



*Process MeNtOR:
An Object Oriented Software
Process*

Technical Overview

Consulting and Capability Improvement Division
Object Oriented Pty Ltd

October 1997

object
oriented pty. ltd.

Table of Contents

1 INTRODUCTION.....	2
2 BACKGROUND AND CREDENTIALS	3
3 WHAT IS <i>PROCESS MENTOR</i>?	4
4 NAVIGATING YOUR WAY AROUND <i>PROCESS MENTOR</i>.....	6
5 PRINCIPLES OF <i>PROCESS MENTOR</i>	9
6 SEPA (SOFTWARE ENGINEERING PROCESS ARCHITECTURE).....	11
7 SOFTWARE ENGINEERING PROCESSES.....	17
8 ANALYSIS AND DESIGN CONCEPTS.....	20
9 DEPLOYING <i>PROCESS MENTOR</i>.....	26
10 DISCUSSION	30
11 BIBLIOGRAPHY	31

1 Introduction

Object-oriented technology is gaining widespread acceptance in the information technology arena. Many organisations are investigating the technology and considering what the implications are for their system development efforts.

To maximise the effectiveness of object-oriented development, considerable emphasis must be placed on object-oriented analysis and design techniques. Within mature organisations, these techniques need to be formalised within an object-oriented methodology (also known as a software process).

Object Oriented Pty Ltd has developed a pragmatic and industrial-strength, object-oriented (OO) software process called *Process MeNtOR*[™].

This white paper provides a brief background to *Process MeNtOR*, a discussion of what *Process MeNtOR* is, how it has been developed, how it is structured, and how it should be adopted and used within an organisation.

2 Background and Credentials

As a leader in object-oriented technology, Object Oriented Pty Ltd is in an ideal position to assist organisations in adopting object-oriented methodologies. Our work on *Process MeNtOR* has evolved over eight years (more than 30 person years of development effort) of research, development and practical application. The experience and resources we offer includes specialist expertise in object-oriented methodologies and expertise in the area of object-oriented software development.

Object Oriented's senior consultants have been involved in advising clients on object-oriented methodologies for several years. Our Principal Consultant, Dr Julian Edwards has a Ph.D. in object-oriented methodologies with extensive experience in the development of object-oriented methodologies for large organisations. Dr. Edwards is the co-author of a book on object-oriented methodologies (Henderson-Sellers and Edwards, 1994) which has been published in Prentice-Hall's Object Oriented Series.

The strategy behind *Process MeNtOR* is based on a number of assumptions:

- Publicly available object-oriented methods provide a body of tested techniques and approaches which can be built upon and tailored by organisations.
- No single published object-oriented methodology provides a complete solution nor does it necessarily cover the complete software development lifecycle; each has its strengths and its weaknesses.
- A methodology should support developers, not restrict them.
- A methodology is not just a set of guidelines and deliverables, but is also an accumulation of an organisation's experience and hence is, to some degree, organisation-specific.
- A methodology will, by necessity, evolve over time.
- A methodology should be consistent with the relevant software standards which require increasing levels of maturity with respect to software processes (e.g. ISO9000 software standards, Software Engineering Institute's five-level Capability Maturity Model).

3 What is *Process MeNtOR*?

Process MeNtOR is a software process for building and implementing component-based software systems using object oriented, or object-based, development approaches. *Process MeNtOR* is to be used by those people involved in software development projects including project managers, quality assurance personnel, software engineers, testers and architects. The goal of *Process MeNtOR* is to produce *high quality systems which are delivered on time and on budget*. This is achieved by providing guidance and support for project team members through a project lifecycle, a set of deliverables and a set of processes for team members to follow. In essence, *Process MeNtOR* provides the blueprint and operating procedures for running a component based systems development organisation. *Process MeNtOR* is, therefore, a key tool for managing projects effectively and ensuring the resulting software systems are of high quality and are delivered on time and budget.

Process MeNtOR consists of (see *Figure 1*):

- Hard Copy Manuals
- Online Web Version
- Deliverable Templates
- Fully Worked Example.

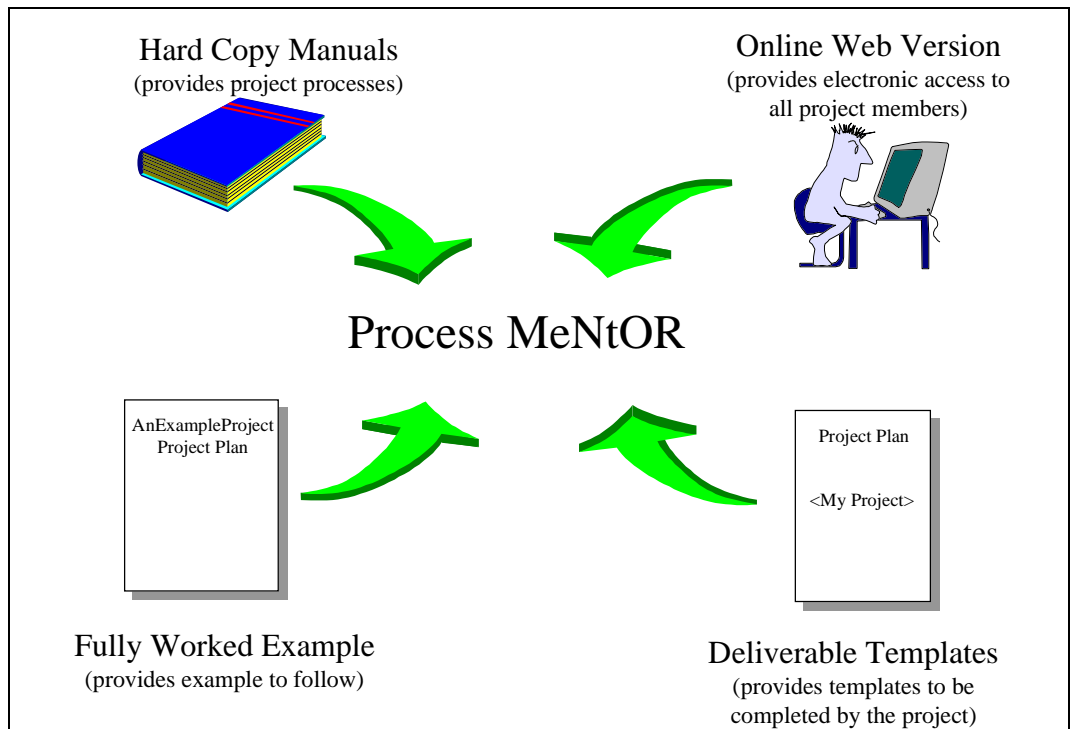


Figure 1 Components of *Process MeNtOR*

Process MeNtOR is supported by a complete set of deployment services to ensure the effective implementation of the process, including:

- formal training courses
- workshops
- consulting and coaching services.

Process MeNtOR provides projects with a tested approach to developing and managing a range of different projects, as well as providing a number of the key elements required for ISO 9000 and AS3563 quality accreditation.

3.1 The Development of *Process MeNtOR*

Process MeNtOR has evolved over eight years of research, development and practical application. Over the last eight years *Process MeNtOR* has been applied to numerous projects together with Object Oriented Pty Ltd's consulting and mentoring support. This practical experience has been combined with in-house and collaborative research efforts with a number of academic institutions.

Process MeNtOR is under continuous improvement through direct feedback from our clients and consultants as well as through Object Oriented Pty Ltd's own substantial internal development efforts.

Process MeNtOR is currently in Version 3 Release 1

3.2 Tool Support

CASE support for *Process MeNtOR* is available through a range of third party CASE products which support UML.

In general, CASE tools are those tools that provide support for analysis, design and documentation. Examples of tools that support object-oriented development that fall into this category include, amongst others, Rational Rose, Paradigm Plus, Select and System Architect. In general, most of these tools are customisable, or support a number of techniques and notations. Generally a full tools strategy will be required to most effectively deploy *Process MeNtOR* within the organisation.

4 Navigating Your Way Around *Process MeNtOR*

Process MeNtOR is structured into ten Process Manuals. Together these provide the details of the process, techniques and deliverables of *Process MeNtOR* (see *Figure 2*). These Process Manuals are:

- Process Manual 1 - Software Engineering Process Architecture
- Process Manual 2 - M-SEP1 - Large Scale Software Development Process
- Process Manual 3 - M-SEP2 - Small Scale Software Development Process
- Process Manual 4 - M-SEP3 - Package Procurement Process
- Process Manual 5 - Exploration & Development Process Units
- Process Manual 6 - Management & Testing Process Units
- Process Manual 7 - Deliverables
- Process Manual 8 - C++ Standards
- Process Manual 9 - Smalltalk Standards
- Process Manual 10 - Java Standards

An overview of each of these Process Manuals is provided below.

