project organisation, project directing, project control and report writing (Sommerville,1989; Van der Walt,1993; Shtub et al,1994). The ESMI facilitates the management of integration projects by dividing the integration process into cycles, each with a prescribed WBS, deliverables, review points and management procedures for planning, monitoring and controlling a project. All project management tasks are completely dependent on the availability of proper information regarding the integration process. Some of the management tasks are unique to a cycle whereas others are generic, i.e. they should be done during every cycle of the ESMI. The various management activities within each of the above-mentioned task categories will now be reviewed within the context of integration.

Thorough **planning** of an integration project ensures that the project is completed on time and within budget. Planning detail is documented in the SPMP and is revised as the project provides better information. Planning activities for an integration project includes:

- establishing the boundaries and scope of the project during the Feasibility Cycle;
- identifying the objectives and applicable constraints for each cycle of the integration project;
- analysing the Problem Statement within the context of the business and conducting an overall estimation of whether the identified requirements can be satisfied using the existing software and hardware resources during the Feasibility Cycle;
- identifying the top-level hardware and system software architectures (e.g. the hardware and software platforms, implementation language and compilers) during the Architecture Cycle;
- defining milestones;
- identifying the standards, methods, techniques and automated tools to be used during the integration life-cycle;

- compiling the Work Breakdown Structure (WBS) of the project during the Architecture Cycle. A WBS allows for the hierarchical breakdown of the work to be accomplished to complete a project into smaller manageable tasks, thereby providing a greater probability that every major and minor activity will be accounted for. It is a planning tool which links objectives with resources and activities in a logical framework and acts as a starting point for other management activities such as size and cost estimation, scheduling, staffing as well as project control. A Cost Breakdown Structure (CBS) are obtained by associating a cost with each of the WBS elements and can be used as a controlling tool;
- determining the skills required to perform the identified tasks of the WBS;
- estimating and allocating the resources required to meet the objectives of each cycle of the integration project. Resources include time, money, personnel and equipment;
- scheduling the tasks identified in the WBS during the Architecture Cycle;
- reviewing and updating the WBS, CBS and schedules after each cycle of integration;
- updating the SPMP as well as scheduling and allocation for the next cycle of integration.

Project **organisation** comprises project structuring, the enforcement of standard procedures and logistical support. Project structuring involves the organisation of human resources in order to indicate functional relationships, delegate responsibility and authority as well as establish communication channels (Roman,1986). This includes decisions regarding organisation structure, control structure, interfaces with sections within the organisation and external contractors, as well as the structuring of both development and SQA teams. Project structuring is usually performed during the Architecture Cycle. The following three levels of management are distinguished for large projects such as an integration project:

- top level management, e.g. the project manager;
- middle management is typically responsible for a functional group within an integration project, e.g. the person reponsible for SQA;

- junior management is responsible for one of the tasks within a functional group, e.g. the person responsible for the development of data communications as part of the total integration project.

Procedures necessary for sound software engineering practice as specified by standards for software development (SABS ISO 9000,1987; DOD-STD-2176A,1988), must be enforced by project management throughout the integration life-cycle. Logistical support aims at providing all resources (e.g. personnel, hardware and automated tools) needed to ensure a cost-effective integrated system. Activities of logistical support are performed throughout the cycles of the ESMI and may include:

- hardware benchmarking in order to identify the hardware that will deliver the best performance within the allocated budget during the Architecture Cycle;
- the acquisition, installation and support of hardware and software necessary for the development of a software system;
- the compiling of a System Maintenance Plan which includes the backup, corrective maintenance and disaster recovery procedures during the Implementation Cycle.

Project **directing** entails the management of people involved in a project and giving direction to the software development process in order to deliver a high quality integrated system. Directing is performed throughout the integration project and includes:

- leading the project team in achieving goals;
- giving direction in fluid situations;
- motivating individuals and groups;
- team building by making use of techniques such as team building meetings;
- managing conflict among team members;
- user liaison.

Charette (1986) is of the opinion that it is management's responsibility to motivate and encourage technical members of the integration team to apply reuse whenever possible. Software reuse is discussed in Section 5.3.5.

Project **control** involves monitoring the progress of an integration project to date and taking corrective action when integration does not proceed according to the SPMP, configuration management, quality management as well as risk management. Monitoring the progress focuses on meeting the objectives of each cycle (established during Issue Formulation and Assessment and documented in the SPMP). Configuration management in terms of an integration project involves the identification, organisation and control of software modifications done by an integration team. Quality management ensures that quality issues are specifically addressed during all the cycles of the ESMI. It involves the measuring of the quality of the deliverables to date, the quality of the integration process as well as the adherence to standards. Risk management aims at identifying, addressing and eliminating risk items throughout the cycles of the ESMI. Risk management is discussed in Section 5.3.2.

**Report writing** includes:

- formulating a Problem Statement as well as compiling both the Feasibility Report and the Preliminary Project Proposal during the Feasibility Cycle;
- compiling the Architecture Document, SPMP, PRMP, SQAP as well as the Project Proposal during the Architecture Cycle;
- updating the SPMP during the Analysis and Design Cycles.

Humphrey (1989) proposes three levels of software process modelling:-

- The **Universal level** provides a global view of a project to be taken by top level management.
- The **Worldly level** guides middle management through the sequence of working tasks for each development phase. It also defines prerequisites and results of tasks.

- The **Atomic level** provides a detailed description of how the individual tasks of the Worldly level must be performed. The format of deliverables is specified. Junior management uses this level of information to guide a group of project participants to complete a specific task.

These three levels of process modelling provide the information needed by the different levels of management. The Universal Level provides an overall view of an integration project in the form of all the cycles of the ESMI. Top level management, (usually the project manager) is involved in the planning and scheduling of tasks, the allocation of manpower and the budgeting of the project throughout its life-cycle. This level of management is also responsible for monitoring the progress of the tasks of an integration project to assure that the project stays on schedule and within budget. In large projects such as an integration project, the project manager is assisted by middle management to coordinate the different lower level tasks such as system integration, testing, customer training and system implementation.

Each cycle of the ESMI comprises a number of management and technical tasks to be performed and which represents the Worldly Level. Middle management (e.g. the software manager) is concerned with the Worldly Level issues of integration. At this level the software solution is modelled on an abstract level, providing the logical structure of the software problem. Task prerequisites and task deliverables for each cycle of the ESMI are identified, i.e. for each task to be performed the inputs, the resources required or participants involved and the deliverables produced should be modelled. The deliverables of the Worldly Level contributes to the baseline of deliverables on the Universal Level. Middle management is concerned with the scheduling of resources to complete the set milestones and the monitoring of the integration process to assure that the integration stays within the schedule and budget. Middle management requires input from both top and junior management in order to perform these tasks.

A phase is further decomposed into Atomic Level activities. At the Atomic Level data structures and logical design of objects are of importance. Junior management (e.g. a group leader of a software development team) reports on the progress of integration to middle management and is

involved in the planning and execution of the techniques and procedures of the methods chosen to perform Atomic Level tasks such as the COCOMO estimation technique (Du Plessis & Van der Walt,1992). The COCOMO technique is discussed in Section 5.3.1.

## 4.5 Summary

The integration methodology should have a risk-driven approach and should be based on the OO paradigm. OO software development requires iteration among life-cycle phases, a prototyping strategy and incremental building of the product. The ESMI conforms to these requirements. In order to reduce the chance of project failure, strong emphasis is laid on risk analysis throughout the integration life-cycle. Various reviews held throughout the life-cycle, ensure deliverables of high quality as well as control of the integration process. The ESMI makes provision for evolutionary development and consists of five cycles: the Feasibility, Architecture, Analysis, Design and Implementation Cycles.

Work in each quadrant of the cycles follows a generic format. During Issue Formulation and Assessment (Quadrant 1) the objectives of the cycle are identified and a strategy formulated to reach the objectives according to plan. Existing assets are assessed and alternative strategies identified to reach the objectives. Constraints are identified and quality metrics established.

During Analysis and Evaluation of Alternatives (Quadrant 2) the results of the assessment in Quadrant 1 are analysed. Risk analysis is performed on the identified alternative strategies. Risk avoidance planning is done and areas of risk are reduced by making use of modelling techniques such as prototyping and simulation. At the end of this quadrant a commitment is made to the strategy with the lowest risk.

Development activities and tasks are performed during the Development Quadrant. During Review and Planning (Quadrant 4) the deliverables of the third quadrant are verified against specifications and quality metrics. Planning for the next cycle is done.

The ESMI was presented followed by a synopsis of the representation schema, methods, techniques, procedures as well as cycle deliverables. The Object Modeling Notation (Rumbaugh et al,1991) is the representation schema recommended for use with the ESMI. The most relevant methods include OOA for analysis, OOD for design as well as OOP for programming. Relevant techniques include those for performing a cost-benefit analysis, for risk management and SQA as well as rapid prototyping and software reuse. Procedures for cost estimation (an integral task of cost-benefit analysis) as well as for the modelling of the system were identified. The various cycle deliverables of the ESMI were summarised in Section 4.3.7. The integration methodology is based on the ESMI but due to the limited scope of this dissertation, it will not be covered in further detail here.

Project management was covered in the final section of the chapter. Project management should ensure that the integration project is completed on time, within budget and that the requirements of the user are satisfied. Major categories of project management tasks were identified. They are project planning, project organisation, project directing, project control as well as report writing. **Project planning** comprises activities to ensure that the integration project is completed on time and within budget such as the compiling of the WBS and CBS during the Architecture Cycle. **Project organisation** entails project structuring, the enforcement of standard procedures as well as logistical support. Project structuring is usually performed during the Architecture Cycle whereas the enforcement of standard procedures and logistical support are activities of all cycles of the ESMI.

**Project directing** comprises the management of people involved in a project as well as ensuring an integrated system of high quality. Project directing is performed throughout the integration life-cycle. **Project control** involves monitoring the integration project and taking corrective action when integration does not proceed according to the SPMP. In addition, it also involves configuration management, quality management and risk management which is performed throughout the cycles of the ESMI. **Report writing** entails the compilation and updating of the various reports to be delivered during the integration process.

Some management tasks are unique to a cycle of the ESMI whereas others are generic and should be done during every cycle. Management of a large project such as the integration project, cannot be seen as a task of a single member of the development team. Instead, the management of an integration project should be seen as a task performed at the different levels of software development, i.e. at the Universal, Worldly and Atomic Levels. The Universal Level provides a global view of an integration project to be taken by top management. The Worldly Level guides middle management through the sequence of tasks for each cycle of the ESMI. The Atomic Level provides junior management with a detailed description of the individual tasks of the Worldly Level in order to guide a group of integration project participants to complete a specific task.

# CHAPTER 5
# METHODS, TECHNIQUES, PROCEDURES AND AUTOMATED SUPPORT FOR THE ESMI

## 5.1 Introduction

The ESMI is supported by various methods, techniques and procedures. As a result of the limited scope of this dissertation only a few of these are discussed here. The most relevant methods of the ESMI include OOA, OOD and OOP. OOA is used during the Analysis Cycle to provide an OO description of the problem domain in order to obtain a complete and consistent statement of the requirements. OOD is a method for design which involves the refinement of the analysis models to obtain a detailed basis for implementation. OOP is used for implementing the design in order to obtain an executable software system.

The most relevant techniques of the ESMI include those for cost-benefit analysis, risk management, Software Quality Assurance (SQA) as well as rapid prototyping and software reuse. A cost-benefit analysis involves the comparison of estimated future benefits against projected future costs. Risk management is performed throughout the integration project and is essential in order to prevent risk items from becoming either threats to the successful outcome of an integration project or to result in rework during the integration project. Risk assessment is a step of risk management performed during the second quadrant of all the cycles of the ESMI and it comprises risk identification, risk analysis and risk prioritisation. Rapid prototyping is used during risk analysis as well as for the gathering of requirements. SQA techniques should be performed throughout all cycles of the ESMI, but it is particularly used during the regular reviews of the fourth quadrant of all cycles. It is essential to ensure an integrated system of high quality. Software reuse results in a shorter integration process as well as more reliable software and technical members of the integration team should be encouraged to reuse software components whenever possible.

Procedures for both the modelling of the system to be integrated as well as cost estimation are reviewed. The methods, techniques and procedures of the ESMI should be automated as far as possible. Automated support including a repository, will increase productivity during the integration life-cycle.

## 5.2. Methods for the ESMI

OOA, OOD and OOP are now discussed as methods for the ESMI.

### 5.2.1 Object-Oriented Analysis (OOA)

Analysis is the study of a problem domain leading to a complete, consistent and feasible statement of the requirements of the system, prior to taking some action. Requirements include functional operations as well as non-functional requirements in the form of quantified operational characteristics such as ease of use, reliability, availability, maintainability and performance. Requirements also include any applicable design constraints. OO shifts development effort into analysis. Traditional approaches to analysis map from problem domain indirectly to functions and subfunctions (functional decomposition) or from problem domain to data flows and bubbles (data flow approach). OOA directly maps the problem domain and system responsibility into a model. It is used during the third quadrant of the Analysis Cycle of the ESMI (Section 4.3.2).

Booch (1994) defines OOA as

> ***" a method of analysis that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain."***

OOA is based on the uniform application of the principles of OO (reviewed in Appendix C). With OOA, the domain of discourse in the real world is modelled in terms of object types and what happens to those object types. OOA is intended to facilitate the understanding, formulation and communication of OO descriptions of the problem domain. It increases the internal consistency of analysis results by treating the different analysis activities as a natural whole. Inheritance is used to explicitly represent commonality. Specifications build with OOA are resilient to change and analysis results can be reused. In addition, OOA provides for a continuum and consistent underlying representation for analysis and design. According to Coad and Yourdon (1991), OOA is very useful for systems with extensive responsibilities as well as for systems with many classes and objects. The legacy systems of Chapter 1 are likely to have these characteristics.

Many authors hold the view that the boundaries between OOA and OOD are blurred (Wirfs-Brock et al,1990; Stroustrup,1991; Schach,1996). The focus of each is, however, quite distinct. During OOA the real-world is modelled by discovering the classes and objects that form the vocabulary of the domain of discourse whereas during OOD the abstractions and mechanisms that provide the behaviour which this model requires, are invented (Booch,1994). OOA therefore deals with "problem-domain" objects whereas OOD expands these objects into "solution domain" objects.

Shlaer et al (1988) describes an approach to OOA which consists of three phases: static modelling of objects, dynamic modelling of states and events as well as functional modelling. According to Rumbaugh et al (1991) OOA comprises:

1. writing a problem statement;
2. building an Object Model which consists of the object model diagram and the repository;
3. developing a Dynamic Model which consists of state diagrams and a global event flow diagram;
4. constructing a Functional Model which consists of data flow diagrams (DFD's) as well as constraints;
5. verifying, iterating and refining the three models.

Coad and Yourdon (1991) identify five major activities of OOA:

1. finding classes and objects;
2. identifying structures;
3. identifying subjects;
4. defining attributes;
5. defining services.

These activities are not sequential steps and there is no significant order in which one should move from one activity to the next. The activities guide the analyst from high levels of abstraction (e.g.

problem domain classes and objects) to increasingly lower levels of abstraction (structures, attributes and services). The OOA model is presented in five layers:

1. Subject Layer;
2. Class-&-Object Layer;
3. Structure Layer;
4. Attribute Layer;
5. Service Layer.

The five layers gradually present more and more detail. The Subject Layer is useful for more complex problem domains, to guide readers through a larger model and partition work into packages.

Booch (1994) emphasises that analysis focuses upon behaviour and the objective is to identify classes and objects (as well as their roles, responsibilities and collaborations) which form the vocabulary of the problem domain. It is inappropriate to address issues of class design, representation or other tactical decisions during this phase.

By focusing on behaviour, the function points of the system may be identified. Function points denote the external observable and testable behaviours of a system which are often the mappings of inputs to outputs. From a user's perspective, a function point represent some primary activity of a system in response to some event whereas from an analyst's perspective, it represents a fixed measure of behaviour.

The two primary activities of the method of Booch (1994) are domain analysis and scenario planning. Domain analysis seeks to identify the classes and objects that are common to a specific problem domain. Scenario planning is the central activity of Booch's method and makes use of techniques such as Use-Case and Behaviour Analysis as well as Class / Responsibilities / Collaborators (CRC) Cards. Quality-assurance personnel should participate in scenario planning, since scenarios represent behaviours that can be tested.

Jacobson et al (1992) defines Use-Case as

> ***"a particular form or pattern or exemplar of usage, a scenario that begins with some user of the system initiating some transaction or sequence of interrelated events."***

Use-Case Analysis may be applied as early as requirements analysis and it involves the enumeration of the scenarios (without elaborating upon them) which are fundamental to the operation of the system by users, domain experts and the development team. Each scenario is then studied by means of storyboard techniques, i.e. the objects which participate in the scenario, the responsibilities of each object as well as the operations invoked by the objects to collaborate with other objects, are identified (Booch,1994).

Behaviour analysis focuses upon dynamic behaviour as the primary source of classes and objects, i.e. classes are based upon groups of objects that exhibit similar behaviour. A CRC Card is a 3x5 index card used by an analyst to write the name of a class, its responsibilities and its collaborators (Beck & Cunningham,1989). One card is created for each class identified as relevant to the scenario. As the integration team walks through the scenario, new responsibilities may be assigned to an existing class, certain responsibilities may be grouped to form a new class and the responsibilities of one class may be divided into more refined classes. The CRC cards can be arranged according to the generalisation / specialisation or aggregation hierarchies among classes to reflect the static semantics of the scenario, as well as according to the flow of messages among prototypical instances of each class to reflect the dynamic semantics of the scenario (Booch,1994).

