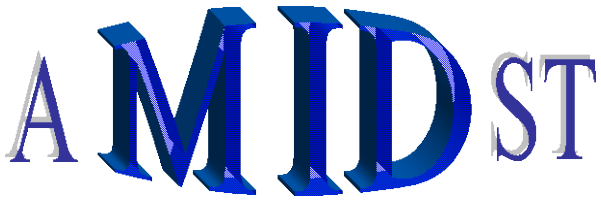

Modeling the Object-Oriented Software Process

OPEN and the Unified Process

REFERENCE	:	AMIDST/WP2/N002/V02
DATE OF ISSUE	:	February 5, 1999
ACCESS RIGHTS	:	public
STATUS	:	final
EDITOR	:	Klaas van den Berg
AUTHOR[S]	:	Mehmet Aksit, Klaas van den Berg, & Pim van den Broek

SYNOPSIS

Two major object-oriented process models are discussed: the OPEN model and the Unified Process model. Quality assurance in the latter model is assessed.

Document History

DATE	VERSION	MODIFICATION
November 26, 1998	01	Initial draft
February 5, 1999	02	Final draft

Abstract

A short introduction to software process modeling is presented particularly object-oriented models. Two major industrial process models are discussed: the OPEN model and the Unified Process model. In more detail, the quality assurance in the Unified Process tool (formally called Objectory) is reviewed.

Table of Contents

1. INTRODUCTION 1
1.1 ISO STANDARD SOFTWARE DEVELOPMENT PROCESS 1
1.2 OVERVIEW 3

2. MODELING SOFTW ARE PROCESSES 4
2.1 HISTORY 4
2.2 AIMS 4
2.3 ASPECTS 5
2.4 APPROACHES 5
2.5 ISSUES 5
2.6 OBJECT TECHNOLOGY 6
2.7 INDUSTRIAL SOFTWARE PROCESS MODELS 6

3. THE OPEN PROCESS 8
3.1 INTRODUCTION 8
3.1.1 Initial Planning and Development 8
3.1.2 Build Development 10
3.2 THE OPEN TOOLBOX 11

4. THE UNIFIED PROCESS - OBJECTORY 13
4.1 INTRODUCTION 13
4.1.1 Activities 13
4.2 SOFTWARE QUALITY 15
4.2.1 Introduction 15
4.2.2 Objectory 16
4.2.3 Evaluation 19

5. CONCLUSION 22

REFERENCES 23

1. Introduction

In the development of a software system usually several phases are identified. The main phases of the classical waterfall model are requirements analysis, specification, design, implementation, testing and maintenance. This model and alternative life cycle models such as prototyping and incremental development are described in most software engineering textbooks (Sommerville, 1996; Pressman, 1997; Pfleeger, 1998).

A software engineering development methodology comprises a conceptual framework, a notation for its models, and a process, i.e. a description of which activities have to be carried out in which order. Tools may assist the development process. For this purpose, models of the software process itself are developed. ISO-IEC provides a standard for software life cycle processes.

1.1 ISO Standard Software Development Process

The International Standard ISO-IEC 12207 (ISO/IEC 12207, 1995) establishes a common framework for software life cycle processes, with well-defined terminology, that can be referenced by the software industry. It contains processes, activities, and tasks that are to be applied during the acquisition of a system that contains software, a stand-alone software product, and software service and during the supple, development, operation, and maintenance of software products. Software includes the software portion of firmware.

This Standard groups the activities that may be performed during the life cycle of software into five primary processes, eight supporting processes, and four organisational processes. Each process is divided intro a set of activities; each activity is divided into a set of tasks.

The five **Primary Life Cycle Processes** are the following:

- *Acquisition*: Defines the activities of the acquirer, the organisation that acquires a system, software product, or software service.
- *Supply*: Defines the activities of the supplier, the organisation that provides the system, software product or software service to the acquirer.
- *Development*: Defines the activities of the developer, the organisation that defines and develops the software product.
- *Operation*: Defines the activities of the operator, the organisation that provides the service of operating a computer system in its live environment for its users.
- *Maintenance*: Defines the activities of the maintainer, the organisation that provides the service of maintaining the software product; that is, managing modifications to the software product to keep it current and in operational fitness. This process includes the migration and retirement of the software product.

The eight **Supporting Life Cycle Processes** are the following:

- *Documentation*: Defines the activities for recording the information produced by a life cycle process.

- *Configuration management*: Defines the configuration management activities.

- *Quality*: Defines the activities for objectively assuring that the software Assurance products and processes are in conformance with their specified requirements and adhere to their established plans. Joint reviews, Audits, Verification and Validation may be used as techniques of Quality Assurance.

- *Verification*: Defines the activities (for the acquirer, the supplier, or an independent party) for verifying the software products and services in varying depth depending on the software project.