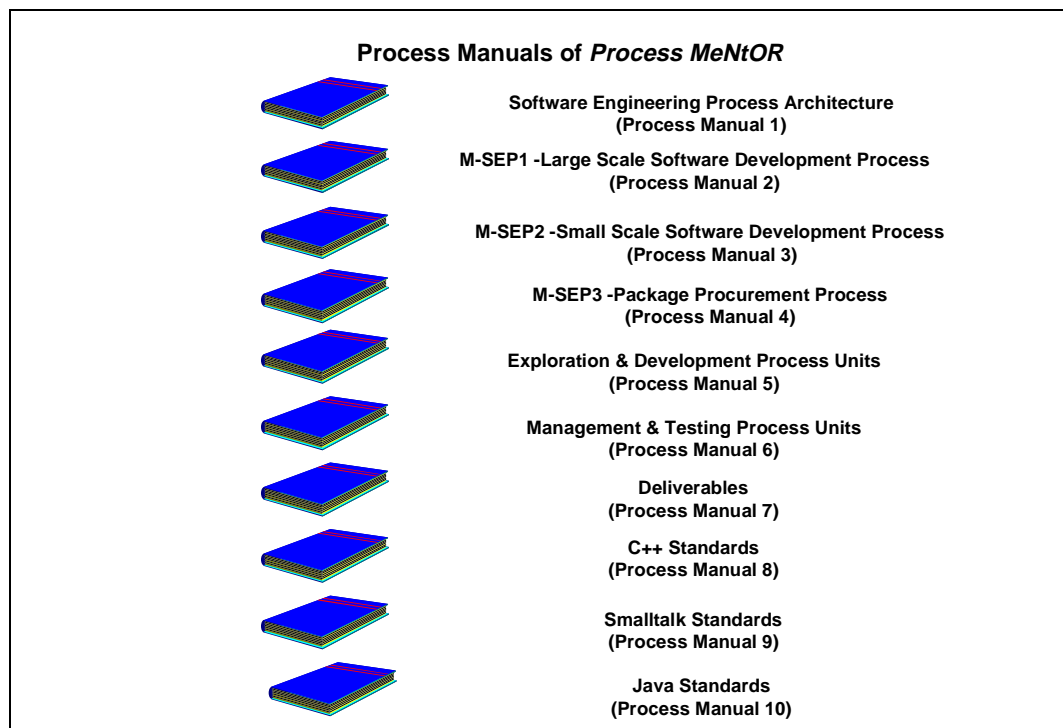


Figure 2 *Process MeNtOR* Process Manuals.

4.1 Process Manual 1: Software Engineering Process Architecture

This Process Manual provides the formal definition of the *Process MeNtOR* process architecture and contains those elements that are common across the approach such as the Concepts, Notation and Glossary. This Process Manual provides the key reference for the approach and should be

the entry point for people using *Process MeNtOR* on projects. This Process Manual should be read first to provide context for the other material.

4.2 Process Manual 2: MSEP-1 - Large Scale Software Development Process

This Process Manual provides a detailed description of the phases of *Process MeNtOR's* software engineering process for large scale software development known as MSEP-1. MSEP-1 is aimed at large scale software development projects and programmes of work. As such it provides a discussion of each phase, the activities and tasks to be undertaken and the deliverables to be produced for each phase.

This Process Manual should be read by those persons involved in managing and guiding large scale software *development* projects.

4.3 Process Manual 3: MSEP-2 - Small scale Software Development Process

This Process Manual provides a detailed description of the phases of *Process MeNtOR's* software engineering process for small to medium scale software development known as MSEP-2. MSEP-2 is aimed at small to medium scale software development projects. As such it provides a discussion of each phase, the activities and tasks to be undertaken and the deliverables to be produced for each phase.

This Process Manual should be read by those persons involved in managing and guiding small to medium scale software *development* projects.

4.4 Process Manual 4: MSEP-3 - Package Procurement Process

This Process Manual provides a detailed description of the phases of *Process MeNtOR's* Package Procurement Process known as MSEP-3. MSEP-3 is aimed at projects that involve procuring software packages. As such it provides a discussion of each phase, the activities and tasks to be undertaken and the deliverables to be produced for each phase of a project when procuring and deploying a piece of package software.

This Process Manual should be read by those persons involved in managing *package procurement* projects .

4.5 Process Manual 5: Exploration & Development Process Units

This Process Manual provides a detailed discussion of the activities and deliverables involved in exploring and developing projects within *Process MeNtOR*. It provides a detailed description of the principles, activities and tasks of each process including a detailed set of guidelines.

This Process Manual should be read by those persons involved in the core investigation and development processes of *Process MeNtOR*. This Process Manual is likely to be the most heavily utilised by all members of the project team.

4.6 Process Manual 6: Management & Testing Process Units

This Process Manual focuses on aspects of Project Management, Quality Assurance, Metrics and Testing. It covers issues such as review points, testing approaches and development of quality plans.

This Process Manual should be read by Project Managers, Quality Managers, Test Managers and others involved in the planning and review of projects. This Process Manual is likely to be utilised throughout the development activities.

4.7 Process Manual 7: Deliverables

This Process Manual is a collection of deliverable templates. Templates are default document structures for the deliverables defined by *Process MeNtOR*. Templates contain a considerable amount of guidance on the way in which deliverables should be completed and, as such, act as a form of guidelines.

This Process Manual will be utilised by all members of the project team throughout the phases of the project.

4.8 Process Manual 8: C++ Standards

This Process Manual provides the C++ standards and development guidelines used within *Process MeNtOR*. It should be read by all developers involved in C++ development.

4.9 Process Manual 9: Smalltalk Standards

This Process Manual provides the Smalltalk standards and development guidelines used within *Process MeNtOR*. It should be read by all developers involved in Smalltalk development.

4.10 Process Manual 10: Java Standards

This Process Manual provides the Java standards and development guidelines used within *Process MeNtOR*. It should be read by all developers involved in Java development.

5 Principles of *Process MeNtOR*

The figure below shows the underlying principles and constructs used within *Process MeNtOR*. These are discussed briefly below.

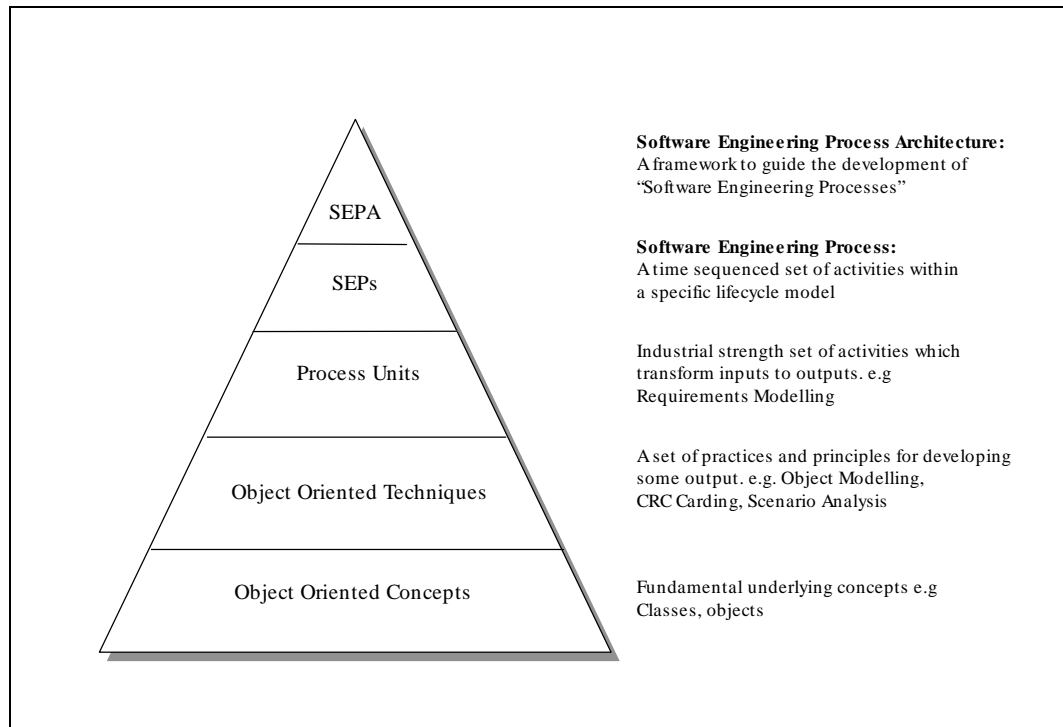


Figure 3 The fundamental principles underlying *Process MeNtOR*.