Scenario planning typically involves:

- the identification of all primary function points of the system, where possible, grouped into clusters of functionally related behaviours according to hierarchies of functions;
- the capturing of the descriptions of the functions of the system by making use of scenarios. Each scenario represents some particular function point. Techniques such as Use-Case and Behaviour Analysis as well as CRC Cards are used to storyboard scenarios. Document the scenarios by means of object diagrams which illustrate the

objects that initiate or contribute to behaviour and that collaborate to complete the activities of the scenario. Documentation should include a script, showing the events that trigger each scenario and the resulting ordering of actions as well as assumptions, constraints and performance issues of each scenario;

- the generation of secondary scenarios which illustrate behaviour under exceptional conditions, if needed;
- the development of a finite state machine for the classes of objects which are significant or essential to a scenario. A finite state machine shows the life-cycle of certain objects;
- the search for patterns among scenarios and the expression of these patterns in terms of more abstract, generalised scenarios or in terms of class diagrams showing the associations among key abstractions;
- the update of the evolving repository to include the new classes and objects identified for each scenario, along with their roles and responsiblities.

All the above-mentioned methods for OOA have their strengths and weaknesses. Iivari (1995) compares six methods for OOA. He is of the opinion that OOA should cover structure abstraction (i.e. object classes and subsystems), their functionality (i.e. operations, methods and responsibilities) as well as their interaction. The method proposed by Rumbaugh et al (1991) conforms to these requirements. According to Iivari (1995) the strengths of Rumbaugh et al's (1991) method are:

- the balanced attention to the three perspectives of structure, function and behaviour;
- the support provided for object modelling, as well as
- the fact that the method is generally well-defined.

Iivari (1995) regards the facts that the perspectives taken are not clearly integrated and that minimal support is provided for modelling at the organisational level as weaknesses of the method. The advantages (which are corroborated by the author's study of Rumbaugh et al's method as part of a special topic module on OO), however, outweigh these disadvantages and despite the existence of

the latter, the method of Rumbaugh et al (1991) will be recommended for use with the ESMI. Other methods for OOA, such as the method of Booch (1994), are also considered good candidates which can be used with the ESMI.

The process and notations used by an OOA method should assist in the production of a coherent set of models which constitute a reasoned and convincing description of the problem domain and from which a software system can be developed. The notations used should ensure precise communication about complex domains. Consequently the notations should be:

- unambiguous, i.e. the models should have a single discernible meaning;
- abstract, i.e. a model should not be cluttered with unnecessary detail;
- consistent, i.e. it should be possible to check whether different models of the same system conflict.

The logical model is the result of analysis. OOA ends when the resulting analysis deliverables have been validated by the user, domain expert, integration team as well as the SQA team. The deliverables of OOA serve as the models from which an OOD may be started.

### 5.2.2 Object-Oriented Design (OOD)

Reasons for distinguishing a design phase are the following:-

- In order to seamlessly change the analysis models to source code, the objects should be refined.
- The actual system must be adopted to the implementation environment. Issues such as performance requirements, concurrency, the DBMS to be used, should be considered.

During design the physical issues of the solution are addressed. The analysis models will be further refined in the light of the actual implementation environment. OOD is used during the third quadrant of the Design Cycle of the ESMI (Section 4.3.2) and involves determining the implementation of each class, association, attribute and operation. The analysis classes are

transformed into a computerised model that belongs to the solution domain. In general, OOD is concerned with the dynamic behaviour of objects.

Booch (1994) defines OOD as

> ***" a method of design encompassing the process of object-oriented decomposition and a notation for depicting both logical and physical, as well as static and dynamic models of a system."***

Wirfs-Brock et al (1990) propose an OOD method which consists of an exploratory phase followed by an analytical phase. The exploratory phase involves the finding of the classes from the problem domain, the determination of the knowledge and operations for which each class is responsible as well as the determination of the collaborations between classes. During the analytical phase the inheritance relationships between classes are examined in order to build class hierarchies and the collaborations between objects are streamlined. Wirfs-Brock et al (1990) are of the opinion that this method will provide a complete set of class specifications to be implemented. Hierarchy graphs are used to show the inheritance relationships between classes, Venn diagrams are used to show which responsibilities are common between classes and Collaboration graphs are used to show the classes and subsystems within a system and the paths of collaboration between them.

The method of Rumbaugh et al (1991) implies the elaboration of the analysis models in order to provide a detailed basis for implementation. Design documentation includes the detailed Object, Dynamic and Functional Models. The method involves:

- obtaining the operations for the Object Model from the other models, i.e. each process in the Functional Model as well as each event in the Dynamic Model represent an operation;
- designing algorithms to implement operations at minimal cost, i.e. selecting data structures for the algorithms, defining new internal classes and operations, and assigning responsibility for operations not associated with a single class;

- optimising access paths to the data, i.e. adding redundant associations to minimise access cost and maximise convenience, rearranging computations for greater efficiency, and saving derived values to avoid recomputation of complicated expressions;
- implementing software control;
- increasing inheritance by adjusting class structure, i.e. rearranging and adjusting classes and operations, abstracting common behaviour out of groups of classes as well as using delegation to share behaviour where inheritance is semantically invalid;
- designing the implementation of associations, i.e. analysing the scope of associations and implementing each association either as a distinct object or by adding object-valued attributes to the classes of the association;
- determining the precise representation of object attributes, as well as
- packaging classes and associations to form modules.

According to Booch (1994), primary activities of OOD include Architectural Planning, Tactical Design and Release Planning. Architectural Planning involves planning the layers and partitions of the overall system and includes a logical decomposition which represents a clustering of classes, as well as a physical decomposition which represents a clustering of modules and the allocation of functions to different processors. The objective is to create a domain-specific application framework which can be successively refined. Architectural Planning comprises:-

- Allocate the clustering of function points from the analysis products to layers and partitions of the architecture. Functions that build upon one another should fall into different layers whereas functions that collaborate to produce behaviours at a similar level of abstraction should fall into partitions which represent peer services.
- Validate the architecture by creating an executable release which partially satisfies the semantics of a few system scenarios as derived from analysis.
- Instrument that architecture and assess its weaknesses and strengths. Identify the risk of each key architectural interface so that resources can be allocated meaningfully as evolution commences.

Tactical Design involves making decisions regarding policies. Tactical Design typically comprises:-

- List the common policies that must be addressed by disparate elements of the architecture. Some policies are foundational, i.e. they address domain-independent issues such as error handling and memory management whereas other are domain-specific, i.e. they include idioms and mechanisms which are relevant to the specific domain such as database management in information systems.
- Develop a scenario for each common policy in order to describe the semantics of that policy.
- Document each policy and complete a peer walkthrough in order to broadcast its architectural vision.

Release Planning serves to identify a controlled series of architectural releases, each growing in functionality. A typical order of events includes:

- organise the scenarios identified during analysis, in order of foundational to subordinate behaviours;
- allocate the related function points to a series of architectural releases whose final delivery represents the production system;
- adjust the goals and schedules of this stream of releases so that delivery dates are sufficiently seperated to allow adequate development time and to ensure that releases are synchronised with other development activities, such as documentation;
- do task planning, i.e. identify a WBS and identify development resources which are necessary to achieve each architectural release.

Once again the method of Rumbaugh et al (1991) is recommended for use with the ESMI but other good candidates such as the method of Booch (1994) can also be used. Rumbaugh et al's (1991) method for OOD provides a multi-dimensional perspective which is required by large legacy systems and it involves the elaboration of the models of his OOA method.

The products of OOD can be used as "blueprints" for completely implementing a system using OOP methods (Booch,1994).

### 5.2.3 Object-Oriented Programming (OOP)

OOP is a method which makes use of packaging technology. It allows the packaging of functionality so that it can be reused in different applications. OOP is used during the third quadrant of the Design Cycle (Section 4.3.2). According to Cox (1986) conventional programming tools emphasise the relationship between a programmer and his code, while OOP emphasises the relationship between suppliers and consumers of code. Cardelli and Wegner (1985) reckon an OOP language should satisfy the following requirements:

- provide support to objects that are data abstractions with an interface of named operations and a hidden local state;
- each object is an instance of an associated class;
- classes may inherit attributes from superclasses.

Booch (1994) defines OOP as

> ***"a method of implementation in which programs are organised as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships. In such programs, classes are generally viewed as static, whereas objects typically have a much more dynamic nature, which is encouraged by the existence of dynamic binding and polymorphism."***

Flow of control through a program is achieved by means of the invocation of the methods of objects. Objects communicate with one another by passing messages. A message is a request for an object to carry out the sequence of actions in one or more of the methods of its class. A message contains a name which identifies its destination, and may also contain some arguments or

parameters. This is similar to a traditional function or procedure call. In the language Smalltalk a message can be defined as (Rumbaugh et al,1991):

> ***" an invocation of an operation on an object, comprising an operation name and a list of argument values."***

The message, when received, will cause the invocation of an appropriate method within the receiving object. For example, a manager may be responsible for increasing the salaries of his sub-ordinate employees. This would involve an object of the class *Manager* sending a message *Increase_Sal* to each object of the class *Employee* for which that manager has responsibility. When a message is sent, the sender object may stop processing and pass control to the receiver object. When the receiver completes processing it may send a response to the originator. Responses may be synchronous (single thread of control) or asynchronous (multiple threads of control, as in a parallel processing system). The sending object is often referred to as the client and the receiving object as the server.

Object-based programming languages such as earlier versions of Ada, provide direct support for data abstraction and classes, but not for inheritance. In order to be OO, a language must be object-based as well as provide support for inheritance and polymorphism. Eiffel and Smalltalk (developed by the Software Concepts Group at Xerox PARC) are considered OO languages. In the context of client/server development, the trend in the IS/T industry is towards visual programming environments supporting OO such as Delphi. Such an environment is a good candidate for use with the ESMI. The final deliverable of OOP is an executable software system.

## 5.3 Techniques for the ESMI

Techniques of the ESMI include those for cost-benefit analysis, risk management, rapid prototyping, Software Quality Assurance (SQA) as well as software reuse.

### 5.3.1 Cost-Benefit Analysis

The evaluation of economic feasibility requires the organisation to conduct a cost-benefit analysis of the integration project based on the scope and objectives that have been determined. A cost-benefit analysis involves the comparison of estimated future benefits against projected future costs. A procedure for cost estimation is presented in Section 5.4.2.

Two types of cost must be considered, namely:

- one-time costs associated with the integration project, and
- recurring costs.

One-time costs associated with the integration project include the time of users and systems personnel devoted to the integration project as well as computer hardware and software needed for developing and implementing the integration project. Recurring costs include the time of personnel needed to operate and manage the integrated systems as well as hardware and software acquisition costs, if new technology must be obtained.

During the Feasibility Cycle, the costs of integration versus non-integration are analysed. Non-integration costs include the costs of upgrades, differential cost of software licencing and the differential cost of service and support over the period considered. During all cycles of the integration project the costs and benefits of various alternative strategies are compared. The strategy with the largest difference between benefits and costs should be selected as the optimal strategy.

Once the costs are determined, they are weighed against the expected benefits. Intangible benefits, such as the increased availability of information, are difficult to measure in rands and cents. Assumptions must be made in order to assign a rand value to intangible benefits. However, certain direct benefits can be measured, including decreases in personnel and faster response time.

Major classes of techniques available for software cost estimation include (Boehm,1984; Marco,1990; Lederer & Prasad,1992):-

1. Algorithmic Models provide one or more algorithms which produce a software cost estimate as a function of a number of variables which are considered to be major cost drivers. Numerous models have been suggested for the estimation of costs and schedules (Wolverton,1974; Myers,1978; Goldberg & Lorin,1980; Boehm,1984). A problem with all these models is that they depend on past experience for calibration. If the data was gathered only on small product developments, then estimating the cost and schedule for the integration project becomes suspect. The Putnam model (Meyers,1978) works reasonably well for very large projects, although the model's estimates lack precision.

   In the Constructive Cost Model (COCOMO) equations based upon size, program, computer, personnel and project attributes are used to determine software costs. The Basic, Intermediate and Detailed COCOMO's are distinguished. The Intermediate COCOMO and Detailed COCOMO are significantly more accurate than the Basic COCOMO. After investigating the project data and related COCOMO estimates for the 63 projects in the COCOMO database, Boehm (1984) concludes that COCOMO estimates are reasonably accurate for traditional, single increment software developments, but considerably below actuals for incremental development projects. The COCOMO model does not make provision for more than one cycle of integration to be executed concurrently and it does not take into account the reuse of software components promoted by OO (Van der Walt,1993). With large projects such as an integration project, the concurrent execution of cycles is essential as some subsystems may be in the Design Cycle while others are in the Implementation Cycle. Balda & Gustafson (1990) propose extensions of the Intermediate COCOMO that consider reuse and rapid prototyping.

2. Expert Judgement techniques involve consulting one or more experts, who use their experience and understanding of the proposed project to arrive at an estimate of its cost. The wideband Delphi technique (Farquhar,1970; Boehm et al,1974) can assist in combining the estimates of a number of experts into a single estimate. It comprises the following steps:

(i) a coordinator presents each expert with a specification and an estimation form;

(ii) the coordinator organises a group meeting in which the experts discuss estimation issues with the coordinator and each other;

(iii) the experts fill out estimation forms anonymously;

(iv) the coordinator prepares and distributes a summary of the estimates on an iteration form;

(v) the coordinator organises a group meeting to allow the experts to discuss points where their estimates varied widely;

(vi) the experts fill out estimation forms anonymously and Steps (iv) to (vi) are iterated for as many rounds as appropriate.

3. Estimation by Analogy involves deducing by comparing with one or more completed projects to relate their actual costs to an estimate of the cost of a similar new project. It is equivalent to the similarities and differences estimating technique of Wolverton (1974). A main strength is that the estimate is based on actual experience in a previous project. It is, however, difficult to determine the degree to which the previous project is representative of the constraints, techniques, personnel and functional aspects of the new project.

4. Parkinsonian estimation is based upon Parkinson's Law (Parkinson,1957) which states that work expands to fill the time budgeted for it. The following is an example of a Parkinsonian estimate:

> ***"This flight control software must fit on a 65 536-word machine; its size will therefore be roughly 65 000 words. It must be done in 18 months and there are 10 people available to work on it. It will therefore take roughly 180 man-months."***

Parkinsonian estimation is not recommended as it is not accurate and reinforces poor software development practices.

5. Price-to-Win estimation involves the adjustment of the cost estimate in order to meet the constraints of a project. As a result the money or schedule usually runs out before the project is completed. This type of estimation is not recommended as it generally produces large overruns.

6. Top-Down estimation comprises the derivation of an overall cost estimate for the project, based upon the global properties of the software product. The total cost is then split among the various components. The system level focus of top-down estimating is a major advantage and as it is based upon previous experience on completed projects, it is likely to estimate the costs of system level functions (e.g. the compilation of user manuals and configuration management) accurately. Disadvantages include the lack of a detailed basis for cost justification, difficult low level technical problems that are likely to escalate costs and are often not identified, and components of the software to be developed are sometimes ignored.

7. Bottom-Up estimation comprises the estimation of the cost of each software component by an individual who is usually responsible for developing the component. In order to obtain an estimated cost for the overall product, these costs are summed. Disadvantages are that system level costs associated with software development, e.g. project management, configuration management and SQA, are often ignored and Bottom-Up estimation therefore often results in underestimation. Advantages are that the estimates will be based on a more detailed understanding of the components as they are done by the person responsible for the success of the involved component. Bottom-Up estimation complements Top-Down estimation as the one's weaknesses tends to be the other's strengths, and vice versa.

Although the Parkinson and Price-to-Win methods are unacceptable and do not produce sound cost estimates, none of the above-mentioned alternatives is better than the others in all respects and their strengths and weaknesses are complementary. It is therefore important to use a combination of techniques and to iterate the estimates obtained from each.

### 5.3.2 Risk Management

The objectives of Risk Management, with respect to the integration of legacy systems with the client/server environment, are to identify, address and eliminate risk items before they become either threats to the successful operation of the integrated system or major sources of rework during the integration process (Charette,1991). As illustrated in Figure 5.1, Boehm (1989) considers the primary steps of Risk Management to be Risk Assessment and Risk Control.

Risk Assessment involves Risk Identification, Risk Analysis and Risk Prioritisation, whereas Risk Control involves Risk Management Planning, Risk Resolution and Risk Monitoring. Risk Assessment is a task of the second quadrant of all cycles of the ESMI whereas Risk Control is a management task performed throughout the integration project. Project Management is covered in Section 4.4.

**Risk Identification** will result in lists of specific risk items likely to endanger the successful outcome of an integration project. Risk Identification techniques include risk identification checklists, comparison with previous experience, decomposition and the examination of decision drivers. Decomposition involves the identification of poorly-described areas within the integration project's plans and specification as well as the decomposition of these areas into their constituent elements. Frequently these areas contain serious project risk items.

Decision Driver Analysis involves the analysis of the sources of key decisions regarding the integration project. If a decision has been driven by factors other than technical and management achievability (e.g. politically-driven or marketing-driven decisions), it will frequently be the source of a critical software risk item.