- *Validation*: Defines the activities (for the acquirer, the supplier, or an independent party) for validating the software products of the software project.

- *Joint Review:* Defines the activities for evaluating the status and products of an activity. This process may be employed by any two parties, where one party (reviewing party) reviews another party (reviewed party) in a joint forum.

- *Audit:* Defines the activities for determining compliance with the requirements, plans, and contract. This process may be employed by any two parties, when one party (auditing party) audits the software products or activities of another party (audited party).

- *Problem:* Defines a process for analysing and removing the problems (including resolution nonconformances), whatever their nature or source, that are discovered during the execution of development, operation, maintenance, or other processes.

The four **Organisational Life Cycle Processes** are the following:

- *Management*: Defines the basic activities of the management, including project management, related to the execution of a life cycle process.

- *Infrastructure*: Defines the basic activities for establishing the underlying structure of a life cycle process.

- *Improvement*: Defines the basic activities that an organisation (that is, acquirer, supplier, developer, operator, maintainer, or the manager of another process) performs for establishing, measuring, controlling, and improving its life cycle process.

- *Training*: Defines the activities for providing adequately trained personnel.

As an example, we give the activities in the primary life cycle process *Development*. The 13 activities with related tasks are the following:

1. *Process Implementation*. The tasks are: Define software life cycle model, Document and control outputs, Select and use standards, tools, languages, Document development plans, Deliver all needed products

2. *System requirements analysis*. The tasks are: Specify system requirements, Evaluate requirements against criteria

3. *System architectural design*. The tasks are: Establish top-level architecture, Evaluate architecture against criteria

4. *Software requirements analysis*. The tasks are: Document software requirements, Evaluate requirements against criteria, Conduct joint reviews

5. *Software architectural design*. The tasks are: Transform requirements into architecture, Document top-level design for interfaces, Document top-level design for database, Document preliminary user documentation, Document preliminary test requirements, Evaluate architecture against criteria, Conduct joint reviews

6. *Software detailed design*. The tasks are: Document design for each component, Document design for interfaces, Document design for database, Update user documentation, Document unit test requirements, Update integration test requirements, Evaluate detailed design against criteria, Conduct joint reviews

7. *Software coding and testing*. The tasks are: Document each unit, database and tests, Conduct and document unit testing, Update user documentation, Update integration test requirements, Evaluate code and test results

8. *Software integration*. The tasks are: Document integration plans, Conduct and document integration tests, Update user documentation, Document qualification tests, Evaluate plans and tests against criteria, Conduct joint reviews

9. *Software qualification testing*. The tasks are: Conduct and document qualification testing, Update user documentation, Evaluate tests against criteria, Support audits, Prepare product for next phase

10. *System integration*. The tasks are: Integrate software with hardware & others, Document integration tests, Evaluate integrated system against criteria

11. *System qualification testing*. The tasks are: Conduct and document qualification tests. Evaluate system against criteria. Support audits. Prepare product for installation

12. *Software installation*. The tasks are: Plan installation in target environment. Install software

13. *Software acceptance*. The tasks are: Support acquirer's acceptance tests. Deliver product per contract, Provide training per support contract

Full software life cycle support should cover the processes, the activities and the tasks listed in this standard.

1.2 Overview

In this document we discuss two major software process models, the OPEN model and the Objectory model (recently renamed the Unified Process). In section 2 we start with a short overview on software process modeling. In section 3 we discuss the OPEN-model and the OPEN-toolbox. In section 4 we present the Unified Process model and we review the quality assurance in the Objectory tool.

2. Modeling Software Processes

In this section we give a short history of process modeling, the aims of process modeling. Then we present some aspects and approaches to software process modeling. We conclude this section with some research issues in this field, the prospects of object technology, and finally we introduce two major object-oriented process models.

2.1 History

Software process modeling has relatively a short tradition. Nowadays, there are several workshops and conferences being held yearly. The first international workshop ISPW-1 was held at Runnymede, UK, in 1984 (ISWP-1, 1984). At ISPW-6 in 1990 (Kellner *et al.*, 1990), an example was presented of a software process, which is being used as baseline for the comparison of software process models and environments. The first international conference on software process modeling, ICSP-1, was held at Redondo Beach, CA, USA, in 1991 (ICSP-1, 1991). European workshops on software process technology started at Milan, Italy, with EWSPT-1, also in 1991 (EWSPT-1, 1991). Conferences on software process improvement SPI are closely related to the area of process modeling. At the International Conference on Software Engineering ICSE, there are also regularly contributions on process modeling topics. Some special issues of magazines were devoted to this area (IEEE-SE, 1993; SEJ, 1991).