5.1 Concepts

Underlying the *Process MeNtOR* approach are a set of fundamental *concepts*. These fundamental concepts are related to the modelling methods and project management methods used within *Process MeNtOR*. These include the concepts of the object model including object, classes and inheritance as well as the concepts of iterative and incremental development and rapid application development. *Process MeNtOR* uses the industry standard for object oriented analysis and design known as the Unified Modeling Language.

5.2 Techniques

Built on these fundamental concepts are a set of *techniques* that detail tasks and guidelines for constructing models. Techniques include scenario analysis, CRC carding and object modelling.

5.3 Process Units

In order to apply these techniques in practice it is necessary to package them as *process units*. Process units make the techniques robust and usable in a practical settings by providing such things as deliverable templates, resource requirements and review guidelines. Process units are the reusable building blocks of the methodology and encapsulate a particular area of expertise such as testing or requirements gathering. *Process MeNtOR* currently supports 20 different process units.

5.4 Software Engineering Processes

Process units provide the activities, tasks and guidelines of the method. These are applied within a *Software Engineering Process (SEP)*. A software engineering process is the time sequenced set of activities required to transform a user's requirements into a successful system. A SEP represents the actual set of activities and tasks that a particular project will follow. A project is therefore planned and tracked against a software engineering process. *Process MeNtOR* provides a number of software engineering processes for use by different projects such as Software Development, Small Systems Development and Package Implementation.

5.5 Software Engineering Process Architecture

Just as one tool does not fit all jobs, so one software engineering process does not fit all projects types. There are many different possible software engineering processes, each of which is appropriate for a different project type. Software engineering processes can therefore be modelled and constructed just like any other system. The rules and concepts used for developing SEPs are termed a *Software Engineering Process Architecture (SEPA)*. A software engineering process architecture provides the foundation and framework within which additional SEPs are defined. In essence, a software engineering process architecture is the "method for developing methods".

A key feature of *Process MeNtOR* is that it has, at its core, such a software engineering process architecture. This means that *Process MeNtOR* is more than a single rigid approach to project management. Instead, it provides a framework, or architecture, for the construction of processes for managing projects. This approach provides the organisation with the consistency of a standard set of processes and techniques, but with the flexibility to combine them and tailor them as appropriate.

Process MeNtOR is, therefore, a sophisticated management tool which supports both the project and the organisation.

6 SEPA (Software Engineering Process Architecture)

The two major elements of the *Process MeNtOR* software engineering process architecture are:

- a set of process units
- a set of software lifecycle models

Each of these is discussed in the following sections.

6.1 *Process MeNtOR* Process Units

Process units are essentially reusable descriptions of how to undertake some part of the software development activity. Process units need not be bound to a specific phase of a software lifecycle. They define inputs, outputs (deliverables) and activities, and utilise a set of techniques and guidelines. Over time an architecture's set of processes may be specialised, extended and refined. This specialisation, extension and refinement will occur as the organisation learns more about its software development activities, as it moves into new software development areas and as it changes its practices.

Process units consist of *activities*. All activities undertaken during a software project are linked to a *process unit*. Activities may be further decomposed into *tasks* which are the smallest unit of work subject to management accountability. A task is a well defined work-assignment for one or more project members (see *Figure 4* The relationship between a Process unit, Activity and a Task.).

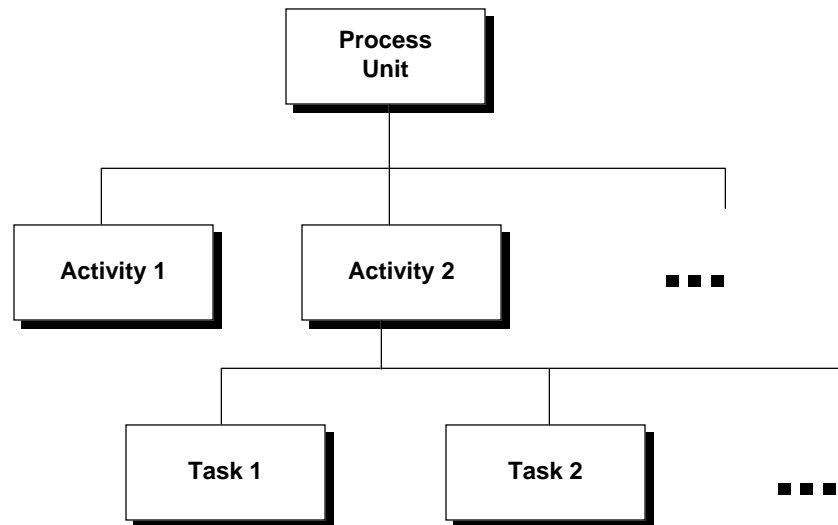


Figure 4 The relationship between a Process unit, Activity and a Task.

Processes, through the application of activities, result in deliverables. Upon completion of the process unit, the deliverable is considered complete. During the execution of a process unit a number of intermediate states of a deliverable may be produced. These intermediate states are useful when tracking the progress of a process unit and are termed a deliverable's *release states* (see *Figure 5*). Process units may execute in parallel with other process units and often interact

during a software project. This interaction may take the form of one process unit requiring the output of another process unit before it can be completed.

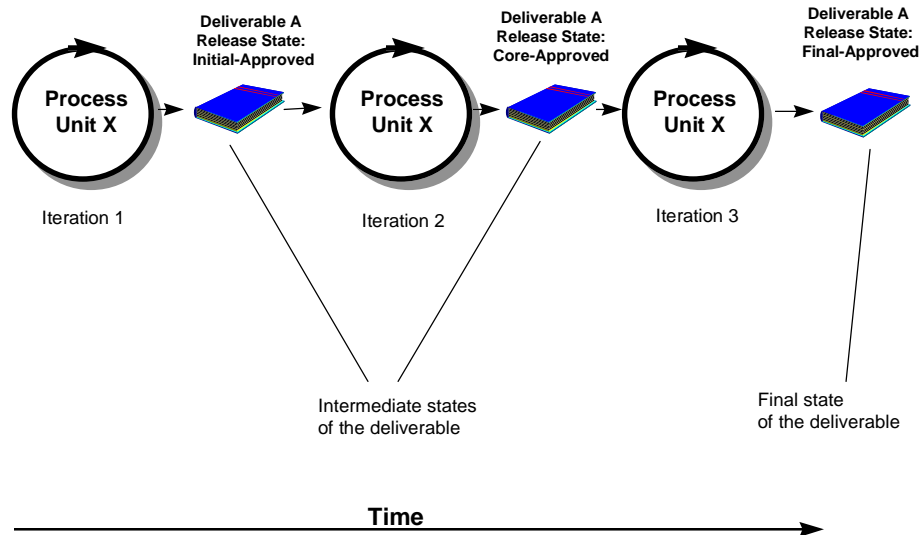


Figure 5 Release states of a deliverable produced during the execution of a process unit.

Process units are, therefore, the major reusable components of a software engineering process architecture. They provide the industrial strength version of a set of techniques which may be applied to the development of software.

Process MeNtOR defines 20 such process units which cover all aspects of the development and management processes. The following is a list of the process units:

- 1• SEP Construction
- 2• Concept Exploration
- 3• Alternative Evaluation
- 4• Programme Development
- 5• Business Engineering
- 6• Requirements Modelling
- 7• User Interface Modelling
- 8• System Modelling
- 9• Subsystem Modelling
- 10• Component Modelling
- 11• Prototyping
- 12• Installation
- 13• Installation process
- 14• Project Management
- 15• Software Cost Estimation
- 16• Quality Assurance

- 17• Software Configuration Management
- 18• Test Management
- 19• System Testing
- 20• Acceptance Testing process

6.2 Process MeNtOR Software Lifecycle Models

Software lifecycle models are the second major component of the process architecture.

A *software lifecycle model* is a framework for the way in which a project or some component of the project may be run. A software lifecycle model is a general description, or template, for projects, or parts of projects.

Examples of software lifecycle models include the waterfall model, the spiral model and the iterative model. Object-oriented development is most usually employed with an iterative or incremental software lifecycle model although this is not mandatory within *Process MeNtOR*. Each of these models provides an abstract description, and a set of basic rules and constraints, of the way in which a software development project may be run. Because they are general models of software development, software lifecycles models are defined and managed at the organisation level (see *Figure 6*) within the context of a software engineering process architecture.