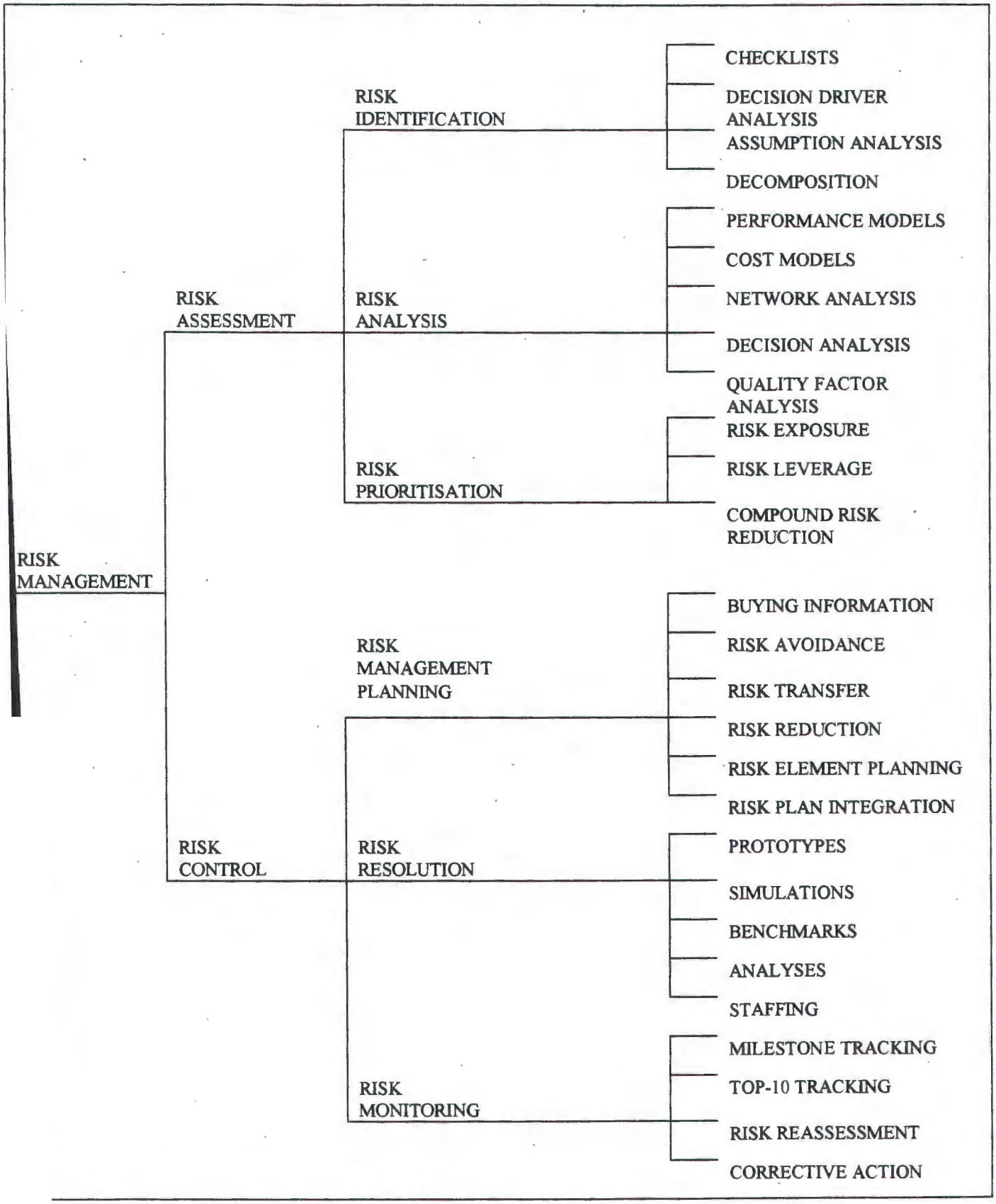


**Figure 5.1** Software Risk Management Steps

One large company developed a list of 54 questions for measuring the risk of a project after analysing its experience with successful and unsuccessful projects. Some of these questions are presented in Appendix D. These questions highlight the risks as well as suggest alternative ways of conceiving and managing a project. If the initial aggregate risk score seems high, analysis of the answers may suggest ways of lessening the risk, for example through reduced scope or lower-level technology. These questions should be answered both prior to senior management's approval of the proposal as well as periodically during implementation to reveal any major changes. The higher the aggregate risk score, the higher should the level of approval be. This approach will ensure that top managers are aware of significant risks and are making appropriate risk strategic-benefit trade-offs.

**Risk Analysis** involves the assessment of both the probability and magnitude of loss associated with each of the identified risk items as well as the assessments of composite risks involved in risk-item interaction (Charette,1989). Risk analysis techniques include network analysis, decision tree analysis and cost risk analysis. Network analysis involves the breaking down of schedule areas into an activity network or PERT chart of its constituent tasks. A PERT chart is a network or graph whose nodes represent project activities and their associated durations, and whose links represent precedence relations between pairs of activities, i.e. if there is a link (arrow) from node A to node B, then activity A must be completed before activity B can start. Various aspects of the PERT chart, such as the presence of highly overlapping paths and multiple critical-path situations, are then analysed for high-risk features. The decision tree structures risk situations in terms of the possible decisions one can make and in terms of the risk exposure factors associated with each decision option. The various options are characterised by their possible outcomes, with their probabilities of occurence as well as the resulting cost or benefit of each outcome.

Cost risk analysis makes use of a software cost estimation model such as the Constructive Cost Model (COCOMO) (discussed in Section 5.3.1). Most software cost models use a series of parameters called cost drivers, e.g. product size, personnel experience and capability as well as hardware constraints, which describe aspects of the project determined to significantly affect its cost. The COCOMO estimates the nominal project effort in man-months as a function of the estimated size of the software product in thousands of delivered source instructions. The nominal

effort is the amount of effort a project of this size would take if it was perfectly average with respect to the other cost drivers. The effects of the project, not being perfectly average in all dimensions, are then calculated by using a set of project effort multipliers as functions of the project's ratings for a set of cost driver variables. A revised COCOMO man-month estimate is then obtained by multiplying together the individual effort multipliers for each of the cost drivers into an overall effort adjustment factor, and multiplying the nominal man-months by the effort adjustment factor. The project's estimated cost can be determined by multiplying the estimated project man-months by an average rand cost per man-month.

**Risk Prioritisation** will produce a prioritised ordering of the risk items identified and analysed. Techniques for the prioritising of risk items include Risk Exposure (RE) analysis and Risk Reduction Leverage (RRL) analysis (Boehm,1989). The quantity RE is defined by the relationship:

RE = Prob(UO) * Loss(UO),

where Prob(UO) is the probability of an unsatisfactory outcome and Loss(UO) is the loss to the parties affected if the outcome is unsatisfactory. RE contours can be presented graphically as functions of Prob(UO) and Loss(UO), where Loss(UO) is assessed on a scale of 0 (no loss) to 1 (complete loss). A project with a risk item in the critical RE area, should attempt to reduce its RE by reducing the Prob(UO) or the Loss(UO).

RRL is a measure of the relative cost-benefit of performing various candidate risk reduction activities. The RRL quantity is defined by:

RRL = RE (before) - RE (after) / Risk Reduction Cost,

where RE (before) is the RE before initiating the risk reduction effort and RE (after) is the RE after the reduction effort has been completed.

**Risk Management Planning** will produce plans for addressing each risk item and includes the coordination of the individual risk-item plans with each other as well as with the overall integration

project plan (Charette,1989). Techniques include checklists of risk resolution techniques, cost-benefit analysis as well as decision analysis of the relative cost and effectiveness of alternative risk-resolution approaches. Boehm (1989) illustrates the Risk Management Planning process as in Figure 5.2. Inputs to the process are:

- the list of prioritised risk items resulting from the risk assessment process;
- the candidate risk resolution techniques considered in determining the RRL quantities for each risk item, and
- the results of the RRL cost-benefit analyses, including the budget, schedule and resources required to achieve the corresponding reductions in RE.

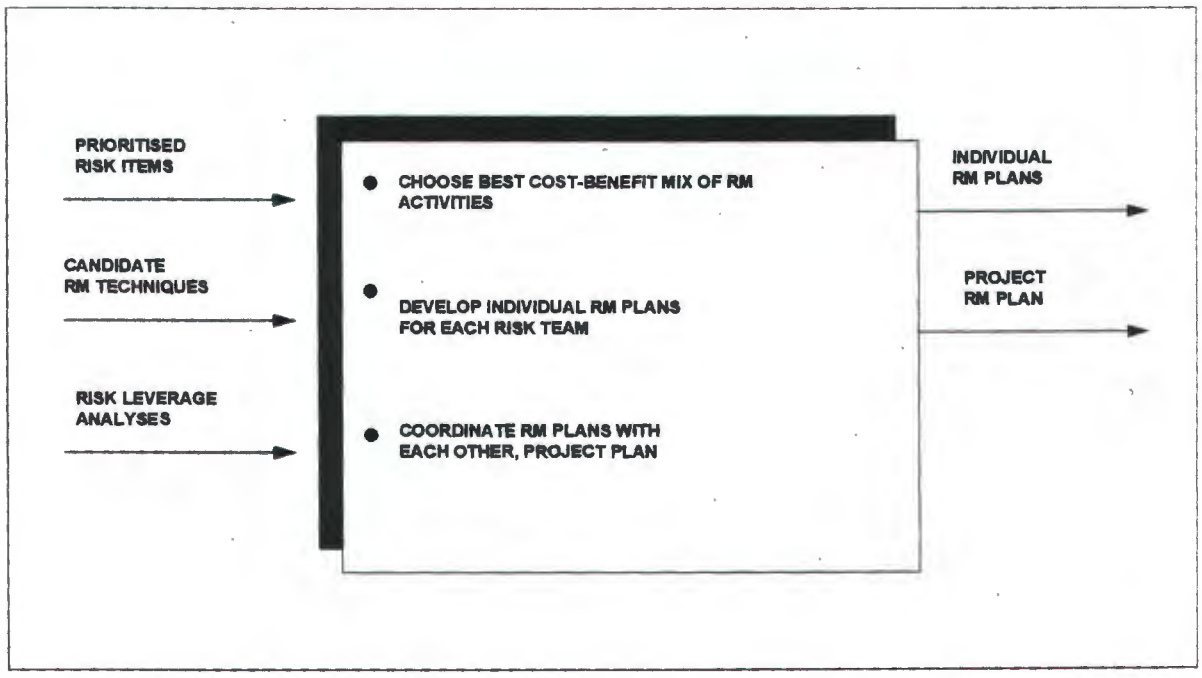


**Figure 5.2** Risk Management (RM) Planning Process

Steps of the Risk Management Planning process proceed in a concurrent and interactive manner. They include choosing the best cost-benefit mix of risk resolution activities, developing individual risk management plans for each risk item as well as coordinating individual risk management plans with each other and with the integration SPMP. All the individual risk management plans are

documented in the Project Risk Management Plan (PRMP) together with an overview of how the individual plans fit together.

Fundamentally, the **Risk Resolution** process involves executing the PRMP, i.e. communicating objectives, establishing responsibilities and performing to satisfy the milestone criteria in the plan. Risk Resolution results in either the elimination or the resolution of risk items. Risks can be resolved by relaxing requirements. Techniques include prototyping (discussed in Section 5.3.3), simulations, benchmarks, design-to-cost approaches as well as incremental development.

**Risk Monitoring** during the integration project involves keeping track of both the progress towards risk resolution as well as appropriate corrective action taken (Charette,1991). Techniques include milestone tracking and a top-10 risk-item list which is highlighted at regular project reviews. Risk management plan milestone tracking involves the monitoring of both the milestones in each individual risk item's risk management plan as well as in the PRMP. This can be done by regular reviews to ensure that the milestones have been accomplished and, if needed, corrective action determined and effected. Project top-10 risk-item tracking involves ranking the project's most significant risk items, establishing a regular schedule for reviewing the project's progress by higher management, beginning each review with a summary of the progress on the top-10 risk items and focusing the review on dealing with problems in resolving the risk items.

Evaluating the risks involved in the integration of a legacy system with the client/server environment, should be seen as a standard part of the feasibility study. According to Peppard et al (1993), direct risks are those that operate to defeat a project before it becomes operational, whereas indirect risks are those generated by the success of a project after it becomes operational. Direct risks can be further subdivided into those that are inherent to the situation and those that are a result of poor management practices. Direct management-type risks with regard to integration, include:

- cost overruns, because of the considerable associated financial uncertainty of systems integration;

- overreliance on consultants. They should be chosen with great caution as they make their money up front and seldom have to live with their failures.

Direct integration-inherent risks include:

- resistance to change, i.e. the integration of systems may result in major changes throughout an organisation; schemes need to be implemented to minimise resistance associated with change;
- changing circumstances and requirements; change is the only constant in organisations, and it should be possible to adapt integration projects to changing requirements;
- uncertainties with emerging technology; integration often involves new technology, coupled with a lack of skills within the organisation.

Indirect risks associated with systems integration include:

- insatiability, i.e. the increased availability and quality of information can result in greater demands for a more extensive usage of the systems involved, and
- easy access, i.e. the increased availability of information, provided by integration, may simplify access to data and therefore increase the possibility of abuse.

Boehm (1989) distinguishes between generic risks and project-specific risks. Generic risks are common to all projects and are covered by standard development plan techniques. Project-specific risks are addressed by project-specific risk management plans and reflect a specific aspect of a given project. The most common project-specific risks are personnel shortfalls, unrealistic schedules and budgets, inappropriate requirements, shortfalls in external components and tasks as well as technology shortfalls.

### 5.3.3 Rapid Prototyping

Rapid prototyping is an iterative technique that allows for the rapid creation of a working model of the system under development. It is useful for gathering requirements and to enhance the

understanding of the proposed system. Prototypes of critical areas and areas identified in the risk analysis of the system may be done to reduce risk.

Lantz (1985) defines software prototyping as

> ***"a collection of information system development methods based on building and using a model of a system for designing, implementing, testing, and installing the system."***

An assumption of prototyping is that it is not necessary to know the full requirements of a system in order to build the model. The full requirements are an important product of prototyping. The initial version of the model does not contain all the processing and validation rules that the system will finally have. The model is used for designing, implementing, testing and installing the system, i.e. as those working with the prototype modify it and add to it, they will be completing the design of the system as well as implement and test the system. The prototype is therefore a vehicle for designing the final version of the system.

As the prototyping process continues, newer versions of programs, that perform more closely to those of the final system, will replace the original versions. The model is therefore used to implement as well as test the system. The prototype eventually becomes the system.

Although prototyping is sometimes regarded as a "quick and dirty" approach used for urgent systems development projects, it has significant advantages. The advantages of prototyping include the improvement of communication between the software engineer and the user. Users usually have difficulty visualising what they want a system to do. Prototyping allows the user to be more involved in the integration process and therefore ensures the delivering of functionally correct software that meets the user's true requirements. For this reason prototyping could also be considered a SQA technique. Prototyping will reduce the total cost of system development and maintenance because errors are detected early in the development process. It contributes to producing the right system the first time (Connell & Shafer,1989).

The advantages of prototypes from the viewpoint of enhancing system quality through better requirements specification, can be categorised into improved functional requirements, improved interaction requirements and easier evolution of requirements (Keen & Gambino,1980; Connell & Shafer,1989). The prototype system reflects the developer's interpretation of the user's needs and will reveal misunderstanding as well as uncover errors in functional logic. The GUI requirements for a system are not always directly addressed in requirements specification. As a consequence, the resultant system may often contain the correct functions, but the GUI may either discourage its use or introduce errors in usage. A prototype is useful to convey the nature of a proposed interactive system. The evolution of requirements is most important in environments like Decision Support Systems (DSS's), where the user needs to employ the system in open-ended ways and no pattern of use can be accurately predicted until some experience is available.

Three categories of prototyping techniques are identified. They are scenarios or simulations, demonstration systems and "version 0" limited working systems. A scenario or simulation presents the user with an example of actual system usage, but only simulates the processing of user data or queries. The eventual application logic is not developed, but some of the development work on the scenario may be applied to the production system. It is used for addressing GUI and some functional requirements. A demonstration system processes a limited range of user queries or data and uses limited files. Either the entire demonstration can be coded as a throw-away or some portion of the system can be applied to the production system. A demonstration provides more insight into processing logic but, because of the limited exposure of the user, it may not be as useful for evolutionary requirements. A "version 0" prototype is a working release of the system intended to be used under conditions of the production environment. It is designed as a test release, but the final system is usually built on version 0 by converting the prototype into a production system, i.e. completing the implementation of functions, adding requested alterations and generating the required documentation.

Prototyping was discussed in Section 3.3 in the context of software methodologies. In the integration process prototyping will be used as a risk-resolution technique during the second quadrant of all cycles of the ESMI (Section 4.3.2). In addition, it will be used for the development of GUI's as well as for a package selection process, where required. Installing the candidate

packages first as prototypes with test data, allows for the evaluation of their strengths and weaknesses and assists in determining how they meet requirements. Prototyping is useful if enhancements and/or modifications to the legacy system are required. Only the part of the system which is planned to be changed needs to be prototyped, e.g. in a test environment with test data. After system testing and acceptance testing in the test environment, the changes become part of the production legacy system.

The use of prototypes has been advocated as cost-effective even without the presence of support tools. Carey and Mason (1983) distinguish a variety of tools which were employed in different situations. They include 4GL's and relational DBMS's.

### 5.3.4 Software Quality Assurance (SQA)

The quality of the integrated system is reflected by the extent to which the integrated system satisfies or exceeds its specifications. The IEEE Standard for SQA (IEEE/ANSI 730-1984, 1984) defines quality assurance as:

> ***"a planned and systematic pattern of all actions necessary to provide adequate confidence that the item or product conforms to established technical requirements."***

Quality should be the primary driver of the entire integration process. SQA involves those system management processes, systems design methods as well as software development techniques and tools that act to ensure that the resulting integrated system meets or exceeds a set of multi-attributed standards of excellence. It involves the examination of deliverables at all cycles of the life-cycle of the integration process, to determine concordance with requirements and specifications. The specifications include system and software requirements specifications, documentation specifications as well as specifications for design and management. The various requirements must be examined with respect to their concordance with various standards as well as with environmental and user needs. Appropriate attributes as well as quantifiable metrics of software quality are needed in order to obtain an early warning indicator of potential difficulties and

to make appropriate design changes early in the integration process. The use of appropriate metrics supports better responsiveness to the needs of users.

Quality infusion into the integration process involves identifying quality attributes important for a specific situation, determining importance weights for these attributes as well as defining and instrumenting operational methods of determining the attribute scores for specific integration approaches. Sage and Palmer (1990) suggest the construction of an attribute tree which will enable the meaning of each software quality attribute to become apparent through the hierarchical structure. This is the equivalent of Boehm's (1976) software quality characteristic tree. The detail of the construction of such an attribute tree is not discussed here, but can be referenced in one of the above-mentioned sources.

SQA is primarily concerned with the detection of the existence of faults as well as with the diagnosis of faults (i.e. the determination of the location and type of fault). SQA indicators should lead to the detection of errors, if any, and the diagnosis of these, such as to identify them as coding errors or logic errors. SQA and associated error detection, diagnosis and correction should be accomplished very early in the integration life-cycle.

Sage and Palmer (1990) state that although it is important to detect and correct errors, it is much better to establish life-cycle design and development procedures so that error occurrence is minimised. Every software professional involved in the integration process, is responsible for ensuring high quality software at all times, whereas the SQA group has additional responsibilities with regard to software quality.

The SQA group is responsible for ensuring the quality of the integration process, thereby ensuring the quality of the integrated system. The group should ensure the correctness of the integrated system as a whole, once it is completed. Members of the SQA group use verification techniques during each cycle's fourth quadrant to verify the deliverables produced during the involved cycle against the specifications and constraints, and to ensure that software is developed according to prescribed quality standards. The advantage of reviews done by a SQA group is that the different skills of the SQA group increase the chances of finding faults. Their responsibilities include:

- the development and implementation of the various SQA standards to which the deliverables must conform, as well as the establishment of the monitoring procedures, practices and policies for assuring compliance with those standards. These standards, procedures, practices and policies are documented in the Software Quality Assurance Plan (SQAP);
- the development and implementation of metrics, testing tools and other SQA techniques;
- the implementation of the resulting SQAP, which includes the documentation of a final SQA report.

There should be managerial independence between the integration team and the SQA group, i.e. the integration and SQA teams report to different managers, neither of whom are able to overrule the other. If serious faults are found in the integrated system as the delivery date approaches, managerial independence will prevent both the manager responsible for development from deciding to deliver faulty software on time, as well as the SQA manager from deciding to deliver the integrated system late while performing further testing.

The notion of testing is the primary tool of SQA. This includes the notion of software design for testability. Verification and validation (V & V) activities that are conducted throughout all cycles of the integration life-cycle are quality control techniques and are part of SQA. SQA is concerned with ensuring that the deliverable adheres to pre-defined standards, while V & V is concerned with how well the deliverable performs. Verification seeks to determine whether the software product is being built correctly whereas validation seeks to determine whether the correct product has been produced from an assumed set of correct specifications.

Execution-based and nonexecution-based testing are distinguished (Goodenough,1979; Schach,1996). Execution-based testing is used for testing executable code. Nonexecution-based testing is used to review deliverables, e.g. an Analysis Document as well as executable code.

Goodenough (1979) defines execution-based testing as a process of inferring certain behavioural properties of executable code, based partially on the results of executing the code in a known

environment with selected inputs. Behavioural properties of code that must be verified includes utility, reliability, robustness, performance and correctness (Schach,1993).

**Utility** is the extent to which a user's needs are met when a correct product is used under conditions permitted by its specifications. **Reliability** is a measure of the frequency and criticality of product failure. **Robustness** is a function of a number of factors including the range of operating conditions, the possibility of unacceptable results with valid input and the acceptability of effects when the product is given invalid input. **Performance** refers to the extent to which the product meets its constraints with regard to response time or space requirements. A deliverable is **correct** if it satisfies its output specifications, independent of its use of computing resources, when operated under permitted conditions (Goodenough,1979).

According to Sage and Palmer (1990), SQA and associated testing can be conducted from either a structural, functional or purposeful perspective. From a structural perspective, software would be tested in terms of micro-level details such as programming language style, control and coding particulars. From a functional perspective, SQA and testing involves treating the software as a blackbox and determining whether the software performance conforms to the software technical requirements specification. From a purposeful perspective software must be tested to determine whether it does what the user really wishes it to do.

Inspections (Fagan,1976; Ackerman,1989) and walkthroughs (Dunn,1984; Schach,1996) are two types of nonexecution-based techniques. The inspection team should include a representative of the team responsible for the current cycle, a representative of the team responsible for the next cycle, a member of the SQA group (tester) as well as a moderator who is the manager and leader of the inspection team. An inspection involves five formal steps:

1. an overview of the document is given by an individual responsible for producing the involved deliverable;
2. the document is distributed to the participants for detailed preparation and to compile checklists of unclear items and possible faults;

3. a participant walks through the document with the inspection team while the participants present their checklists and fault finding commences;
4. the moderator compiles a report of the inspection which is used by the individual responsible for the document to rework and solve all faults and problems noted in the report;
5. the moderator must ensure that all issues raised are followed up and are satisfactorily resolved without introducing new faults.

The group involved in a walkthrough should consist of four to six individuals which include the manager of the group responsible for producing the deliverables, one representative from the group responsible for producing the deliverables, a user representative, a representative of the team who will perform the next cycle of integration as well as a member of the SQA group who will chair the walkthrough.

Once a time and a venue for the walkthrough have been arranged, the material for the walkthrough are distributed to the participants in order to allow for preparation. Each participant should study the material and identify the aspects that he does not understand as well as the aspects that he believes to be incorrect. It is not the task of the walkthrough team to correct faults, but merely to record them for later correction.

The walkthrough can either be participant-driven or document-driven. When participant-driven, participants present their lists of unclear and possibly incorrect items while a person responsible for the deliverables responds to each query, i.e. clarifying unclear aspects as well as either agree that a fault exists or explain why it does not exist. When document-driven, a person responsible for the deliverables, walks the participants through the deliverables with the reviewers interrupting either with their prepared comments or with comments triggered by the presentation. A document-driven walkthrough is likely to be more thorough and is the technique prescribed in the IEEE standard for software reviews (IEEE 1028,1988). Walkthroughs are less formal than inspections and are mainly used as a review technique when integrating legacy systems with the client/server environment.

Boehm (1984) suggests an evaluation criteria for performing V & V which comprises completeness, consistency, feasibility and testability. **Completeness** demonstrates that all product components are present, i.e. no non-existent references, missing functions or missing deliverables. **Consistency** implies that deliverables are traceable, i.e. that no cycle deliverable has conflicting interpretations with the previous cycle deliverable. **Feasibility** implies that the deliverable will save more than it costs to build, irrespective of the cost criteria. **Testability** implies the ability to find an economical method to test the products which will exhibit whether or not they meet the specifications. With the ESMI the issue of software reuse is added to this evaluation criteria of V & V. Reusability is the extent to which a deliverable or part of a deliverable can be reused. Software reuse is discussed in Section 5.3.5.

During the first quadrant of all cycles of the ESMI, the SQA criteria, standards and metrics for the involved cycle are formulated. During the second quadrant testing tools and other SQA techniques are analysed for use in the measurement of the quality criteria for the involved cycle. V & V activities performed during regular reviews in the fourth quadrant of every cycle of the ESMI include:-

- During the Feasibility Cycle the Problem Statement, Legacy System Description Document, Feasibility Report and Preliminary Project Proposal are evaluated.

- During the Architecture Cycle the Legacy Infrastructure Description Document, Architecture Strategy Evaluation Report, Architecture Document SPMP, PRMP as well as the SQAP are evaluated.

- During the Analysis Cycle the Legacy Subsystem Description Document, Subsystem Strategy Evaluation Report as well as the Analysis Document which include the Object, Dynamic and Functional Models, are evaluated.

- During the Design Cycle the System Design Document, Detailed Design Document, Source and Object Code Listings and Test Reports are evaluated and completed subsystems are tested.

- Review techniques used during the fourth quadrant of the Implementation Cycle are known as validation techniques. These techniques involve the testing of the integrated legacy system as a whole to ensure that the specifications and user requirements are met. Program testing is used as a validation technique. It involves the running of a program, the inspection of its input and output as well as the observation of unexpected results. The Integration Strategy Evaluation Report, Training Schedules and Programmes, User Manual, System Maintenance Plan as well as the Acceptance Test Reports are evaluated.

### 5.3.5 Software Reuse

Software reuse refers to an approach to software development in which software is not developed from scratch, but software components of other products are used to facilitate the development of a different product with different functionality. Software is reused when it is used as part of software other than that for which it was initially designed. Software reuse encompasses source code, object code, requirement specifications, design specifications, physical designs, test and development plans and automated documentation (Horowitz & Munson,1984; Jones,1984).

Schach (1996) distinguishes between accidental reuse and deliberate reuse. Accidental reuse happens if the developers of a new product realise that a component of a previously developed product can be reused in the new product. Deliberate reuse imply the utilisation of software components constructed specifically for the purpose of possible reuse in future products. The advantage of deliberate reuse is that software components specially constructed for use in future products are more likely to be well-documented and thoroughly tested.

Wegner (1984) categorise reusable elements into: those elements that can be reused in a number of applications (e.g. a math function); those that are used in successive versions (e.g. a new version of an application based on a previous version); those that are reused whenever the program containing the element is executed (e.g. a compiler); and those that are reused in a program (e.g. a subroutine).

A high degree of reuse results in less code to be written for each new application and consequently there is less code to maintain and the development process is shortened. In order to attain

reusability, modules should be designed to have high cohesion and low coupling (Schach,1996). Cohesion refers to the degree of interaction within a module and coupling refers to the degree of interaction between different modules. For the highest degree of reusability, modules should have functional cohesion and data coupling. A module that performs exactly one action or achieves a single goal has functional cohesion. Two modules are data-coupled if all parameters are homogeneous data items.

The main goal of OO development is to achieve reusability (Rumbaugh,1991). Reusability is enhanced by means of the OO principles, i.e. data encapsulation, classification, inheritance, polymorphism, data abstraction and modularity. Class libraries consisting of reusable classes can assist users of OO languages. Development environments which support class libraries such as the environment for the language Eiffel which incorporates seven libraries of classes (Meyer,1990), are available in the IS/T industry. Reusability will be implemented in the ESMI by making use of these reusable component libraries, as well as by adding the issue of reuse to the quality evaluation criteria (Section 5.3.4). Motivating and encouraging technical members of the integration team to apply reuse whenever possible are considered project management tasks (Section 4.4).

Prerequisites for the development of reusable components include standards for the development of the component, quality metrics for components and evaluation criteria for acceptability of components for a component library.

## 5.4 Procedures for the ESMI

Procedures for the modelling of the system to be integrated, as well as for cost estimation are now discussed.

### 5.4.1 Modelling

During OOA the system to be integrated is modelled in terms of Object, Dynamic and Functional Models. A summary of the different steps for constructing these models as presented by Rumbaugh et al (1991), follows.

Building an Object Model comprises:

1. the identification of object classes;
2. the creation of a data dictionary which contains descriptions of classes, attributes and associations;
3. the addition of associations between classes;
4. the addition of attributes for objects and links;
5. the use of inheritance to organise and simplify object classes;
6. the testing of access paths using scenarios and the iteration of the above-mentioned steps as needed, as well as
7. the grouping of classes into modules.

Constructing a Dynamic Model involves:

1. the preparation of scenarios of typical interaction sequences;
2. the identification of events between objects and the preparation of an event trace for every scenario;
3. the preparation of an event flow diagram for the system;
4. the development of a state diagram for each class which has important dynamic behaviour, and
5. the verification for consistency and completeness of events shared among the state diagrams.

Developing a Functional Model comprises:

1. the identification of input and output values;
2. the use of DFD's to show functional dependencies;
3. the description of each function;
4. the identification of constraints, and
5. the specification of the optimisation criteria.

### 5.4.2 Cost Estimation

Cost estimation forms an integral part of a cost-benefit analysis (Section 5.3.1). A procedure for software cost estimation comprises the following steps (Boehm,1984; Lederer & Prasad,1992):-

1. Establish objectives of the cost estimate. These objectives should determine the level of detail and effort required to perform the subsequent steps.
2. Plan for required data and resources. Compile a software cost estimating miniproject plan which includes the purpose of the estimation, the products and schedules of the estimation, the responsibilities for each product, the procedures and cost estimating tools and techniques to be used, the required resources (e.g. data, time, money) needed to complete the estimate as well as the assumptions (e.g. availability of key personnel) under which the above estimates are to be delivered, given the above resources.
3. Ensure software requirements are specific, unambiguous and quantitative wherever possible (i.e. with respect to estimating objectives). This will allow for a more accurate cost estimation.
4. Determine as much technical detail as is consistent with cost estimating objectives. The more detail explored, the better the understanding of the technical aspects of the software and the more accurate the estimates will be.
5. Make use of a combination of independent techniques and sources to avoid the weaknesses of any single method and to utilise their joint strengths.
6. Compare and iterate estimates, i.e. determine reasons for different estimate values with different cost-estimation techniques by identifying the components of the cost in each.
7. Regularly review the estimates once the integration project has started to compare these estimates to actual costs and progress. A useful technique is the cost-schedule-milestone chart which involves the graphical representation of both the estimated number of months required to achieve major project milestones as well as the actual cost and schedule associated with the achievement of these milestones.

In the case of a significant difference between the estimates and the actuals, an investigation can be initiated and corrective action taken.

## 5.5 Automated Support for the ESMI

The term Computer-aided Software Engineering (CASE) in the context of integration refers to the ability of computers to assist software engineers in every step of the integration project by helping to carry out much of the drudge work associated with an integration project.

The simplest form of CASE is the software tool which assists in only one aspect of the integration project. A variety of such tools are available in the market, e.g. report generators and screen generators to assist in the construction of a rapid prototype.

A CASE workbench is the term used for a collection of tools supporting one or two tasks (each comprising a collection of activities) of the integration project. Commercially available workbenches include Analyst / Designer and Excelerator.

The next item in the progression of CASE is the Software Engineering Environment (SEE) which supports either the complete software process or a large portion of it. Charette (1986) defines a SEE as being

***"The process, methods, and automation required to produce a software system."***

The basic function of the **process** for integrating a legacy system with a client/server environment, is to describe the sequence of events required to integrate a specific legacy system. The **methods** include all those required to define, describe, abstract, modify and document the integrated legacy system and are defined by the process. **Automation** involves the use of the computer to implement the necessary methods. According to Charette (1986) an "ideal" SEE consists of a complete process model which is fully supported by methods which are in turn fully automated.

Language-centered, structure-oriented and toolkit environments are distinguished (Dart et al,1987; Fugetta,1993).

Language-centered environments are built around a single programming language and are generally designed for rapid development. Examples include the Smalltalk environment (Goldberg,1984) for the Smalltalk language. Disadvantages of these environments are that they usually only provide support for the implementation phase of a SDLC, that there is a lack of support for programming-in-the-large and that there is usually no support for project management. A major strength is their support for rapid prototyping.

Structure-oriented environments are constructed around a structure editor and are tied to the specific programming language whose structure is built into the editor. Examples include the FLOW environment (Doodley & Schach,1985). Disadvantages of these environments are that they usually only provide support for the implementation phase of a SDLC and that there is a lack of support for programming-in-the large. These disadvantages are, however, not applicable to the FLOW environment.

Toolkit environments consist of a collection of tools combined to provide support to a variety of activities. These environments are not limited to a specific programming language. Examples include the Unix Programmer's Workbench (Dolotta et al,1984). A disadvantage of toolkit environments is that the tools are usually not integrated. A strength is the ability to add any required tool to the environment.

Charette (1986) is of the opinion that a large software system such as the legacy systems identified, cannot be built without a SEE. The use of a SEE during the integration project is aimed at integrating the integration process with the techniques used to integrate legacy systems with client/server environments and automating the result. It will reduce variation in practice, increase productivity as well as reduce costs. According to Charette (1986) information that is transparent to all methods, automatically produced documentation and a database with good performance are the minimum requirements a SEE should have.

The word *integrated* within the context of an environment may comprise the integration of the user-interfaces, processes, tools, teams as well as management (Schach,1996). User-interface integration results in a common user-interface shared by all tools in the environment. Process integration refers to an environment which supports a specific software process. Tool integration refers to the ability of all tools to communicate via the same data format. Team integration promotes effective team coordination and communication within an environment. Management integration refers to an environment which provides support to the management of the software process, e.g. reports with management information are generated directly from a software project database.

An environment which provides integrated computer support for an OO methodology is termed an OO Information Systems Engineering Environment (OOISEE), also referred to as an OO Computer-aided Software Engineering (OO-CASE) environment. OMTool is a process integrated environment for the Object Modeling Technique (OMT) of Rumbaugh et al (1991)[1] which is a good candidate for use with the ESMI. The Rose environment supports the methodology of Booch (1994). Other integrated development environments supporting OO are emerging, e.g. the Object Management Workbench (OMW) of Intellicorp. These OO environments include a repository, OO-CASE tools as well as a powerful code generator. The term OO-CASE is, however, often used when referring to OO software tools which support only some of the methods and techniques of an OO methodology.

OOISEE tools should provide support for the entire methodology, all project participants, methods and techniques, integration of tools, reuse, prototyping as well as SQA. Inclusion of computer-aided simulation techniques are important for the verification of the feasibility and evaluation of the performance of the target system. So is an automated prototyping tool to be used during the second quadrant of the ESMI in order to complement the functions of a simulation facility and to assist in evaluating whether the requirements specification conforms to the user needs. Advanced tools are used to build a quick version of all or part of the target system. The speed and flexibility of prototypes enable users to experience the application (in particular, the user-interface) early and

[1] The methods of Rumbaugh et al (1991) for OOA and OOD are collectively known as the OMT.

allow changes to be made based on their feedback. Prototyping is discussed in Section 5.3.3. Lantz (1985) identifies the following tools to assist in prototyping:

- a repository to record and organise information about data (repositories are discussed in Section 5.5.1);
- an interactive testing system which allows the use of a terminal to change programs quickly, the submission of frequent tests and the review of test session results without having to wait for batch outputs;
- a test data generator for producing test data to exercise the prototype;
- library control of program modules to aid in making quick changes to the prototype and in moving data between system components by providing version control of programs and database definitions;
- a report writer, screen painter and query language for the quick modification of the prototype.

An OOISEE should include an integrated set of software tools to store requirements specification and design information, and to produce and maintain documentation. Tools should also be available to assist in the verification of the consistency and completeness of the specification through static and dynamic analysis of the specifications already contained in the centralised database. Analysis for completeness, redundancy and consistency of software requirements and design specifications normally requires that the specifications be formulated in a computer-processable form, such as formal specification language statements.

OOISEE tools should perform integrity and consistency checks in order to ensure a fully validated design, e.g. when an object diagram is used to show a scenario with a message being passed from one object to another, a tool can ensure that the message is part of the object's protocol. Constraint checking should be done by tools, e.g. a tool should enforce conventions such as "there are no more than three instances of this class". A tool should indicate if certain classes or methods of a given class are never used, i.e. completeness checking should be done.

An ideal OOISEE for the ESMI should enforce the procedures and standards of this software process model and facilitate its application. It should provide automated support for the techniques, tools and procedures of the ESMI, and make this as well as other software utilities available to the user at an appropriate workstation. It should be convenient to use, support customisation, have an open architecture and have a comprehensive conceptual schema that encompasses the database, process data, tool interfacing and environment evolution (Humphrey,1989).