Software lifecycle models provide a macro-structure for projects by defining key points within the process when reviews and assessments are made of the progress of the project.

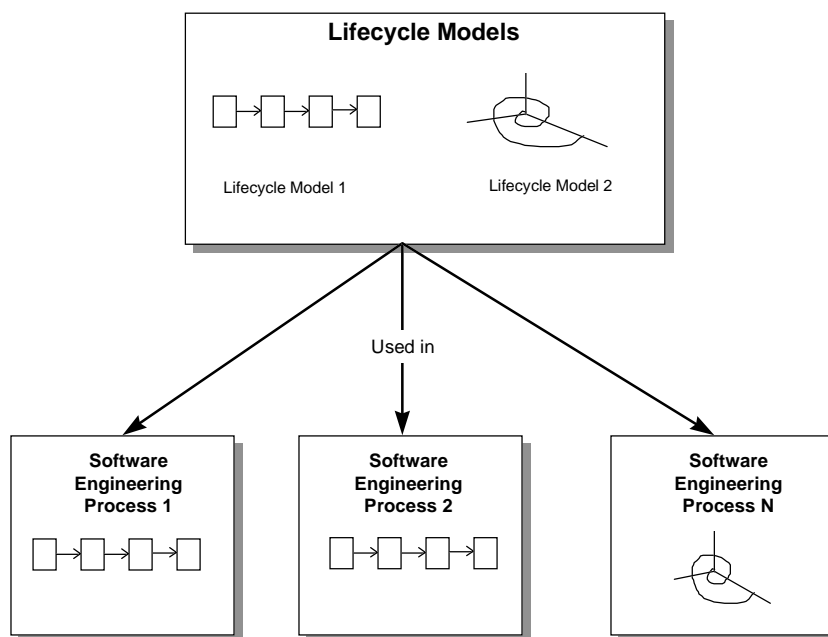


Figure 6 Software lifecycle models are applicable at the organisation level. Projects use instances of these models to actually build software.

Process MeNtOR defines four software lifecycle models. These are the waterfall, incremental, iterative and incremental/iterative software lifecycle models. Any of these software lifecycle models (as well as others which the organisation may define) could be used as the framework for a software engineering process within *Process MeNtOR*.

It is the role of the Project Manager, together with assistance from the Software Process Team, to select a software lifecycle model and to map the appropriate process units to that model.

6.2.1 Waterfall Software Lifecycle Model

The waterfall software lifecycle model is one of the most well-known and well-used software lifecycle models in the development of software today. In a waterfall software lifecycle model, the project follows the phases of the lifecycle in a sequential fashion (*Figure 7*). The Project Manager is still responsible for mapping the process units to the phases but each phase will be undertaken only once and in a sequential manner.

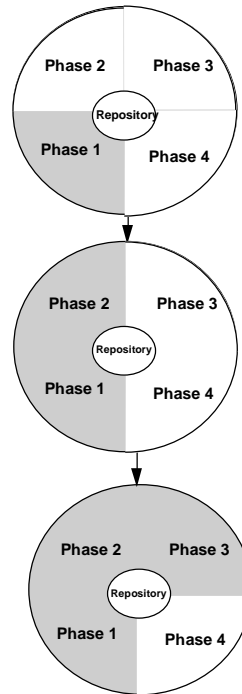


Figure 7 Waterfall Software Lifecycle Model.

6.2.2 Incremental Software Lifecycle Model

An incremental software lifecycle model is one where a project is decomposed into a number of discrete portions with each portion undergoing the full set of defined phases of the software lifecycle once (*Figure 8*). Thus each portion of the software system undergoes the phases of the software lifecycle once, but as there are several portions of the software system the phases of the software lifecycle are repeated once for each portion. In this software lifecycle model the software system is developed incrementally. This software lifecycle model may be used where a system has clearly defined functionality that can be delivered separately to the customer.

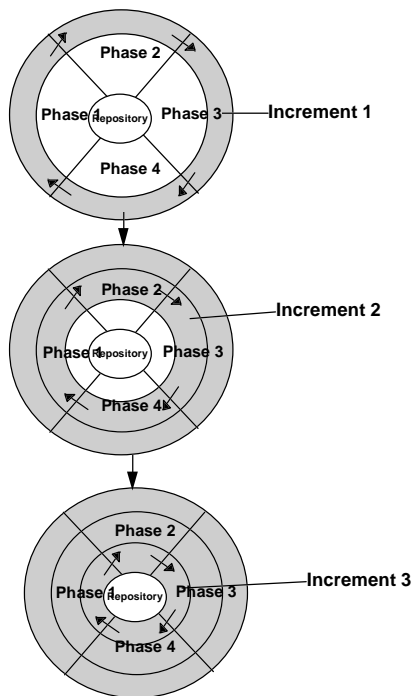


Figure 8 Incremental Software Lifecycle Model.

6.2.3 Iterative Software Lifecycle Model

An iterative software lifecycle model is one where the whole software system will undergo the phases (or some of the phases) of the software lifecycle a number of times. Thus in *Figure 9* the overall software system repeats several of the phases as it iterates over the software lifecycle. This software lifecycle model may be used in projects where the requirements are very poorly defined such as in a decision support system.

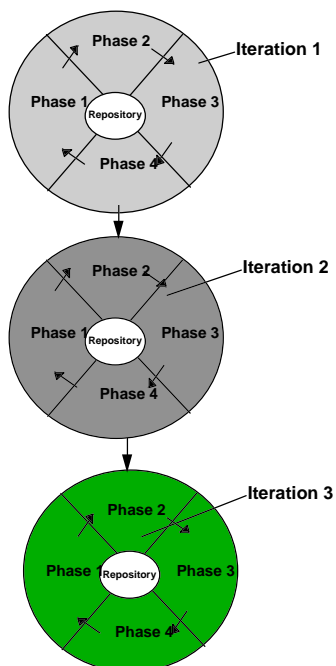


Figure 9 Iterative Software Lifecycle Model.

6.2.4 Iterative & Incremental Software Lifecycle Model

A combined iterative and incremental software lifecycle model is one where the software system will be divided into a number of portions each of which will undergo the phases of the software lifecycle a number of times.

Thus in *Figure 10*, each portion of the software system undergoes several iterations of the software lifecycle. Thus the software lifecycle for the overall project is iterative and incremental. The use of incremental development allows the project to deliver core functionality early in a project, prioritised on the basis of functionality required by the users. Iterative development permits the developer to refine the requirements with the user over a number of iterations with the use of prototypes. This may be applied on large projects where portions of the problem are not well understood.

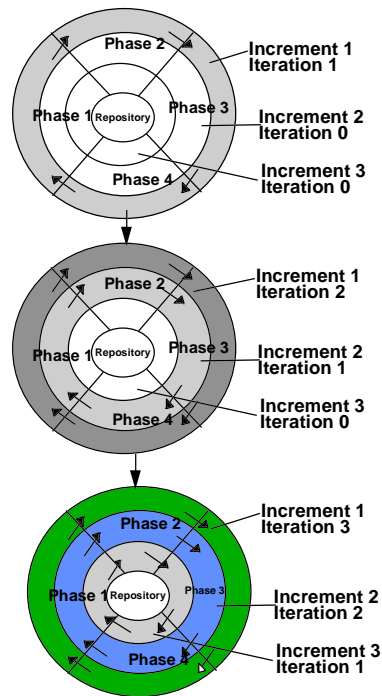


Figure 10 Iterative and Incremental Software Lifecycle Model.

7 Software Engineering Processes

A *software engineering process* (SEP) is the total set of software engineering activities needed to transform user requirements into software. They are created by mapping process units, which are not time-sequenced, to a software lifecycle model to create a time-sequenced set of activities.

A SEP, therefore, defines one particular way of developing software and is itself developed using a process architecture. It is the SEP that is used by projects within an organisation.

Developing or identifying a specific SEP occurs at the start of a project. Over time an organisation using a process architecture will develop a library of SEPs for specific project types. This enables new projects to *select* a SEP rather than having to define one each time. SEPs thus become reusable descriptions of the software development activity.