In addition to the above, it is also most important that the environment should support project management. As discussed in Section 4.4, the tasks for managing projects fall into four categories and OOISEE tools should provide support to all these categories of managing tasks:

1. tools to support project organisation tasks include organisational and other communication devices which link the project team's work to the users at both the managerial and lower levels, e.g. the creation of a user steering committee and progress reports prepared for corporate steering committee;
2. tools which support project directing ensure that the team operates as an integrated unit, e.g. the preparation and distribution of minutes within the integration team on key design evolution decisions;
3. formal planning tools for structuring the sequence of tasks in advance and estimating the time, money and technical resources needed to execute these tasks, e.g. Program Evaluation and Review Technique (PERT);
4. formal control tools for assisting in the evaluation of progress and the location of potential discrepancies so that corrective action can be taken as well as a configuration management tool;
5. tools to assist in the writing and updating of the various reports of the ESMI.

Nonprocedural languages, including Structured Query Language (SQL) and report generators should be integrated into this OOISEE environment. Most important in the environment is the ability to generate OO code directly from the design tool. The OOISEE tools for planning, data and process modelling and for creating designs must be integrated with code generators. The

OOISEE tools should support the ESMI and should be used to draw explicit, detailed diagrams and schematics from which code can be generated.

Booch (1994) identifies seven different kinds of tools that are applicable to OO development:

1. a graphics-based tool supporting the OO notation to be used during analysis to capture the semantics of scenarios, as well as early during development to capture strategic and tactical design decisions, maintain control over the design products and coordinate the design activities of a team of developers;
2. a sophisticated browser that embodies the class structure and module architecture of a system;
3. an incremental compiler which can compile single declarations and statements to assist with debugging;
4. a debugger that embodies class and object semantics, stress testers which stress the capacity of the software in terms of resource utilisation as well as memory-analysts which identify violations of memory access (such as writing to deallocated memory or reading and writing beyond the boundaries of an array);
5. configuration management and version-control tools;
6. a class librarian tool that allows developers to locate classes and modules in the class library according to different criteria and add useful classes and modules to the library as they are developed;
7. a GUI builder to assist in interactively creating dialogs and other windows.

When building an OOISEE for integrating legacy systems with the client/server environment, the emphasis should be on the deliverables required, rather than on the methods and automation of the environment (Charette,1986). A central repository of information about the target system is fundamental to an OOISEE, and should be maintained during each cycle of the integration project. A repository contains metadata (i.e. data about data) regarding systems. Repositories are now discussed.

### 5.5.1 The Repository

McFadden and Hoffer (1991) define a repository as:

> ***" a centralised knowledge base that contains all data definitions, screen and report formats, and definitions of other organisational and system components."***

A repository stores all the information about systems, designs and code in a basically nonredundant fashion for use by all developers. Descriptions regarding classes, associations, attributes and operations of systems, as well as the meaning represented in OO-CASE diagrams are included in the repository. According to Martin (1992):

> ***"The repository is a mechanism for defining, storing and managing information about an organization, its data and systems."***

The developers employ the information in the repository and create new information that, in turn, is placed in the repository. The developers create designs with the help of information from the repository.

Maintaining a repository helps to establish a common and consistent vocabulary that can be used throughout an integration project. A repository can serve as an efficient vehicle for browsing through all the elements of a software project in arbitrary ways. This feature is particularly useful as new members are added to the integration team, who must quickly orientate themselves to the solution already under development. A repository permits architects to take a global view of a software project, which may lead to the discovery of commonalities that otherwise might be missed.

## 5.6 Summary

The most relevant methods, techniques and procedures for the ESMI were discussed. OOA is an OO method for analysis which directly maps the problem domain into a coherent set of models

which constitute a convincing description of the problem domain and from which a software system can be developed. OOD is an OO method for design which implies the refinement of the analysis models in order to provide a detailed basis for implementation. OOP is a method for implementation which implies the conversion of the design into an executable software system. Rumbaugh et al's (1991) OMT was recommended for use with the ESMI.

Techniques used during a cost-benefit analysis are classified into Algorithmic Models, Expert Judgement, Estimation by Analogy, Parkinsonian Estimation, Price-to-Win Estimation, Top-down Estimation as well as Bottom-up Estimation. A combination of these techniques should be used for a single cost-benefit analysis (second quadrant) in order to ensure accurate cost estimations.

Risk management techniques reduce the chance of risk items becoming either threats to the successful integration of a legacy system or to result in rework during an integration project. Risk Management comprises Risk Assessment and Risk Control. Risk Assessment involves Risk Identification, Risk Analysis and Risk Prioritisation. Risk Control involves Risk Management Planning, Risk Resolution and Risk Monitoring. Risk assessment is a task of the second quadrant of all cycles of the ESMI whereas risk control is a management task performed throughout the integration project.

Rapid prototyping allows for the rapid creation of a working model of part of the system to be integrated. It is a technique useful for the gathering of requirements as well as for reducing risk by prototyping risk areas during the second quadrant of all cycles of the ESMI.

SQA comprises all management processes, analysis, design and programming techniques as well as tools which ensure that the resulting integrated system meets or exceeds a predetermined set of standards. The testing of executable code as well as walkthroughs and inspections during regular reviews are techniques used to examine deliverables of the integration life-cycle to ensure that requirements are met and to detect and diagnose faults. The SQA group is primarily concerned with ensuring an integrated system of high quality.

Software reuse allows for the reuse of software components in different products with different functionality than the product for which they were initially designed. Reusable components include object code, source code, requirement specifications, physical designs as well as documentation. The advantages of software reuse include a shorter integration process due to less components to be developed from scratch as well as increased reliability because components reused in different applications will have less faults. Reusability will be implemented in the ESMI by making use of reusable component libraries, as well as by adding the issue of reuse to the quality evaluation criteria (Section 5.3.4). Motivating and encouraging technical members of the integration team to apply reuse whenever possible are considered project management tasks (Section 4.4).

Procedures for the modelling of the system to be integrated as well as for cost estimation (an integral part of a cost-benefit analysis) were reviewed after which automated support for the ESMI was discussed. As recommended in Section 3.5, the use of a SEE in an organisation should be postponed until the organisation has reached CMM level 3. For CMM level 3 organisations, an integrated environment which supports the methods of the ESMI should be used. OMTool which supports the Object Modeling Technique of Rumbaugh et al (1991) is a good candidate for this type of organisations. Additional tools will be needed for the activities of the ESMI not supported by OMTool, e.g. management tools to assist in planning and scheduling. If the use of a SEE is not feasible, the next best alternative is a CASE workbench (a collection of appropriate tools) that provides support to some of the activities of the ESMI. The workbench should at least include the following:

- a tool that supports the graphical aspects of OOA;
- tools such as report and screen generators to speed up rapid prototyping;
- a tool built around a repository, to ensure that every record in the repository occurs in the design and that all aspects of the Analysis Model are incorporated in the design;
- coding tools for OOP which simplify the programmer's task and increase productivity such a text editors and debuggers;
- version control and configuration management tools to ensure that the appropriate version of each module is compiled and linked after faults have been corrected during the Implementation Cycle, as well as

- management information tools such as a scheduling tool to keep track of the assignment of tasks to project members, a tool which generates PERT charts, tools to assist with planning as well as to monitor the development process as a whole.

It is recommended that the utilisation of the ESMI for integrating a legacy system with the client/server environment within an organisation, is postponed until the organisation involved has reached CMM level 3. An instance of the conceptual model of Chapter 3 (Figure 3.2) for such an organisation is illustrated in Figure 5.3. As illustrated in this figure, the techniques of the ESMI are based on the OO paradigm. These techniques are used by Rumbaugh et al's (1991) OMT, which is, in turn, implemented by the procedures of the ESMI. The integration methodology is based on the ESMI and the OO paradigm is visualised by means of the Object Modeling Notation (Rumbaugh et al,1991) discussed in Section 4.3.3. The OMTool environment provides for computerised support for the integration methodology. The deliverables of the ESMI (Section 4.3.7) are produced as a result of the integration process.

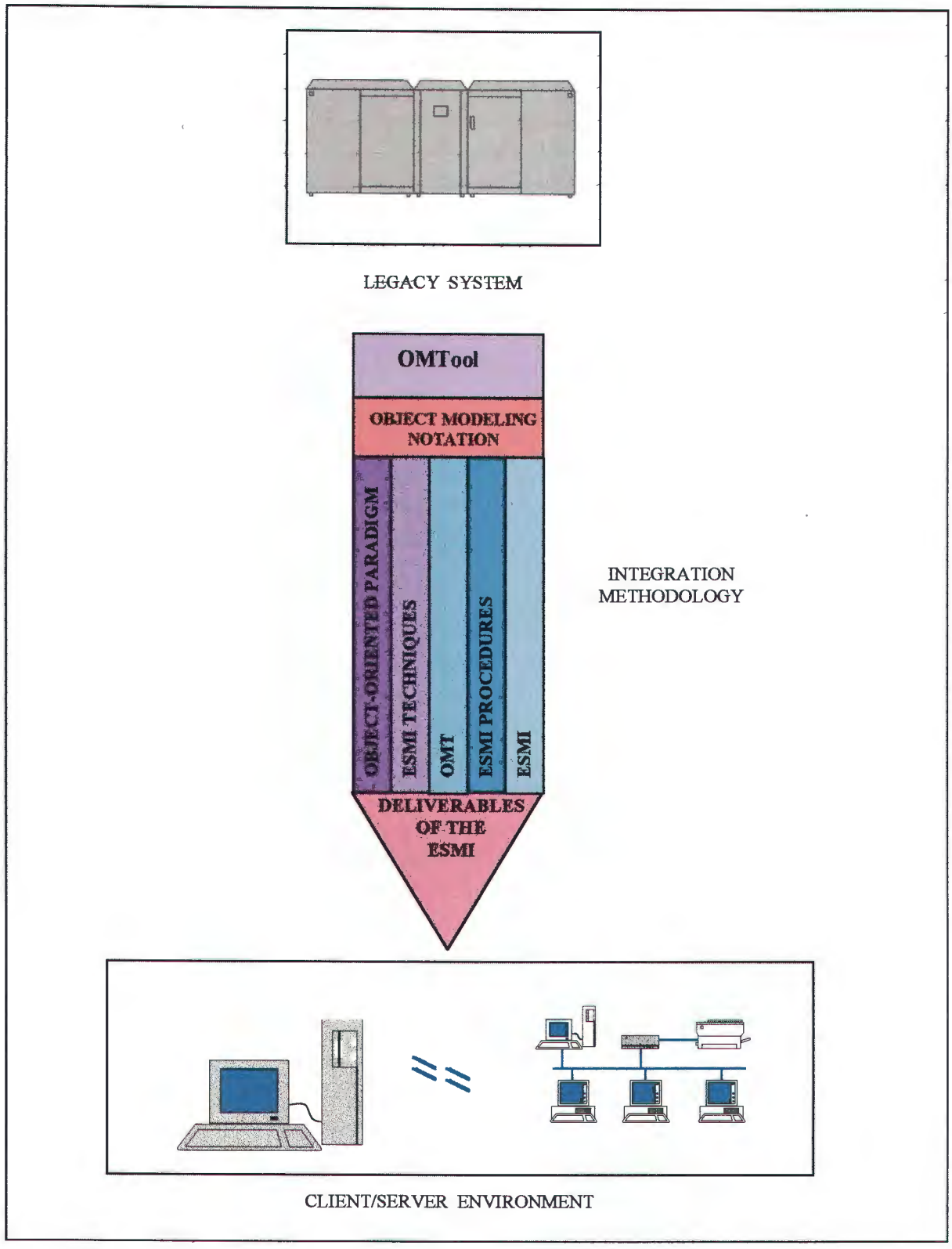


**Figure 5.3** An Instance of the Conceptual Model

# CHAPTER 6

# SUMMARY AND CONCLUSIONS

## 6.1 Introduction

In this chapter, the investigation is reviewed in terms of the objectives of the study as envisaged in Chapter 1. A summary of the work is presented after which the research results are summarised and conclusions are stated. Proposed areas for further investigation are given in the last section of the chapter.

## 6.2 Summary of Investigation

The investigation was based on the hypotheses that a methodology as well as a software process model on which the methodology is to be based for integrating legacy systems with the client/server environment, can be developed. A method of investigation was established to guide the investigation in terms of the objectives, assumptions and constraints as outlined in Chapter 1.

A literature survey concerning legacy systems, client/server environments and trends for integrating legacy systems with the client/server environment in the IS/T industry, resulted in the compilation of a questionnaire for identifying the characteristic properties of legacy systems. After a preliminary screening of all legacy systems within SASOL, five legacy systems in various application domains were identified. The questionnaire was used for identifying the characteristic properties of these systems.

A second literature survey regarding established methodologies, their features and technical characteristics as well as the various perspectives taken by them, was conducted. This information was used, along with the most dominant characteristics yielded by the questionnaire, to derive a synthesis for an integration methodology. The investigation culminated in the proposal of a software process model on which the integration methodology will be based. Due to the limited scope of this dissertation, the integration methodology was not prescribed in full detail.

## 6.3 Summary of Results and Conclusions

Legacy systems represent large investments in application software, staff skills, system software and hardware in organisations. Major problems of these systems include the high maintenance costs of mainframe environments, inaccessible data as well as the lack of GUI's. The primary focus of new software development is on client/server computing which provides for more cost-effective platforms, excellent GUI's as well as reduced network traffic. A need to integrate legacy systems with the client/server environment therefore exists within organisations. Different strategies to accomplish this integration exist and decisions regarding a strategy for integrating a specific legacy system with the client/server environment are primarily influenced by the characteristics of the involved legacy system. A methodology is essential in order to manage and control the integration process.

Despite the above-mentioned problems of mainframe environments, they are known for their stability, maturity and secure nature. Strengths of these environments include batch processing and automated job scheduling. Weaknesses of client/server environments include the lack of both mature client/server techniques and tools as well as experienced professionals. Poor candidates for a total client/server architecture are:

- very large or complex systems;
- systems with high-volume centralised I/O processing;
- systems that require centralised control and security;
- systems which are tightly integrated with other legacy systems.

The questionnaire yielded the most dominant characteristics of the five legacy systems of SASOL:

- a relatively high number of daily transactions;
- tight integration of legacy systems with each other;
- large capital investments;
- high annual costs;
- high quality;

- critical availability and system performance;
- providing excellent business support;
- medium complexity.

As a result of the tight integration of these systems with each other, they are categorised as poor candidates for a total client/server architecture. A discussion of this matter is outside the scope of this investigation and it is sufficient to say that an integration project involving one of these systems will have high inherent risk.

A methodology is an overall systematic approach to the production of software with basic concepts being a paradigm, representation schema, methods, techniques, procedures as well as deliverables. Six classes of methodologies were distinguished according to the various perspectives taken by established methodologies. They are:

- process-orientation;
- data-orientation;
- behaviour-orientation;
- cross-references between perspectives;
- meta;
- object-orientation.

Legacy systems were usually developed by making use of either a process-oriented methodology or no methodology at all. The use of no methodology explains the presence of corrupt data and the difficulty with which maintenance is usually performed on these systems.

The above-mentioned characteristic properties of the legacy systems determined the features and technical requirements of an integration methodology. They are:-

- The methodology should be based on the OO paradigm. OO software development requires iteration among life-cycle phases, a prototyping strategy as well as the incremental building of a product.
- The methodology should provide for a risk-driven approach.

The revised spiral model for OO development (Du Plessis & Van der Walt,1992; Van der Walt,1993) conforms to these requirements and was customised to derive the Enhanced Spiral Model for Integration (ESMI). The most relevant methods, techniques and procedures of the ESMI were identified and discussed. It is recommended that the utilisation of the ESMI for integrating a legacy system with a client/server environment within an organisation, is postponed until the involved organisation has reached Capability Maturity Model (CMM) level 3. It remains to be seen if the ESMI, on which the integration methodology will be based, can be successfully applied in practice to integrate a legacy system with the client/server environment.

## 6.4 Areas for Further Investigation

The following areas for further investigation were identified:

- the identification of the characteristic properties of more legacy systems by means of structured interviews rather than questionnaires;
- the detailed definition of the deliverables of the ESMI (Section 4.3.7);
- prescription of the detail Work Breakdown Structure (WBS) of the integration methodology;
- empirical studies applying the integration methodology and the ESMI;
- the development of a Software Engineering Environment (SEE) to support the integration of legacy systems with the client/server environment.

# LITERATURE REFERENCES

Ackerman A. F., Buchwald L. S. & Lewski F. H., "Software Inspections: An Effective Verification Process," *IEEE Software*, Vol. 6, May 1989, pp. 31-36.

Alavi M., "An Assessment of the Prototyping Approach to Information Systems Development," *Communications of the ACM*, June 1984, pp. 556-563.

Balda D. M. & Gustafson D. A., "Cost Estimation Models for the Reuse and Prototype Software Development Life Cycles," *ACM SIGSOFT Software Engineering Notes*, Vol. 15, July 1990, pp. 42-50.

Balzer R., Cheatham T.E. & Green C., "Software Technology in the 1990s: Using a New Paradigm," *Computer*, November 1983, pp. 39-45.

Beck K. & Cunningham W., "A Laboratory for Teaching Object-Oriented Thinking," *SIGPLAN Notices*, Vol. 24(10), October 1989.

Benington H.D., "Production of Large Computer Programs," *Proceedings of ONR Symposium Advanced Programming Methods for Digital Computers*, June 1956, pp. 15-27.

Berard E.V., *Essays on Object-Oriented Software Engineering*, Volume I, Prentice-Hall, 1993.

Boehm B. W., Bosch C. A., Liddle A. S. & Wolverton R. W., "The Impact of New Technologies on Software Configuration Management," *TRW Report to USAF-ESD*, June 1974.

Boehm B. W., "Software Engineering," *IEEE Transactions on Computers*, December 1976, pp. 1226-1241.

Boehm B. W., "Software Engineering Economics," *IEEE Transactions on Software Engineering*, Vol. SE-10, January 1984, pp. 4-21.