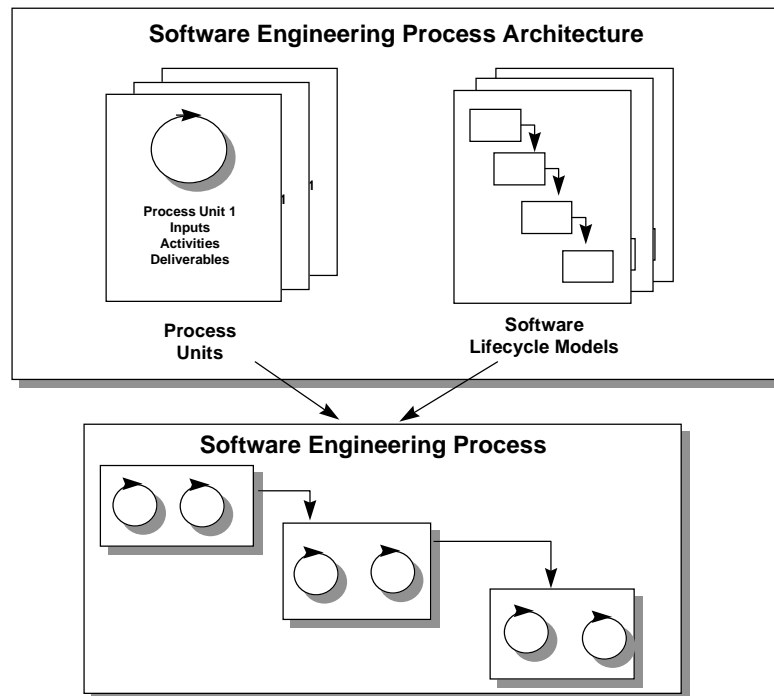


Figure 11 The relationship between process units, software lifecycle models and a software engineering process.

Process MeNtOR provides three software engineering processes known as:

- M-SEP1 - Large Scale Software Development Process
- M-SEP2 - Small Scale Software Development Process
- M-SEP3 - Package Procurement Process

Software projects should follow one of these SEPs to develop software. The SEPs define the sequence of activities, the major milestones and the deliverables to be produced by projects. The SEPs uses the process units to provide the detail on the techniques and activities to be performed during the project.

7.1 An Example SEP

Within *Process MeNtOR* one predefined software engineering process (M-SEP1) is provided that consists of three related software lifecycles.

The three related software lifecycle are termed the:

- *Programme Lifecycle*

- *Product Lifecycle*
- *Project Lifecycle*

The relationship between these software lifecycles is shown in *Figure 12*. Essentially the Programme Lifecycle consists of a set of projects, each of which undergoes the Project Lifecycle. A Project Lifecycle maybe initiated to develop a major new version, including creating a new software system (both known as *Major Software Releases*), or to develop an enhancement to an existing software system (known as a *Minor Software Release*), or to rectify defects in the software system (known as a *Patch Software Release*). These different types of development are modelled as part of the Product Lifecycle.

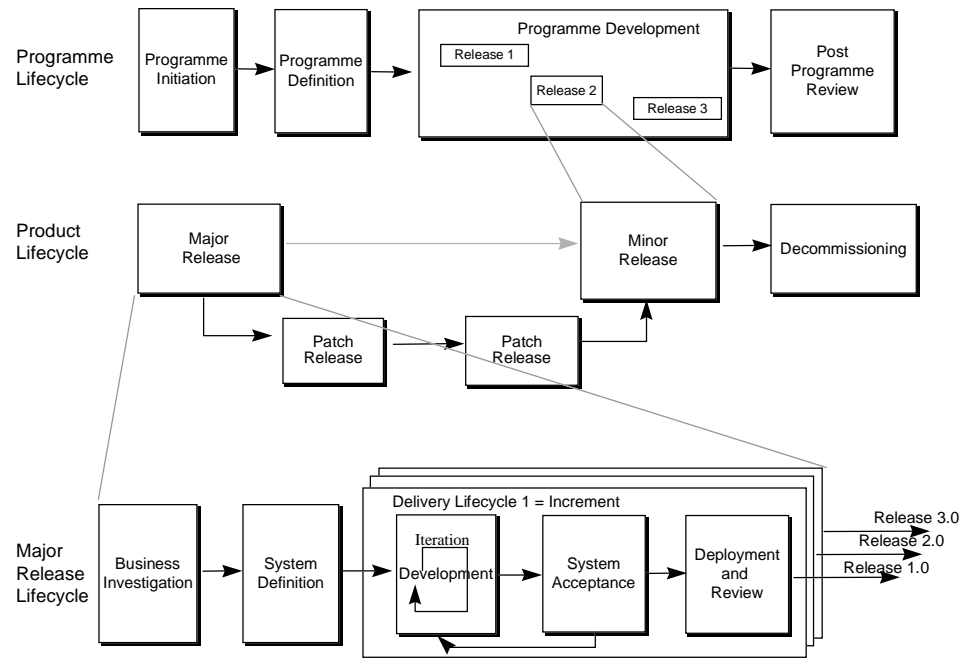


Figure 12 Software lifecycles of Process MeNtOR's predefined software engineering process (M-SEP1).

7.2 Deliverables

Information generated during the execution of the software lifecycles described above is placed into a deliverable. These deliverables are then organised into a series of *workbooks* (see *Figure 13*). Workbooks provide a default structure for the deliverables of the processes used in a software lifecycle. In the same way that a software lifecycle may be customised, so too can the workbooks.

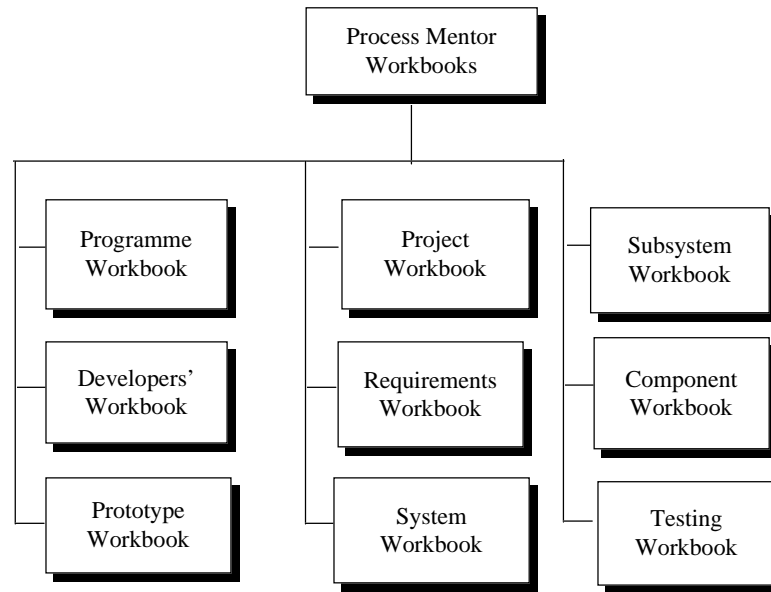


Figure 13 A typical Workbook Structure.

Process MeNtOR provides a set of default templates for each deliverable produced during the execution of a process. These deliverables may also have a set of release states.

8 Analysis and Design Concepts

This section introduces the main analysis and design concepts used within *Process MeNtOR*. This section is not intended as a rigorous definition of these concepts or the process units that use them. Instead, it provides a broad introduction to enable the reader to familiarise themselves with the main ideas used in *Process MeNtOR*. This section assumes some familiarity with object-oriented concepts. For more information on object-oriented concepts and techniques, the reader is referred to the references at the end of this paper.

8.1 Relationship to UML and OO Methods

Process MeNtOR is a software development *process* that supports and incorporates a number of OO *methods*. (A process is defined as the full set of activities and tasks required to develop an industrial strength application. A method is defined as the concepts and notation used by a process to express designs). *Process MeNtOR* is built on top of the Object Management Group (OMG) industry standard UML (Unified Modelling Language). This means *Process MeNtOR* supports recognised industry standards and can therefore be used with a range of different tools.

A number of other OO methods are able to be used with *Process MeNtOR* including Booch (Booch, 1994), OOSE (Jacobson et al., 1994), Responsibility Driven Design (RDD) (Wirfs-Brock et al., 1990), the Object Modelling Technique (OMT) (Rumbaugh et al., 1991) and MOSES (Henderson-Sellers and Edwards, 1994), although most of these have themselves been superseded by UML.

8.2 Scenarios

A scenario is the key mechanism within *Process MeNtOR* for analysts, designers and business users to explore and record interactions with a (software) system.

A *scenario* is simply a textual description of an interaction with a system. There are two types of scenario used in the *Process MeNtOR* processes - those focused on the problem and those focused on the solution. Scenarios focused on the problem that the *business* is trying to solve are called *Business Scenarios*. Scenarios focused on the *solution* provided by a particular system are called *System Scenarios* (Figure 14). In general, scenarios should be understandable by all members of the project team, including the business users.

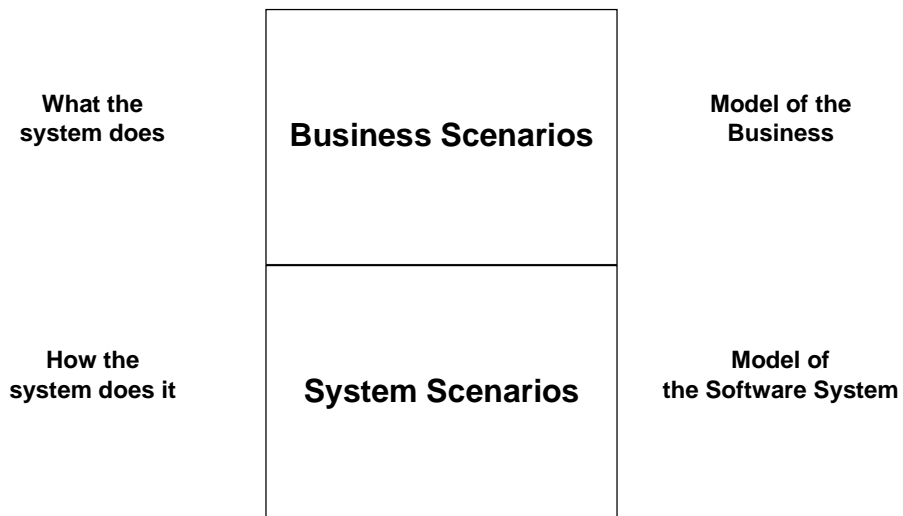


Figure 14 Business Scenarios focus on the business requirements, System Scenarios focus on the software solution.

Scenarios are supported by a graphical scenario diagram which is a graphical representation of the scenarios and the relationships both internally and externally. *Figure 15* shows an example of a graphical scenario diagram.

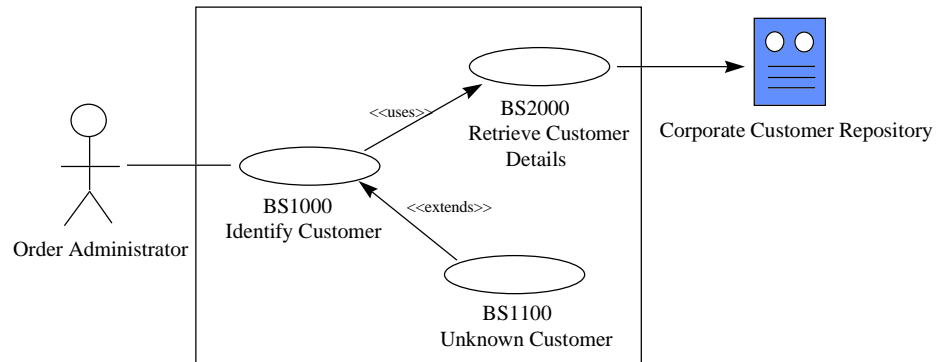


Figure 15 Graphical Scenario Diagram.

Business scenarios are identified in the Requirements Modelling process, usually through workshop sessions with business users. Business scenarios are often refined following feedback from the Prototyping process and after further workshops. Refining business scenarios through the involvement of business users is a key aspect of the Requirements Modelling process. In general, business scenarios should be independent of the underlying technology and software system being used to implement them. Business scenarios thus describe WHAT the system must do. Business scenarios are documented in the Requirements Workbook.

System scenarios refine and transform the business scenarios to a particular solution. They focus on HOW the software system will achieve the business scenarios given particular technology and other constraints (such as budget and time constraints). System scenarios are developed in the System Modelling process by the Architecture Team. System scenarios are documented in the System Workbook.

8.3 Classes and Relationships

Object-oriented development uses the concept of a class as its fundamental unit of analysis, design and implementation.

A class is a model which captures the commonalities of a set of entities in a domain. A class captures both the information (state or attributes) about the concept as well as the behaviour (operations or functions) of the concept. For example, a model of a bank *account* in an object-oriented system would model the fact that an *account* has information in it such as an *account number*, a *creation date* and an *owner*. It would also model the fact that a bank *account* can be *opened* and *closed* and money *withdrawn* and money *added* to an account (i.e. functions that can be performed on an account).

A class may model a concept in the problem domain (such as bank accounts) or in the solution domain (such as windows in a graphical user interface). Classes can be used during analysis, to develop logical models, and during design/implementation to develop executable systems.

Classes are related to one another by relationships. Relationships between classes include association, aggregation and inheritance. Association is when one class knows about another, for example, a bank account knows about its owner. Aggregation is when one class is composed of another, for example, a car is composed of a door and chassis. Inheritance is when one class is a special type of another class, for example, a cheque account is a special type of bank account. These relationships are used to model the structure of classes in a system. The class structure is recorded as a class diagram (*Figure 16*).

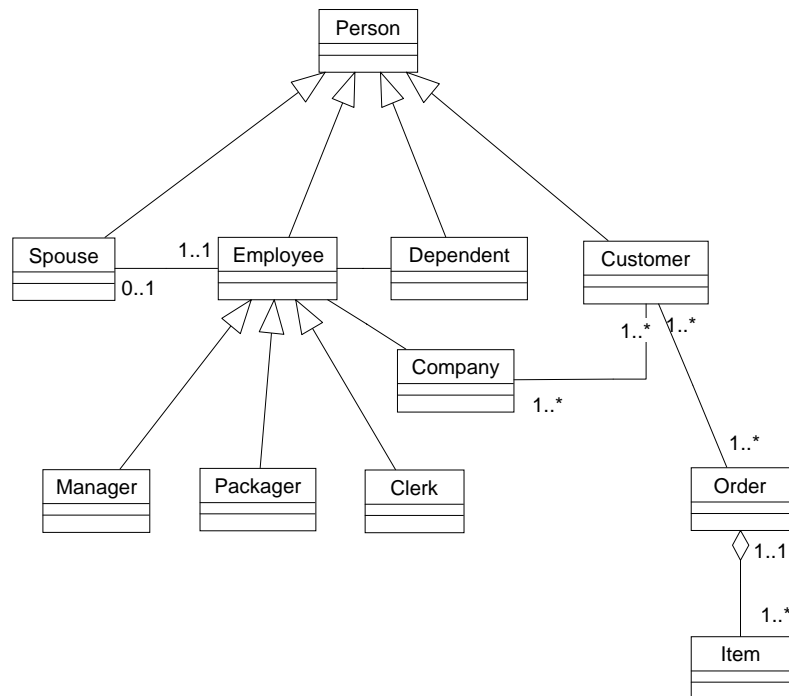


Figure 16 Class Diagram using UML notation.

Classes implement the scenarios of a system by sending messages to one another. These messages ask other classes to perform operations (functions) for them. Message passing is shown in object scenario diagrams or interaction diagrams (*Figure 17*).

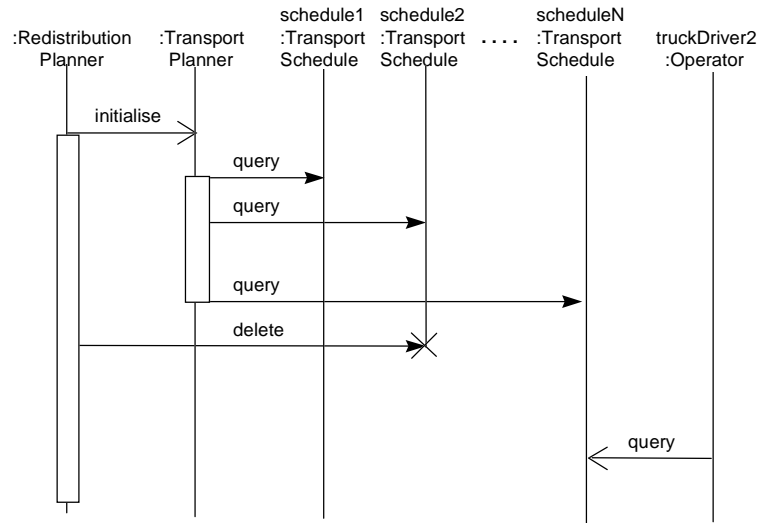


Figure 17 An Interaction Diagram using UML notation.

Scenarios are transformed into class diagrams and interaction diagrams through techniques such as CRC (Class Responsibility Collaboration) workshops during the Requirements Modelling, System Modelling and Component Modelling processes. These workshops allocate operations (pieces of a scenario) to different classes used in the scenario. Through message passing, classes are able to implement the system scenarios.

These models are developed in a high-level form during the early periods of a project and are refined to more detailed design models as the project progresses. An important point is that the same concepts apply throughout analysis, design and implementation. Thus a class identified in analysis can also appear in design and implementation.