Boehm B. W., "Verifying and Validating Software Requirements and Design Specifications," *IEEE Computer*, Vol. 17 No. 1, January 1984.

Boehm B. W., "A Spiral Model for Software Development and Enhancement," *ACM Sigsoft Software Engineering Notes*, Vol. 11 No. 4, 1986.

Boehm B. W., *Tutorial : Software Risk Management*, IEEE Computer Society Press, 1989.

Booch G., "Object-Oriented Development," *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 2, 1986, pp. 211-221.

Booch G., *Object-Orientation Analysis and Design*, The Benjamin / Cummings Publishing Company Inc, 1994.

Capper N. P., Colgate R. J., Hunter J. C. & James M. F., "The Impact of Object-Oriented Technology on Software Quality: Three Case Histories," *IBM Systems Journal* 33, No. 1, 1994, pp. 131-157.

Cardelli L. & Wegner P., "On Understanding Types, Data Abstraction and Polymorphism," *ACM Computing Surveys*, Vol. 17(4), 1985, p. 481.

Carey T. T. & Mason R. E. A., "Information System Prototyping: Techniques, Tools and Methodologies," *The Canadian Journal of Operational Research and Information Processing*, August 1983, pp. 177-191.

Charette R. N., *Software Engineering Environments - Concepts and Technology*, McGraw-Hill, 1986.

Charette R. N., *Software Engineering Risk Analysis and Management*, McGraw-Hill, 1989.

Charette R. N., "Risk Management to Maximise Commercial Opportunities," *Proceedings of the Software Tools Conference*, Blenheim Online England, 1991.

Coad P. & Yourdon E., *Object-Oriented Analysis*, Prentice-Hall, 1991.

Conger S., *The New Software Engineering*, International Thomson Publishing, 1994.

Connell J. L. & Shafer L., *Structured Rapid Prototyping*, Prentice-Hall, 1989.

Cox B. J., *Object-Oriented Programming - An Evolutionary Approach*, Addison-Wesley, 1986.

Currit P. A., Dyer M. & Mills H. D., "Certifying the Reliability of Software," *IEEE Transactions on Software Engineering*, January 1986, pp. 3-11.

Dart S. A., Ellison R. J., Feiler P. H. & Habermann A. N., "Software Development Environments," *IEEE Computer*, November 1987, pp. 18-28.

Davis G., Block C., Kang K. C., Chikofsky E. & Teichroew D., "Usage of the System Encyclopedia Manager (SEM) System with the System Analysis and Design Language for Ada (SALA)," *IFIP TC-2 Conference on System Description Methodologies*, Hungary, 1983, pp. 157-207.

Department of Defence, "Military Standard - Defence System Software Development - DOD-STD-2176A," Department of Defence, Washington, 1988.

Dolotta T. A., Haight R. C. & Mashey J. R., *UNIX Time-sharing System: The Programmer's Workbench in : Interactive Programming Environments*, Barstow D. R., Shrobe H. E. & Sandewall E. (Editors), McGraw-Hill, 1984, pp. 353-369.

Doodley J. W. M. & Schach S. R., "FLOW: A Software Development Environment Using Diagrams," *Journal of Systems and Software*, August 1985, pp. 203-219.

Dunn R. H., *Software Defect Removal*, McGraw-Hill, 1984.

Du Plessis A. L., Bornman C. H. & Teichroew D., "ELSIM SEE: A Software Engineering Environment for Real-Time Systems," *Proceedings of ISETT Conference,* Italy, May 1986.

Du Plessis A. L. & Van der Walt E., "Modeling the Software Development Process," *Conference Proceedings on Information Systems Concepts: Improving the Understanding*, Alexandria, Egypt, April 1992.

Fagan M. E., "Design and Code Inspections to Reduce Errors in Program Development," *IBM Systems Journal*, No. 3, 1976, pp. 182-211.

Farquhar J. A., "A Preliminary Inquiry into the Software Estimation Process," *The Rand Corporation*, August 1970.

Forge S., *Developing Cooperative and Client/server Systems*, McGraw-Hill, 1995.

Fuggetta A., "A Classification of CASE Technology," *IEEE Computer*, December 1993, pp. 25-38.

Gane C. & Sarson T., *Structured Systems Analysis: Tools and Techniques*, Prentice-Hall, 1979.

Gilb T., *Principles of Software Engineering Management*, Addison-Wesley, 1988.

Goldberg A., *Smalltalk-80: The Interactive Programming Environment*, Addison-Wesley, 1984.

Goldberg R. & Lorin H., *The Economics of Information Processing - Vol. 2*, John Wiley & Sons, 1980.

Gomaa H., "Software Development of Real-Time Systems," *Communications of the ACM*, July 1986, pp. 657-668.

Goodenough J. B., "A Survey of Program Testing Issues, Research Directions in Software Technology," *The MIT Press*, 1979, pp. 316-340.

Grosvenor J. B. M., *Mainframe Downsizing to Upsize your Business*, Prentice-Hall, 1994.

Hatley D. J. & Pirbhai I. A., *Strategies for Real-Time System Specification*, Dorset House, 1987.

Henderson-Sellers B. & Edwards J.M., "The Object-Oriented Systems Life-Cycle," *Communications of the ACM*, September 1990, pp. 142-159.

Horowitz E. & Munson J., "An Expansive View of Reusable Software," *IEEE Transactions on Software Engineering*, Vol. SE-10, No. 5, September 1984.

Humphrey W. S., *Managing the Software Process*, Addison-Wesley, 1989.

Institute of Electrical and Electronic Engineers, "Standard for Software Quality Assurance Plans," IEEE/ANSI Std. 730-1984, 1984.

Institute of Electrical and Electronic Engineers, "Standard for Software Project Management Plans," IEEE 1058.1, 1987.

Institute of Electrical and Electronic Engineers, "Standard for Software Reviews and Audits," IEEE 1028, 1988.

Iivari J., "Object-Orientation as Structural, Functional and Behavioural Modelling: A Comparison of Six Methods for Object-Oriented Analysis," *Information and Software Technology*, Vol. 37, No. 3, 1995.

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

Jacobson I., Christerson M., Jonsson P. & Overgaard G., *Object-Oriented Software Engineering*, Addison-Wesley, 1992.

Jones T., "Reusability in Programming: A Survey of the State-of-the-Art," *IEEE Transactions on Software Engineering*, Vol. SE-10, No. 5, September 1984.

Kavanagh P., *Downsizing for Client/Server Applications*, Academic Press Inc, 1995.

Keen P. & Gambino T. J., "The mythical man-month revisited," *Proceedings of APL*, 1980, pp. 630-48.

Korson T. & McGregor J. D., "Understanding Object-Oriented: A Unifying Paradigm," *Communications of the ACM*, September 1990, Vol. 33, No. 9.

Lantz K. E., *The Prototyping Methodology*, Prentice- Hall, 1985.

Lederer A. L. & Prasad J., "Nine Management Guidelines for Better Cost Estimating," *Communications of the ACM*, Vol. 35, February 1992, pp. 51-59.

Loftus C. W. et al, *Distributed Software Engineering*, Prentice Hall, 1995.

Marco A., *Software Engineering - Concepts and Management*, Prentice-Hall, 1990.

Marion W., *Client/Server Strategies - Implementations in the IBM Environment*, McGraw-Hill, 1994.

Martin J., *Object-Oriented Analysis and Design*, Prentice Hall, 1992.

McCabe T. J., "Reverse Engineering, Reusability, Redundancy: The Connection," *American Programmer*, October 1990.

McCracken D. D. & Jackson M. A., "Life-Cycle Concept Considered Harmful," *ACM Software Engineering Notes*, April 1982, pp. 29-32.

McFadden F. R. & Hoffer J. A., *Database Management 3rd Edition*, Benjamin/Cummings, 1991.

McFarlan F. W., *Tutorial: Software Risk Management*, IEEE Computer Society Press, 1989.

Meyer B., *Object-Oriented Software Construction*, Prentice-Hall, 1988.

Meyer B., "Lessons from the Design of the Eiffel Libraries," *Communications of the ACM*, Vol. 33, September 1990, pp. 68-88.

Monarchi D. & Puhr G. I., "A Research Typology for Object-Oriented Analysis and Design," *Communications of the ACM*, Vol. 35, No. 9, 1992.

Myers W., "The Need for Software Engineering," *IEEE Computer*, Vol. 11 No. 2, February 1978.

Olle T. W., *Information Systems Methodologies - A Framework for Understanding*, Addison-Wesley, 1988.

Orr K., *Structured Requirements Definition*, Ken Orr and Associates, 1981.

Parkinson G. N., *Parkinson's Law and Other Studies in Administration*, Houghton-Mifflin, 1957.

Paulk M. C., "How ISO 9001 compares with the CMM," *IEEE Software*, Vol. 12, No. 1, 1995.

Peppard J., *I.T. Strategy for Business*, Pitman Publishing, 1993.

Redelinghuys M. & Nienaber R., "Planning and Implementing a Strategy for Open Systems," Special Topic Module, UNISA, 1994.

Roman D. D., *Managing Projects: A Systems Approach*, Elsevier Science, Amsterdam, Nederland, 1986.

Royce W. W., "Managing the Development of Large Software Systems: Concepts and Techniques," *Proceedings of Wescon*, August 1970.

Rumbaugh J., Blaha M., Premerlani W., Eddy F. & Lorensen W., *Object-Oriented Modeling and Design*, Prentice-Hall, 1991.

Sage A. P. & Palmer J. D., *Software Systems Engineering*, Wiley Interscience, New York, 1990.

Selby R. W., Basili V. R. & Baker F. T., "Cleanroom Software Development: An Empirical Evaluation," *IEEE Transactions on Software Engineering*, September 1987, pp. 1027-1037.

Schach S., *Software Engineering 2nd Edition*, Aksen Associates, 1993.

Schach S., *Classical and Object-Oriented Software Engineering 3rd Edition*, Aksen Associates, 1996.

Scroen R. & Meltz M., "Office Systems - Issues and Standards," *Proceedings of the South African Client/server Conference*, 1992.

Shlaer S., Mellor S. J., Ohlsen D. & Hywari W., The Object-Oriented Method for Analysis, *Proceedings of the Tenth Structured Development Forum*, 1988.

Shtub A., Bard J. & Globerson S., *Project Management - Engineering, Technology and Implementation*, 1994.

Simonds D., "Client/server Migration Strategies," *Proceedings of the South African Client/server Conference*, 1992.

Sneed H. M., "Planning the Re-engineering of Legacy Systems," *IEEE Software*, January 1995, pp. 24-34.

Sommerville I., *Software Engineering, 3rd Edition*, Addison-Wesley, 1989.

South African Bureau of Standards, "Code of Practice for Quality Systems - Model for Quality Assurance in Design / Development, Productivity, Installation and Servicing," SABS ISO(9001), The Council of the SABS, 1987.

Stroustrup B., *The C+ Programming Language, 2nd Edition*, Addison-Wesley, 1991.

US Air Force Systems Command, "Software Risk Abatement," AFSC/AFLC pamphlet, pp.800-845, Andrews AFB, 1988.

Van der Walt E., "Software Project Management for Object-Oriented Development," *MSc Dissertation*, University of South Africa, 1993.

Warnier J. D., *Logical Construction of Systems*, Van Nostrand Reinhold, 1981.

Wasserman A. I., Pircher P. A., Shewmake D. T. & Kersten M. L., "Developing Interactive Information Systems with the User Software Engineering Methodology," *IEEE Transactions on Software Engineering*, Vol. SE12, No. 2. February 1986, pp. 326-345.

Watterson K., *Client/server Technology for Managers*, Addison-Wesley, 1995.

Wegner P., "Capital-Intensive Software Technology," *IEEE Software*, Vol. 1, No. 3, July 1984.

Wheeler T, *Open Systems Handbook*, Bantam Books, 1992.

Wessels W., "The New Way of Computing Demystified," *Proceedings of the South African Client/server Conference*, 1992.

Wilkie G., *Object-Oriented Software Engineering*, Addison-Wesley, 1993.

Wirfs-Brock R., Wilkerson B. & Wiener L., *Designing Object-Oriented Software*, Prentice-Hall, 1990.

Wolverton R. W., "The Cost of Developing Large-Scale Software," *IEEE Transactions on Computers*, June 1974, pp. 615-636.

Xephon (Firm), *The Mainframe in Open Systems*, Berkshire England Xephon, 1993.

Yourdon E. & Constantine L. L., *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*, Prentice-Hall, 1979.

# APPENDIX

# A

## The Questionnaire

The purpose of this questionnaire is to determine the characteristics of the legacy systems existing within SASOL. The following legacy systems were identified: the Sastech Financial System, the Sastech Material Management System, the GL System, the PAMM System and the MIMS System.

Please complete one questionnaire, as thoroughly as possible, for each system that you are involved with. . Enter an X in the correct box where appropriate. Questionnaires should be returned to Linda Redelinghuys, IB 4185, Tel. 492628.

***1. PERSONAL DETAILS**

1.1 Name: _____

1.2 Telephone Number: _____

1.3 Are you :

| A User of the System | Responsible for Maintaining the System |
|---|---|

**NOTE:** A user of the system should only complete the sections marked with an * (i.e. Sections 1, 2.1, 6 to 9 and 11 to 13). If you are responsible for maintaining the system, please complete all sections.

**2. LEGACY SYSTEM DETAILS**

*2.1 Name of the System: _____

2.2 Platform: _____

2.3 Operating System: _____

2.4 DBMS: _____

Version: _____ Latest version available: _____

2.5 Application Language: _____

Version: _____ Latest version available: _____

2.6 The software methodology used for development: _____

## 3. SIZE

3.1 Number of Users: _____

3.2 Number of Online Programs: _____

3.3 Number of Batch Programs: _____

3.4 Size of the Database: _____

3.5 Number of Transactions per Day: _____

## 4. GROWTH

4.1 Monthly Growth Percentage of the Database: _____

4.2 Are high volume database updates typical of the system?

| YES | NO |
|---|---|

## 5. INTEGRATION WITH OTHER SYSTEMS

5.1 With which other systems is the relevant system integrated, or to which other systems do interfaces exist?

| | Name | Platform | Operating System |
|---|---|---|---|
| (i) | _____ | _____ | _____ |
| (ii) | _____ | _____ | _____ |
| (iii) | _____ | _____ | _____ |
| (iv) | _____ | _____ | _____ |
| (v) | _____ | _____ | _____ |

***6. COST**

6.1 The time (in years and months) that the system has been in use in a production environment: _____

6.2 The annual **total** cost to run the system: _____

6.3 The cost of development and maintenance to date:_____

6.4 The cost of hardware (including support) to date:_____

***7. QUALITY**

7.1 The degree to which the system satisfies user requirements:

| Unsatisfactory | Generally Unsatisfactory | Generally Satisfactory | Completely Satisfactory |
|---|---|---|---|

7.2 How accurate are the original specifications, i.e. how accurate do the original specifications describe the existing functionality of the system?

| Unsatisfactory | Generally Unsatisfactory | Generally Satisfactory | Completely Satisfactory |
|---|---|---|---|

7.3 Is additional functionality needed in the system?

| YES | NO |
|---|---|

7.4 Is additional ease of use needed?

| YES | NO |
|---|---|

7.5 Is the underlying technology working well or is it obsolete?

| Obsolete<br>Unsatisfactory | Obsolete<br>Satisfactory | Current<br>Unsatisfactory | Current<br>Satisfactory |
|---|---|---|---|

7.6 How accurate is system documentation?

| Unsatisfactory | Generally<br>Unsatisfactory | Generally<br>Satisfactory | Completely<br>Satisfactory |
|---|---|---|---|

7.7 What is the quality of the data, i.e. is it missing, incorrect or corrupt, does it require business rules for interpretation?

| Unsatisfactory | Generally<br>Unsatisfactory | Generally<br>Satisfactory | Completely<br>Satisfactory |
|---|---|---|---|

## *8. CRITICAL FACTORS

Indicate the criticality of the following factors:

8.1 Cost of running the system:

| | | | |
|---|---|---|---|
| Not Critical | Fairly Critical | Critical | Extremely Critical |

8.2 Security:

| | | | |
|---|---|---|---|
| Not Critical | Fairly Critical | Critical | Extremely Critical |

8.3 Availability:

| | | | |
|---|---|---|---|
| Not Critical | Fairly Critical | Critical | Extremely Critical |

8.4 System Performance:

| | | | |
|---|---|---|---|
| Not Critical | Fairly Critical | Critical | Extremely Critical |

8.5 Others:

_____

_____

_____

_____

***9. BUSINESS SUPPORT**

9.1 How well are organisational processes supported by the system:

| Unsatisfactory | Generally Unsatisfactory | Generally Satisfactory | Completely Satisfactory |
|---|---|---|---|

9.2 Is the system mission-critical for the user's business, i.e. will money be lost if the system fails?

| YES | NO |
|---|---|

**10. AVAILABLE SKILLS**

10.1 What percentage of the employees involved in the development and maintenance of the system, is still employed by the organisation?

| <=25% | <=50% | <=75% | > 75% |
|---|---|---|---|

10.2 What percentage of the employees with the necessary skills to maintain the system is still employed by the organisation?

| <=25% | <=50% | <=75% | > 75% |
|---|---|---|---|

10.3 How difficult is it to find people with the appropriate skills to maintain the system in the external market?

| Very Easy | Fairly Easy | Fairly Difficult | Extremely Difficult |
|---|---|---|---|

***11. UNIQUENESS**

11.1 What is the nature of the system, e.g. financial, material management?

_____

11.2 How complex is the system?

| Low Complexity | Fair Complexity | Medium Complexity | High Complexity |
|---|---|---|---|

11.3 Are the functional aspects of the system unique to the organisation?

| Not Unique | Fairly Unique | Unique | Extremely Unique |
|---|---|---|---|

11.4 Does packaged software exist with similar functionality?

| YES | NO |
|---|---|

***12. MAINTENANCE HISTORY**

12.1 How often is corrective maintenance required?

| Very Seldom | Fairly Seldom | Fairly Often | Very Often |
|---|---|---|---|

12.2 Is the corrective maintenance limited to certain parts of the system?

| YES | NO |
|---|---|

***13. OTHER**

Please add any other comments you regard as interesting and/or relevant (use another page if necessary):