The model and deliverables therefore tend to be refined rather than transformed during the software engineering process.

8.4 Architectures, Subsystems and Components

Classes are the fundamental units of decomposition within object-oriented development. However, as the number of classes in a system increases, it becomes necessary to partition the system into larger sized chunks. *Process MeNtOR* uses the concepts of System, Subsystem, Components and Sub-Components which are documented in a System Architecture, Subsystem Model and Component Model respectively (a Sub-Component is documented in a refined Component Model) (Figure 18).

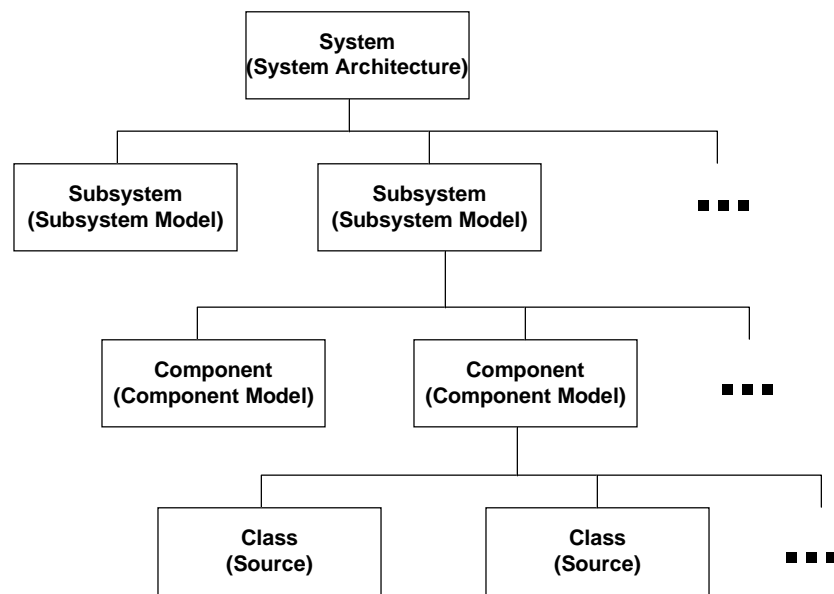


Figure 18 The decomposition structure of a large system.

The System Architecture plays a central role in object-oriented software systems. Indeed, if the software system is to have a long life, undergo extensive maintenance or expand into a product family, it is essential that an architecture be developed. (This is sometimes referred to as *architectural integrity*, or *architectural vision*.) An architecture provides the guiding direction for the overall software system in terms of development philosophy and subsystem decomposition. The software system architecture is constrained by the overall Programme Architecture if the software system is part of a larger programme of work.

Subsystems are the units of decomposition below the architecture. Subsystems encapsulate and isolate different parts of a system, for example, separating the database layer from the graphical user interface layer from the business model layer. From a management perspective, subsystems are units of team development with one team often responsible for a single subsystem. This subsystem will then be designed and specified according to the requirements of the System Architecture. Hence, each subsystem team can essentially work independently of other subsystems as far as the internal design is concerned. Each subsystem has its own design document, called a Subsystem Model which is part of the Subsystem Workbook.

Components are the unit of decomposition below a subsystem and are the lowest level unit before a class. A component is of the order of three to ten classes and is usually the responsibility of a single developer. It has a set of responsibilities which must be met, the design of which is the responsibility of the developer. The degree to which developers are permitted to design their own components is a function of their experience and importance of the component to the overall system. Components are documented in Component Specifications and Component Designs that are stored as part of the Component Workbook.

9 Deploying *Process MeNtOR*

9.1 Deployment Lifecycle

Deploying *Process MeNtOR* in your organisation or project for the first time requires a planned programme of training, workshops and mentoring. Described below is a generic model for deploying *Process MeNtOR* (see *Figure 19*). This model is normally tailored to address the specific needs and requirements of the project team members. Typically the people involved in the deployment planning are the Project Managers, Software Development Managers, Process Engineers and Training Managers.

The Deployment activity is typically divided into four phases:

- 1• Project Establishment Phase
- 2• Project Driving Phase
- 3• Project Review Phase
- 4• Support Phase

The following diagram illustrates these four phases, and the processes used during this deployment.

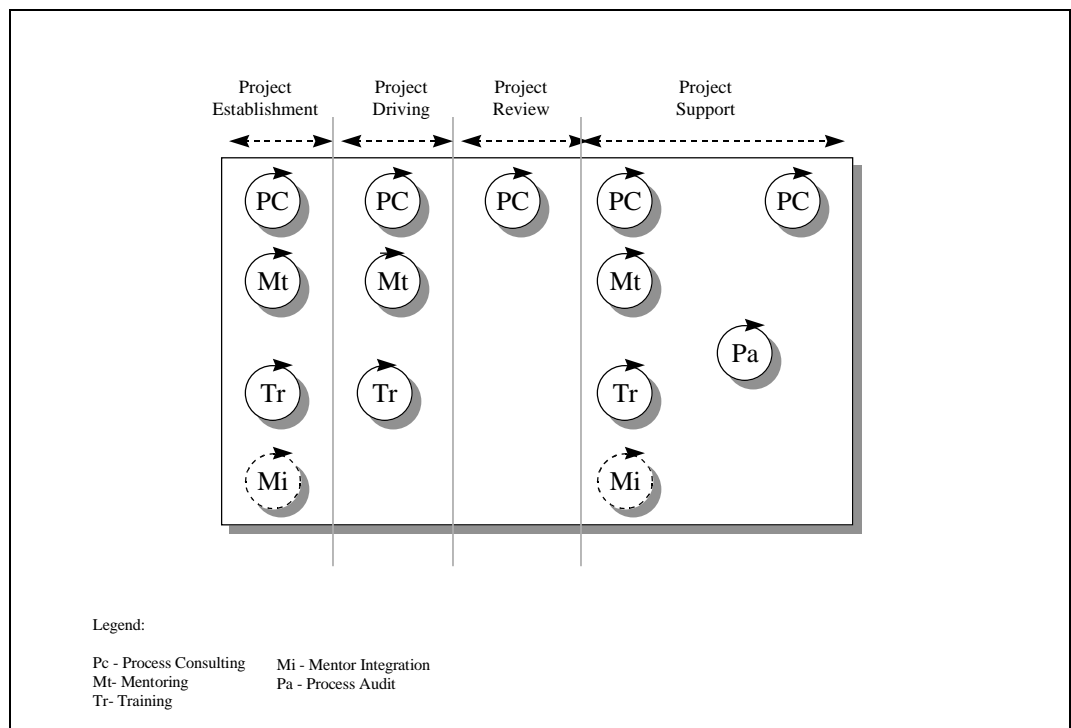


Figure 19 Typical Lifecycle for Deploying *Process MeNtOR*

9.1.1 Project Establishment Stage

This stage involves the initial deployment of *Process MeNtOR* into the project team. This involves setting up the *Process MeNtOR* material, installing the software products and undertaking the initial planning sessions. This is usually done by the Project Manager, Team Leaders and Process Consultant.

During this phase training plans are developed, training and workshops undertaken and *Process MeNtOR* integrated with the other processes and tools. Activities include:

- 1• Deploying *Process MeNtOR* material
 - Provide Manuals to staff
 - Deploy Online copy and training
 - Install templates and tailor them to the project
- 2• Defining a project Software Engineering Process (SEP) or selecting a SEP
- 3• Planning the process automation
- 4• Training
 - Define Training Plan (formal courses and workshops)
 - Undertake Training Plan
- 5• Coaching
 - Bring project team up to speed in processes
- 6• Undertaking the tool integration.

9.1.2 Project Driving Stage

The project driving stage involves leading the project team through all aspects of *Process MeNtOR*. Activities include:

- 1• Coaching in *Process MeNtOR*
 - Assign coaches to key teams
- 2• Training
 - Undertake Training Courses
 - Undertake Workshops.

9.1.3 Project Review Stage

The Project Review stage occurs once the project is complete. Activities include:

- 1• Review project and the application of *Process MeNtOR*
- 2• Define process improvement activities.

9.1.4 Support Phase

Once *Process MeNtOR* has been successfully deployed within the organisation, the deployment moves onto the Support phase. This phase promotes:

- 1• Periodic audits of the process being conducted (through the *Process Audit (Pa)*)
- 2• A reduced requirement for dedicated coaching services
- 3• The support of an ongoing training programme
- 4• Product upgrades.

9.2 Training and Coaching

Supporting the above deployment lifecycle is a complete training and coaching programme.

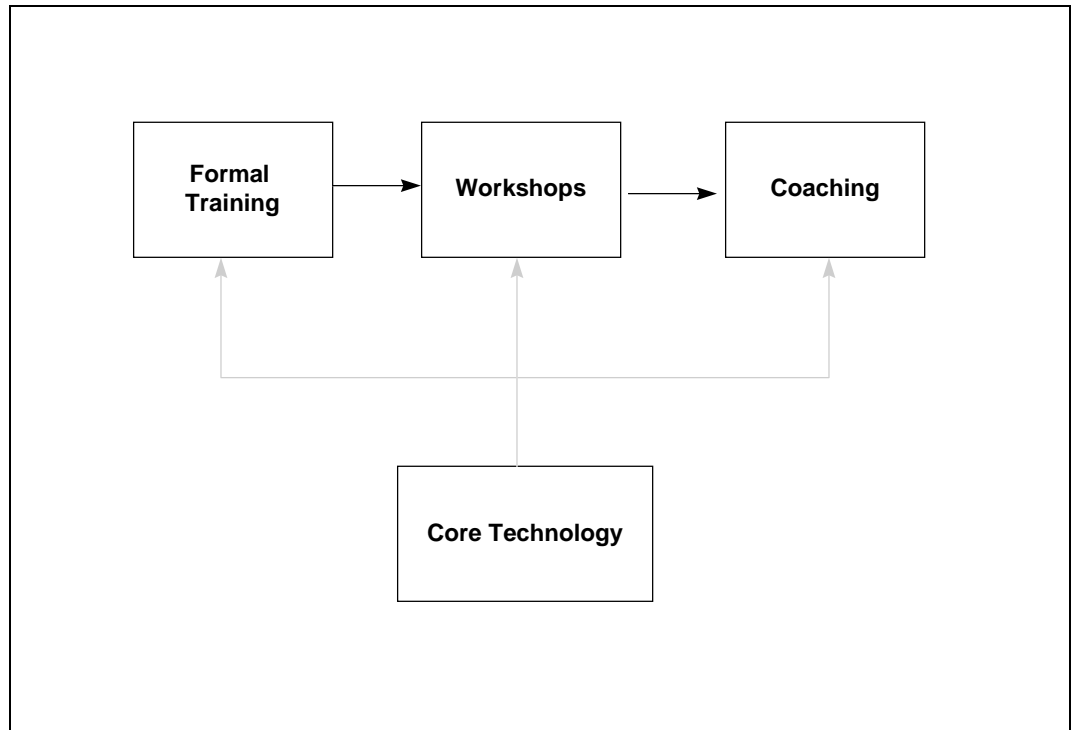


Figure 20 Steps in the Skills Development Process.

9.2.1 Formal Training

An essential first step in adopting any new software process is participation in formal training courses. In the case of *Process MeNtOR*, these include formal training in the concepts, principles and process units of *Process MeNtOR*. Such formal training is most effective when it is delivered by experienced presenters. The standard *Process MeNtOR* courses include:

- 1• *Process MeNtOR* Overview
- 2• Requirements Modelling
- 3• System and Subsystem Modelling
- 4• Component Modelling
- 5• Testing.

However, as essential as formal training is, it is only a start and may result in participants acquiring as little as 10 per cent of the knowledge and skills they will ultimately require to become truly proficient in the approach.

9.2.2 Workshops

The best way to consolidate the knowledge obtained from formal training courses and to teach new skills based upon that knowledge, is to have people attend one or more workshops as soon as possible after they complete their formal training. Workshops are organised around real life situations, such as the design of a particular system, and can contribute as much as another 30 per cent of the knowledge and skills that will ultimately be required to become truly proficient in the technology. Because of the very practical hands-on nature of workshops, it is essential that the presenter of a workshop be a very experienced practitioner.

9.2.3 Coaching

The most intensive and productive learning process occurs when people who have participated in formal training and workshops get the chance to conduct a real project. The project should be carried out under the supervision of an experienced coach/mentor. This coach is responsible for the design and development of strategic parts of the project and advises and assists the project by their actions. The work of this experienced mentor is termed *mentoring* or *coaching*. It is during this period, under the direct supervision of a consultant, that people acquire the bulk (as much as 60%) of the knowledge and skills that they require to become truly proficient in the approach.

10 Discussion

Process MeNtOR supports not just a project using OT but also the organisation. It is able to achieve this by using a software engineering process architecture. A software engineering process architecture permits different project types and changing organisation factors to be managed in a consistent fashion across an organisation, whilst also maintaining flexibility for the organisation procedures.

As an example of the way in which *Process MeNtOR* may be used, imagine that an organisation, which normally develops medium-sized software systems utilising a waterfall software lifecycle model, decides to undertake the development of a small packaged product. This new project has a different set of characteristics to those normally undertaken by the organisation. Therefore, this new project has different requirements for its software engineering process.

Process MeNtOR enables the organisation to develop an appropriate software engineering process for that project, assuming the organisation does not already have a software engineering process defined for such a project type. Developing such a software engineering process will require the selection of a software lifecycle model and the mapping of software processes to the selected software lifecycle model. Development of a particular software engineering process would follow the activities and guidelines of the software lifecycle model process, taking into account the experience of the organisation from previous software projects.

Once a software engineering process for this type of project has been defined, it can be named, refined over a number of projects and stored as part of the organisation's software engineering process library. Future projects of the same type can then use this process knowing it has been tested and refined.

Adopting *Process MeNtOR* requires a planned programme to transition the organisation to the new approach. This transition may or may not be part of a broader technology transitioning process depending upon whether an organisation is adopting object technology at the same time as they are adopting *Process MeNtOR*.

Hence, *Process MeNtOR* is able to offer an organisation a disciplined approach to object-oriented software development, whilst ensuring that this approach can remain open and flexible.

11 Bibliography

References cited in the paper are listed in this section together with a bibliography on object-oriented software development.

- Adams, S. and Burbeck, S., 1992, Software assets by design, *Object Magazine*, 2(4).
- Basili, V. and Turner, A.J., 1975, Iterative enhancement: a practical technique for software development, *IEEE Trans. Software Eng.*, BE-1(4), 390–396
- Beck, K. and Cunningham, W., 1989, A laboratory for teaching object-oriented thinking, *OOPSLA '89, SIGPLAN Notices*, 24(10), 1–6
- Boehm, B.W., 1986, A spiral model of software development and enhancement, *ACM Software Engineering Notes*, 11(4), 14–24
- Booch, G., 1994, *Object-oriented Analysis and Design with Applications*, Benjamin Cummings, 2nd Ed.
- Chidamber, S. and Kemerer, C., 1991, Towards a metric suite for object-oriented design, in *Proc. OOPSLA'91, Sigplan Notices*, 26(11), 197–211
- Coad, P. and Yourdon, E., 1991, *Object-Oriented Analysis*, 2nd edition, Prentice-Hall, 233pp
- Cox, B.J., 1986, *Object Oriented Programming: An Evolutionary Approach*, Addison-Wesley, Reading, MA, 274pp
- Firesmith, D.G., 1993, *Object-Oriented Requirements Analysis and Logical Design: A Software Engineering Approach*, J. Wiley and Sons, New York, 575pp(in press)
- Graham, I., 1991, *Object-Oriented Methods*, Addison-Wesley, Wokingham, UK, 410pp
- Henderson-Sellers, B. and Edwards, J., 1994, *BOOKTWO of Object Oriented Knowledge: The Working Object*, Prentice-Hall, pp594.
- Henderson-Sellers, B. and Edwards, J.M., 1990, The object-oriented systems life cycle, *CACM*, 33(9), 142–159
- Humphrey, W., 1989, *Managing the Software Process*, Addison-Wesley, Reading, MA
- Jacobson, I., Christerson, M., Jonsson, P. and Overgaard, G., 1992, *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison Wesley, 524pp
- Lorenz, M., 1993, *Object-Oriented Software Development: A Practical Guide*, Prentice Hall, New Jersey, 227pp
- Meyer, B., 1988, *Object-oriented Software Construction*, Prentice Hall, Cambridge, 534pp
- Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F. And Lorensen, W., 1991, *Object-oriented Modelling and Design*, Prentice Hall, New Jersey, 500pp
- Sommerville, I., 1989, *Software Engineering* (3rd edition), Addison–Wesley, Wokingham, UK, 653pp
- Stroustrup, B., 1991, *The C++ Programming Language*, (2nd edition), Addison-Wesley, Reading, MA, 328pp
- Wirfs-Brock, R.J., Wilkerson, B. and Wiener, L., 1990, *Designing Object-Oriented Software*, Prentice Hall, 368pp