_____

_____

_____

_____

_____

_____

_____

_____

**THE END - Thank you for your time and co-operation!**

# APPENDIX

# B

## Questionnaire Results

| Question No | System | Unsatisfactory | Generally Unsatisfactory | Generally Satisfactory | Completely Satisfactory | No Response | Total |
|---|---|---|---|---|---|---|---|
| 7.1 | MMS | 0 | 0 | 7 | 0 | 0 | 7 |
| | FS | 0 | 0 | 6 | 0 | 0 | 6 |
| | GL | 0 | 1 | 2 | 0 | 0 | 3 |
| | PAMM | 0 | 0 | 3 | 1 | 0 | 4 |
| | MIMS | 0 | 0 | 5 | 1 | 0 | 6 |
| 7.2 | MMS | 0 | 0 | 7 | 0 | 0 | 7 |
| | FS | 0 | 1 | 4 | 0 | 1 | 6 |
| | GL | 0 | 1 | 2 | 0 | 0 | 3 |
| | PAMM | 0 | 0 | 4 | 0 | 0 | 4 |
| | MIMS | 0 | 0 | 5 | 1 | 0 | 6 |
| 7.3* | MMS | 4 | 0 | 0 | 3 | 0 | 7 |
| | FS | 2 | 0 | 0 | 4 | 0 | 6 |
| | GL | 3 | 0 | 0 | 0 | 0 | 3 |
| | PAMM | 4 | 0 | 0 | 0 | 0 | 4 |
| | MIMS | 6 | 0 | 0 | 0 | 0 | 6 |
| 7.4* | MMS | 5 | 0 | 0 | 2 | 0 | 7 |
| | FS | 4 | 0 | 0 | 2 | 0 | 6 |
| | GL | 1 | 0 | 0 | 2 | 0 | 3 |
| | PAMM | 2 | 0 | 0 | 2 | 0 | 4 |
| | MIMS | 6 | 0 | 0 | 0 | 0 | 6 |
| 7.5 | MMS | 0 | 0 | 4 | 2 | 1 | 7 |
| | FS | 0 | 1 | 1 | 2 | 2 | 6 |
| | GL | 0 | 0 | 1 | 2 | 0 | 3 |
| | PAMM | 0 | 0 | 0 | 4 | 0 | 4 |
| | MIMS | 3 | 1 | 0 | 2 | 0 | 6 |
| 7.6 | MMS | 0 | 1 | 4 | 1 | 1 | 7 |
| | FS | 1 | 2 | 3 | 0 | 0 | 6 |
| | GL | 0 | 1 | 0 | 2 | 0 | 3 |
| | PAMM | 0 | 0 | 2 | 2 | 0 | 4 |
| | MIMS | 1 | 2 | 2 | 1 | 0 | 6 |
| 7.7 | MMS | 0 | 0 | 6 | 0 | 1 | 7 |
| | FS | 0 | 0 | 5 | 1 | 0 | 6 |
| | GL | 0 | 0 | 0 | 3 | 0 | 3 |
| | PAMM | 0 | 0 | 4 | 0 | 0 | 4 |
| | MIMS | 3 | 0 | 2 | 1 | 0 | 6 |
| TOTAL | MMS | 9 | 1 | 28 | 8 | 3 | 49 |
| | FS | 7 | 4 | 19 | 9 | 3 | 42 |
| | GL | 4 | 3 | 5 | 9 | 0 | 21 |
| | PAMM | 6 | 0 | 13 | 9 | 0 | 28 |
| | MIMS | 19 | 3 | 14 | 6 | 0 | 42 |

**Table B.1** Quality

* "Yes" to the answer was regarded as "Unsatisfactory" whereas "No" was regarded as "Completely Satisfactory".

| Question No | System | Not Critical | Fairly Critical | Critical | Extremely Critical | No Response | Total |
|---|---|---|---|---|---|---|---|
| 8.1 | MMS | 0 | 2 | 3 | 1 | 1 | 7 |
| | FS | 0 | 2 | 0 | 4 | 0 | 6 |
| | GL | 0 | 2 | 1 | 0 | 0 | 3 |
| | PAMM | 0 | 1 | 3 | 0 | 0 | 4 |
| | MIMS | 0 | 3 | 1 | 0 | 2 | 6 |
| 8.2 | MMS | 2 | 2 | 1 | 1 | 1 | 7 |
| | FS | 0 | 1 | 3 | 2 | 0 | 6 |
| | GL | 0 | 0 | 3 | 0 | 0 | 3 |
| | PAMM | 0 | 1 | 3 | 0 | 0 | 4 |
| | MIMS | 1 | 0 | 4 | 1 | 0 | 6 |
| 8.3 | MMS | 2 | 1 | 2 | 1 | 1 | 7 |
| | FS | 0 | 1 | 4 | 1 | 0 | 6 |
| | GL | 0 | 0 | 2 | 1 | 0 | 3 |
| | PAMM | 0 | 0 | 2 | 2 | 0 | 4 |
| | MIMS | 1 | 0 | 3 | 2 | 0 | 6 |
| 8.4 | MMS | 0 | 1 | 3 | 2 | 1 | 7 |
| | FS | 0 | 2 | 3 | 1 | 0 | 6 |
| | GL | 0 | 0 | 2 | 1 | 0 | 3 |
| | PAMM | 0 | 0 | 4 | 0 | 0 | 4 |
| | MIMS | 1 | 0 | 3 | 2 | 0 | 6 |

**Table B.2** Critical Factors

| Question No | System | Unsatisfactory | Generally Unsatisfactory | Generally Satisfactory | Completely Satisfactory | No Response | Total |
|---|---|---|---|---|---|---|---|
| 9.1 | MMS | 0 | 0 | 7 | 0 | 0 | 7 |
| | FS | 0 | 0 | 5 | 0 | 1 | 6 |
| | GL | 0 | 0 | 3 | 0 | 0 | 3 |
| | PAMM | 0 | 0 | 3 | 1 | 0 | 4 |
| | MIMS | 0 | 2 | 2 | 1 | 1 | 6 |
| 9.2* | MMS | 0 | 0 | 0 | 6 | 1 | 7 |
| | FS | 0 | 0 | 0 | 6 | 0 | 6 |
| | GL | 2 | 0 | 0 | 1 | 0 | 3 |
| | PAMM | 0 | 0 | 0 | 4 | 0 | 4 |
| | MIMS | 0 | 0 | 0 | 5 | 1 | 6 |
| TOTAL | MMS | 0 | 0 | 7 | 6 | 1 | 14 |
| | FS | 0 | 0 | 5 | 6 | 1 | 12 |
| | GL | 2 | 0 | 3 | 1 | 0 | 6 |
| | PAMM | 0 | 0 | 3 | 5 | 0 | 8 |
| | MIMS | 0 | 2 | 2 | 6 | 2 | 12 |

**Table B.3** Business Support

* "Yes" to the answer was regarded as "Completely Satisfactory" whereas "No" was regarded as "Unsatisfactory".

| Question No | System | Unavailable | Fairly Available | Available | Highly Available | No Response | Total |
|---|---|---|---|---|---|---|---|
| 10.1 | MMS | 1 | 1 | 0 | 0 | 0 | 2 |
| | FS | 1 | 0 | 1 | 0 | 0 | 2 |
| | GL | 1 | 1 | 0 | 0 | 0 | 2 |
| | PAMM | 0 | 0 | 0 | 2 | 0 | 2 |
| | MIMS | 0 | 1 | 0 | 0 | 0 | 1 |
| 10.2 | MMS | 0 | 2 | 0 | 0 | 0 | 2 |
| | FS | 0 | 1 | 1 | 0 | 0 | 2 |
| | GL | 0 | 2 | 0 | 0 | 0 | 2 |
| | PAMM | 0 | 0 | 0 | 2 | 0 | 2 |
| | MIMS | 0 | 1 | 0 | 0 | 0 | 1 |
| 10.3 | MMS | 0 | 2 | 0 | 0 | 0 | 2 |
| | FS | 0 | 2 | 0 | 0 | 0 | 2 |
| | GL | 0 | 2 | 0 | 0 | 0 | 2 |
| | PAMM | 0 | 2 | 0 | 0 | 0 | 2 |
| | MIMS | 1 | 0 | 0 | 0 | 0 | 1 |
| Total | MMS | 1 | 5 | 0 | 0 | 0 | 6 |
| | FS | 1 | 3 | 2 | 0 | 0 | 6 |
| | GL | 1 | 5 | 0 | 0 | 0 | 6 |
| | PAMM | 0 | 2 | 0 | 4 | 0 | 6 |
| | MIMS | 1 | 2 | 0 | 0 | 0 | 3 |

**Table B.4** Available Skills

| Question No | System | Low Complexity | Fair Complexity | Medium Complexity | High Complexity | No Response | Total |
|---|---|---|---|---|---|---|---|
| 11.2 | MMS | 0 | 3 | 2 | 1 | 1 | 7 |
| | FS | 0 | 1 | 4 | 1 | 0 | 6 |
| | GL | 0 | 0 | 3 | 0 | 0 | 3 |
| | PAMM | 0 | 2 | 2 | 0 | 0 | 4 |
| | MIMS | 0 | 0 | 4.5* | 0.5* | 1 | 6 |

**Table B.5** Complexity

* An answer was split between two alternate response options.

| Question No | System | Not Unique | Fairly Unique | Unique | Extremely Unique | No Response | Total |
|---|---|---|---|---|---|---|---|
| 11.3 | MMS | 0 | 2 | 3 | 0 | 2 | 7 |
| | FS | 0 | 4 | 2 | 0 | 0 | 6 |
| | GL | 1.5* | 0 | 1 | 0.5* | 0 | 3 |
| | PAMM | 0 | 3 | 1 | 0 | 0 | 4 |
| | MIMS | 3 | 0 | 2 | 1 | 0 | 6 |
| 11.4** | MMS | 1 | 0 | 0 | 3 | 3 | 7 |
| | FS | 1 | 0 | 0 | 4 | 1 | 6 |
| | GL | 2.5* | 0 | 0 | 0.5* | 0 | 3 |
| | PAMM | 1 | 0 | 0 | 2 | 1 | 4 |
| | MIMS | 6 | 0 | 0 | 0 | 0 | 6 |
| Total | MMS | 1 | 2 | 3 | 3 | 5 | 14 |
| | FS | 1 | 4 | 2 | 4 | 1 | 12 |
| | GL | 4 | 0 | 1 | 1 | 0 | 6 |
| | PAMM | 1 | 3 | 1 | 2 | 1 | 18 |
| | MIMS | 9 | 0 | 2 | 1 | 0 | 12 |

**Table B.6** Uniqueness

| Question No | System | Very Seldom | Fairly Seldom | Fairly Often | Very Often | No Response | Total |
|---|---|---|---|---|---|---|---|
| 12.1 | MMS | 0 | 4 | 2 | 0 | 1 | 7 |
| | FS | 0 | 4 | 1 | 0 | 1 | 6 |
| | GL | 1.5* | 0 | 1.5* | 0 | 0 | 3 |
| | PAMM | 0 | 1 | 2 | 0 | 1 | 4 |
| | MIMS | 1 | 1 | 3 | 0 | 1 | 6 |
| 12.2*** | MMS | 4 | 0 | 0 | 2 | 1 | 7 |
| | FS | 1 | 0 | 0 | 4 | 1 | 6 |
| | GL | 1 | 0 | 0 | 2 | 0 | 3 |
| | PAMM | 2 | 0 | 0 | 0 | 2 | 4 |
| | MIMS | 2 | 0 | 0 | 3 | 4 | 6 |
| Total | MMS | 4 | 4 | 2 | 2 | 2 | 14 |
| | FS | 1 | 4 | 1 | 4 | 2 | 12 |
| | GL | 2.5 | 0 | 1.5 | 2 | 0 | 6 |
| | PAMM | 2 | 1 | 2 | 0 | 3 | 8 |
| | MIMS | 3 | 1 | 3 | 3 | 2 | 12 |

**Table B.7** Maintenance History

---

* An answer was split between two alternate response options.

** "Yes" to the answer was regarded as "Not Unique" whereas "No" was regarded as "Extremely Unique".

*** "Yes" to the answer was regarded as "Very Seldom" whereas "No" was regarded as "Very Often".

# APPENDIX

# C

## Object-Oriented (OO) Principles

## C.1 Introduction

OO principles, which are excellent for managing complexity, are applied throughout all cycles of the integration methodology. The fundamental principles of OO are identity, data abstraction, data encapsulation, information hiding, classification, inheritance, association, polymorphism and modularity. Some of these principles, i.e. data encapsulation, information hiding and modularity, are derived from traditional structured principles. Each of the principles of OO can be used in isolation, but together they complement each other synergistically. The interpretation of these concepts varies and varying degrees of OO are distinguished. Wilkie (1993) distinguishes:

1. **object-based systems**, which support the functionality of objects without supporting classes and inheritance;

2. **class-based systems**, which support the functionality of objects and classes without supporting inheritance;

3. **OO systems**, which support objects, classes and inheritance.

The above-mentioned principles, as well as the OO relationships for structuring are now reviewed.

## C.2 OO Principles

**Identity** is the property of an object that distinguishes it from all other objects. Identity means that data is quantised into discrete, distinguishable entities called objects. Each object has its own unique inherent identity. This identity never changes and does not depend on the object's name or location. The identity is implemented through an object identifier attribute during object creation. An object consists of a data structure (attributes) and operations. Two objects are distinct, even if all their attribute values (such as name and age) are identical. In Figure C.1 the object *Person* has attributes *name* and *age* and operations *change-job* and *change-address*. Identity allows the view of both concrete and abstract entities, together with their relevant operations, as a modelling primitive.

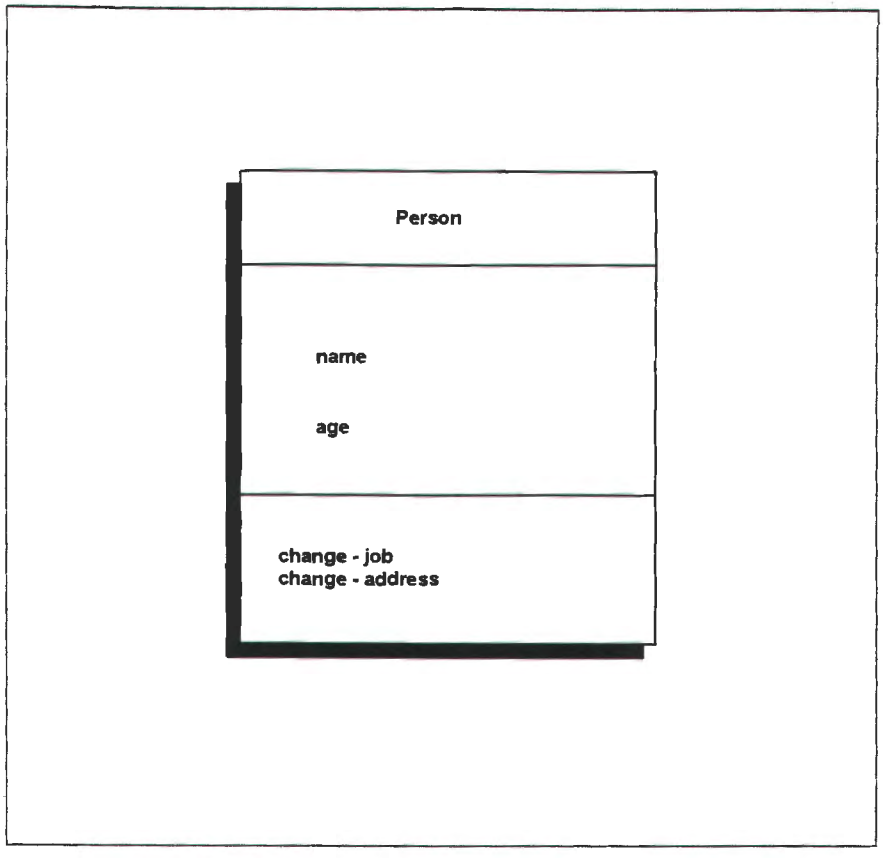


**Figure C.1** An Object

**Data abstraction** refers to the selective examination of certain aspects of a complex problem, in order to isolate those aspects that are important for some purpose, and suppress those aspects that are unimportant until a later stage (Rumbaugh et al,1991). It allows the designer to think at the level of the data structure and the operations performed on it, and only later to be concerned with the details of how that data structure and operations are to be implemented. It is a technique used to master complexity by identifying the important aspects of a phenomenon and ignoring its details. Booch (1994) defines abstraction as:

> ***" An abstraction denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer."***

Procedural abstraction allows the designer to conceptualise the product in terms of high-level procedures, which will only at a later stage be defined in terms of lower-level procedures, until finally, the lowest level is reached. The designer can therefore, at any level, ignore the levels above, as well as the levels below and only be concerned with expressing the product in terms of procedures appropriate to that specific level (Schach,1993).

Iteration abstraction allows a programmer to specify, at a higher level, that a loop is to be used, and then to describe, at a lower level, the exact elements over which the iteration is to be performed, as well as the order in which the elements are to be processed (Schach,1993).

**Data Encapsulation** refers to a data structure, together with the operations to be performed on that data structure. It is a modelling and implementation technique which allows the designer to think at the level of the data structure and its operations, and only later be concerned with the details of how that data structure and operations are to be implemented. Various authors refer to encapsulation and information hiding as if they are synonyms (Meyer,1988; Rumbaugh et al,1991; Booch,1994). However, for the purpose of this dissertation, data encapsulation is considered to be an example of abstraction.

**Information hiding** is a modelling and implementation technique that separates the external aspects (the interface), from the internal, implementation details of the object. The external aspects of an object are accessible to other objects whereas the internal details of an object are hidden from other objects. Users understand what operations (services) may be requested of the object but do not know the details of how the operation (service) is performed (Cox,1986). The data of an object is accessed via its own methods. This protects an object's data from corruption.

As the implementation of an object can be changed without affecting the application that use it, it prevents a program from becoming so interdependent that a small change has massive ripple effects. Although not unique to OO languages, the combination of data structure and behaviour (operations) in a single entity makes encapsulation more powerful than in conventional languages where the data structure and behaviour are separated (Rumbaugh et al,1991). Procedural and iteration abstraction are instances of information hiding.

**Classification** means that objects with the same data structure (attributes) and behaviour operations are grouped into a class (Rumbaugh et al,1991; Booch,1994). A class is an abstract data type that supports inheritance. It describes properties important to an application and ignores the rest. In OO languages, object types are implemented as classes. Each object forms a unique instance of its class. In Figure C.2 there are two object instances of the class *person*. By the grouping of uniform objects, complexity is reduced.

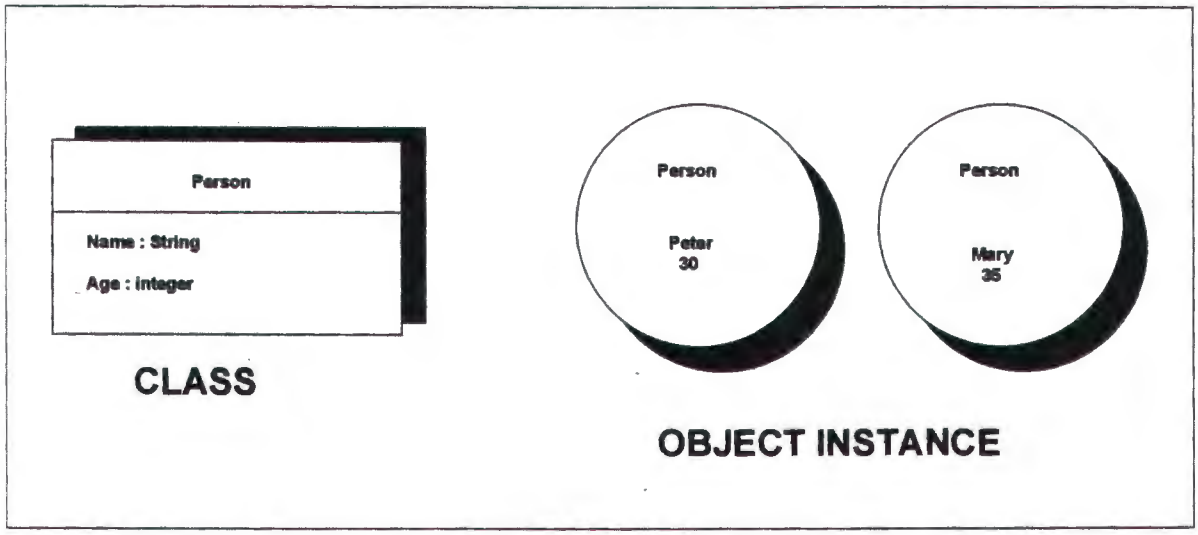


**Figure C.2** A Class with Two Object Instances

**Inheritance** is a mechanism that permits classes to share attributes and operations based on a hierarchical relationship (usually generalisation) (Rumbaugh et al,1991; Booch,1994). The superclass is the class being refined and each refined version is called a subclass. Subclasses inherit both the structure (i.e. the concrete representation of the state of an object) as well as the behaviour of their superclasses. When a class is refined or specialised into successively lower level subclasses, each subclass inherits all the attributes of its superclass and in addition may add its own features. This phenomena is referred to as specialisation. An object which is a member of a class, inherits all the properties of the class. In Figure C.3 the subclasses *employee* and *student* inherits the attributes *name* and *address* from its superclass.

Inheritance provides conceptual simplification that comes from reducing the number of independent features of a system and allows the definition of new data types as extensions of previously defined types in a hierarchical relationship. It reduces the amount of redundant code in a system. Multiple

inheritance permits a class to have more than one superclass and to inherit features from all ancestors (Rumbaugh et al, 1991). Inheritance is a key reusability principle that is unique to the OO paradigm.

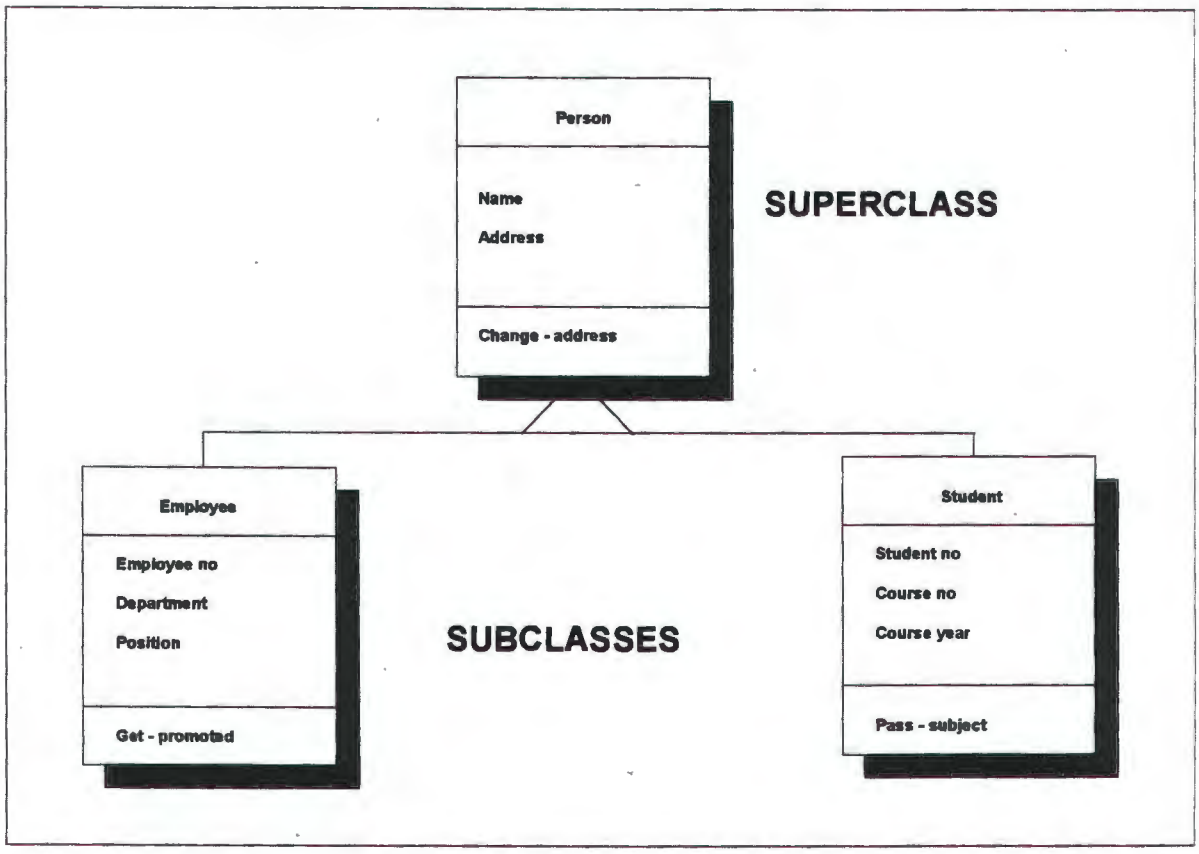


**Figure C.3** Inheritance

**Polymorphism** means that the same operation may behave differently on different classes, i.e. an operation takes on many forms of implementation, depending on the type of object. Meyer (1988) views polymorphism in OO programming as the ability to refer at run-time to instances of various classes. A strength of polymorphism is that a request for an operation can be made without knowing which method should be invoked. These implementation details are hidden from the user.

In Figure C.4 the *Employee* class defines a *retire* operation. In OO implementations, this operation is automatically inherited by all the subclasses of *employee*. An organisation may have different methods for retiring an executive than for retiring an employee. In this situation, the method for

retiring executives overrides the method for retiring employees in general. The example is polymorphic, because the retire operation has a different method of implementation depending on whether an object is an *employee* or an *executive*.

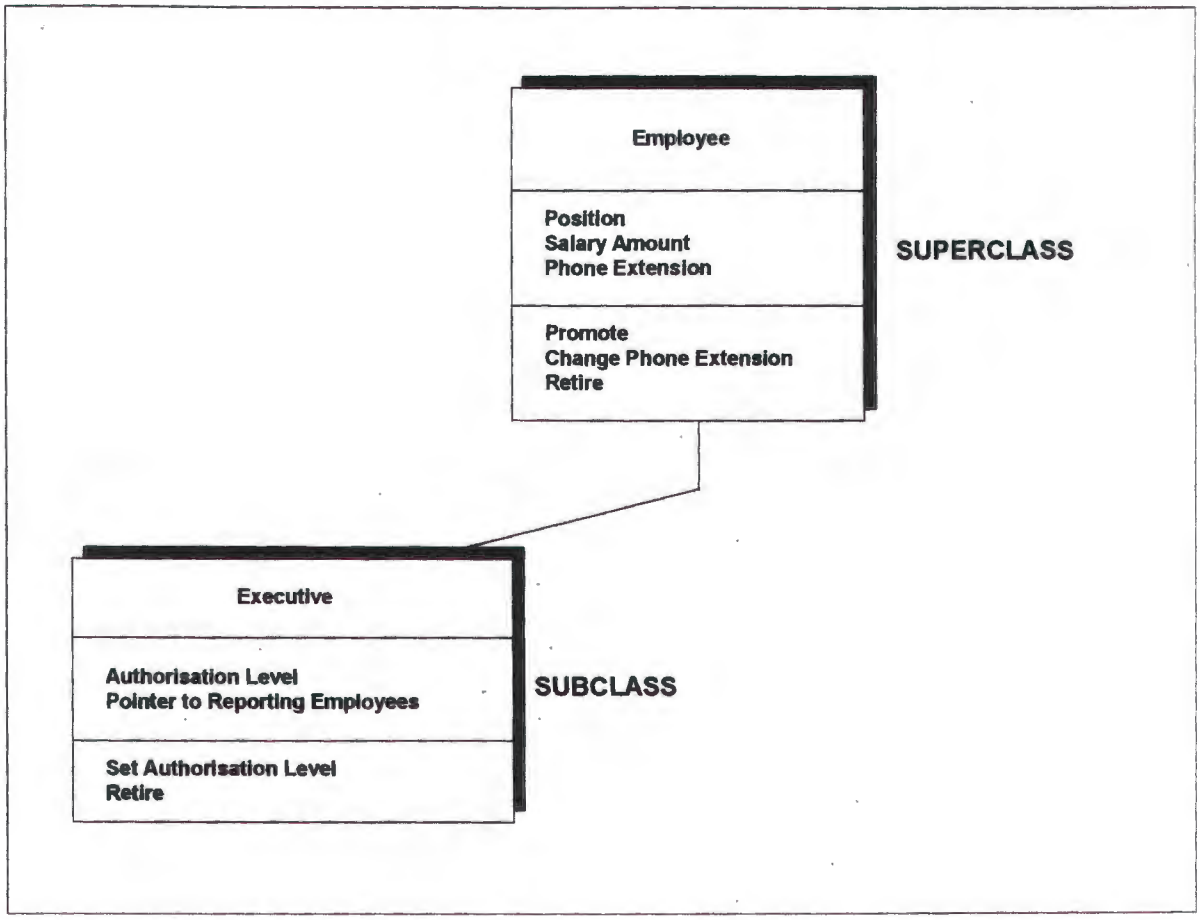


**Figure C.4** A Polymorphic *retire* operation

Object classes provide a natural unit of **modularity**. Booch (1994) defines modularity as:

> ***"the property of a system that has been decomposed into a set of cohesive and loosely coupled modules."***

A module is a logical construct for grouping classes that captures some logical subset of the entire model. Modules provide an intermediate unit of packaging between an entire object model and the

basic building blocks of class and association. It allows the partitioning of a model into manageable pieces (Rumbaugh et al,1991).

## C.3 Relationships for Structuring

Structural concepts relate to a description of aspects of a system concerned with relationships amongst classes. A relationship represents some logical connection between classes. A class is a description of a group of instances with similar properties, common behavioural semantics and relationships. A class represents a particular implementation of a type. OO provides three basic types of class relationships for structuring:

1. the generalisation / specialisation relationship;

2. the whole / part relationship;

3. association.

In a generalisation / specialisation relationship, a subclass specialises the more general structure or behaviour of its superclasses. "Is a" hierarchies therefore denote generalisation / specialisation relationships.

Whole / part relationships are described by "part of" hierarchies, e.g. a *petal* is not a kind of *flower*, it is a part of a *flower*. In terms of its "is a" hierarchy, a high-level abstraction is generalised whereas a low-level abstraction is specialised. A *person* class is at a higher level of abstraction than a *student* class. In terms of its "part of" hierarchy, a class is at a higher level of abstraction than any of the classes that make up its implementation. The class *garden* is at a higher level of abstraction than the class *plant*, upon which it is built.

Association denotes some semantic dependency among otherwise unrelated classes, e.g. the classes *roses* and *candles* are largely independent, but they both represent things that might be used to decorate a dinner table.

According to Booch (1994), the approaches which have evolved in programming languages to capture these generalisation / specialisation, whole / part and association relationships, include:-

1. **Association**, i.e. a relationship among instances of two or more classes describing a group of links with common structure and common semantics, e.g. a person *works-for* a company. It is used to tie together certain events that happen at some point in time or under similar circumstances. An association describes a set of potential links in the same way that a class describes a set of potential objects.
2. **Inheritance** implies a generalisation / specialisation hierarchy. Semantically, inheritance denotes an "is a" relationship, e.g. a *student* "is a" kind of *person*.
3. **Aggregation**, i.e. a special form of association, between a whole and its parts, in which the whole is composed of the parts (Rumbaugh et al,1991). Although not unique to OO programming languages, the combination of inheritance with aggregation is very powerful. Aggregation allows the physical grouping of logically related structures and inheritance allows these common groups to be easily reused among different abstractions (Booch,1994).
4. **Using**, i.e. a relationship denoting that an instance of one class makes use of an instance of another. Objects interact by means of sending messages to one another. These messages may be asynchronous or synchronous, depending on the multitasking capabilities of the system under design.
5. **Instantiation**, i.e. the process of creating instances from classes, e.g. John Smith and Mary Bence are instances of class *Person*.
6. **Metaclass**, i.e. the class of a class. This concept allows for the treatment of classes as objects and is explicitly supported by languages such as Smalltalk and CLOS.

# APPENDIX

# D

## Risk Assessment Questionnaire

**Risk assessment questionnaire** sample from a total of 54 questions

| | Size risk assessment | | Weight |
|---|---|---|---|
| 1. | Total development man-hours for system* | | 5 |
| | 100 to 3,000 | Low-1 | |
| | 3,000 to 15,000 | Medium-2 | |
| | 15,000 to 30,000 | Medium-3 | |
| | More than 30,000 | High-4 | |
| 2. | What is estimated project implementation time? | | 4 |
| | 12 months or less | Low-1 | |
| | 13 months to 24 months | Medium-2 | |
| | More than 24 months | High-3 | |
| 3. | Number of departments (other than IS) involved | | 4 |
| | One | Low-1 | |
| | Two | Medium-2 | |
| | Three or more | High-3 | |
| | **Structure risk assessment** | | **Weight** |
| 1. | If replacement system is proposed, what percentage of existing functions are replaced on a one-to-one basis? | | 5 |
| | 0% to 25% | High-3 | |
| | 25% to 50% | Medium-2 | |
| | 50% to 100% | Low-1 | |
| 2. | What is severity of procedural changes in user department caused by proposed system? | | 5 |
| | Low-1 | | |
| | Medium-2 | | |
| | High-3 | | |
| 3. | Does user organisation have to change structurally to meet requirements of new system? | | 5 |
| | No | -0 | |
| | Minimal | Low-1 | |
| | Somewhat | Medium-2 | |
| | Major | High-3 | |
| 4. | What is general attitude of user? | | 5 |
| | Poor-anti data-processing solution | High-3 | |
| | Fair-some reluctance | Medium-2 | |
| | Good-understands value of DP solution | -0 | |
| 5. | How committed is upper-level user management to system? | | 5 |
| | Somewhat reluctant or unknown | High-3 | |
| | Adequate | Medium-2 | |
| | Extremely enthusiastic | Low-1 | |
| 6. | Has a joint data processing/user team been established? | | 5 |
| | No | High-3 | |
| | Part-time user representative appointed | Low-1 | |
| | Full-time user representative appointed | -0 | |
| | **Technology risk assessment** | | **Weight** |
| 1. | Which of the hardware is new to the company? | | 5 |
| | None | -0 | |
| | CPU | High-3 | |
| | Peripheral and/or additional storage | High-3 | |
| | Terminals | High-3 | |
| | Mini or micro | High-3 | |
| 2. | Is the system software (nonoperating system) new to IS project team?Ψ | | 5 |
| | No | -0 | |
| | Programming language | High-3 | |
| | Data base | High-3 | |
| | Data communications | High-3 | |
| | Other - specify | High-3 | |
| 3. | How knowledgeable is user in area of IS? | | 5 |
| | First exposure | High-3 | |
| | Previous exposure but limited knowledge | Medium-2 | |
| | High degree of capability | Low-1 | |
| 4. | How knowledgeable is user representative in proposed application area? | | 5 |
| | Limited | High-3 | |
| | Understands concept but no experience | Medium-2 | |
| | Has been involved in prior implementation efforts | Low-1 | |
| 5. | How knowledgeable is IS team in proposed application area? | | 5 |
| | Limited | High-3 | |
| | Understands concept but no experience | Medium-2 | |
| | Has been involved in prior implementation efforts | Low-1 | |

**Note :** Since the questions vary in importance, the company assigned weights to them subjectively. The numerical answer to the questions is multiplied by the question weight to calculate the question's contribution to the projects risk. The numbers are then added together to produce a risk score number for the project. Projects with risk scores within 10 points of each other are indistinguishable but those seperated by 100 points or more are very different to even the casual observer.

*Time to develop includes system design, programming, testing and installation.

ΨThis question is scored by multiplying the sum of the numbers attached to the positive response by the weight.

**Source:** This questionnaire is adapted from the Dallas Tire care no. 9-180-006 (Baston Mass: HBS Case Service, 1980).