Although there is no long research tradition in this area, there is already a vast amount of literature available and many research groups are active[1]. In Europe, the PROMOTOR Working Group co-ordinated several projects (Finkelstein *et al.*, 1994). We will briefly mention some modeling issues, which focuses on object technology in software process modeling (see also Fugetta & Wolf, 1996).

2.2 Aims

Curtis (Curtis *et al.*, 1992) gives the following overview with objectives and goals of software process modeling:

- Facilitate human understanding and communication: requires that a group is able to share a common representational format
- Support process improvement: requires a basis for defining and analysing processes
- Support process management: requires a defined process against which actual process behaviours can be compared
- Automatic guidance in performing process: requires automated tools for manipulating process descriptions
- Automatic execution support: requires a computational basis for controlling behaviour within an automated environment.

[1] E.g. Software Process Research Sites, at http://hamlet.cogsci.umassd.edu/SWPI/1docs/SPResearch.html

One can focus on one or more of these objectives in process modeling research. Obviously, automated process enactment requires a more detailed and formalised model then a model just aiming at human understanding the process.

2.3 Aspects

The goal of process improvement has been incorporated in the SEI Software Maturity Model (Paulk *et al.*, 1993). Several key process areas have been identified (Paulk *et al.*, 1993b). These areas can be classified in addressing the following categories or aspects:

- Managerial, such as project planning, subcontract management.
- Organisational, i.e. process definition, change management.
- Engineering aspects: requirement analysis, design, coding, testing, etc.

It is clear that these aspects are present in software process modeling in general. This is obvious in the reference model for process technology presented by Christie (Christie *et al.*, 1996). They distinguish four main elements: the enterprise operations, the process development, the enactment technology and the process assets. The enterprise operations deal with the process effectiveness of organisations, and the technology that exists to support that effectiveness. The process development deals with the construction of process technology assets. These assets support the organisational process activities. The enactment technology deals with the technology components that need to be in place for the construction of effective process enactment systems. The process assets are the parts in the process, which have to be designed for reuse and placed in an asset library.

2.4 Approaches

Basic concepts in software process modeling are the artifacts (i.e. the (sub-) products) and activities or process steps, which produces externally visible state changes to the artifacts (Conradi *et al.*, 1994; Feiler & Humphrey, 1992). Another important concept is the meta-process: the part of the process in charge of maintaining and evolving the whole process, i.e. the production process, its meta-process, and the process support.

As in traditional modeling techniques, one can focus on the data in the process, the artifacts, or on the transformation functions, the activities. Products and process are dual entities. Some approaches in software modeling are process-centred and other approaches are product-centred. The relative merits of both approaches may become apparent in applying the approach to the Software Process Example (Kellner *et al.*, 1990).

2.5 Issues

Two of the key process areas in the capability maturity model at level four are the Quantitative Process Management, aiming at controlling the process performance quantitatively, and Software Quality Management, aiming at a quantitative understanding of the quality of the software products (Paulk *et al.*, 1993b). This means that attributes of process and product entities have to be identified and measures have to be defined and validated. However, many attributes are inherently uncertain (Huff, 1996). Moreover, there are many ambiguities in the process steps. The quantitative support has to cope with uncertainty and ambiguity.

Software products are evolving rapidly due to changing requirements, requiring a high adaptability of the products and composability of solutions. The productivity in software development heavily relies on the reusability of products and subproducts on all levels of the development: not only reuse of code, but also of design (among other by design patterns), frameworks and software architectures.

2.6 Object technology

Object technology has been used in process modeling at various levels. In an assessment of project within the PROMOTOR Working Group (Finkelstein *et al.*, 1994) it appears that object technology is used in the modeling phase of the basic components, in process modeling languages, in supporting tools and enactment engines, and in meta-processes. Object technology has some obvious advantages in process modeling as discussed by Aliee and Warboys (Aliee & Warboys, 1995). Objects provide structural and behavioural views of the system architecture; they provide reusability and encapsulation in design methods, and concurrency in complex systems.

2.7 Industrial Software Process Models

We will focus now on the object-oriented process modeling and support of the software development process. There are two important approaches:

- OPEN, i.e. Object-oriented Process, Environment, and Notation. This methodology has been developed by Graham, Henderson-Sellers, Firesmith and others (Graham, Henderson-Sellers & Younessi, 1997). As modeling language in this approach OML (Open Modeling Language) is used (Firesmith *et al.*, 1997). OML consists of COMN (Common Object Modeling Notation) and a metamodel that describes the individual language models and their relationships. The object-oriented techniques used in this process are described in the OPEN Toolbox of Techniques (Henderson-Sellers, Simons & Younessi, 1998).

- Objectory / Rational Unified Process. The Objectory process has been developed by Jacobson (Jacobson *et al.*, 1992) and has been taken over by Rational. It is now named the Unified Process (Jacobson, Booch & Rumbaugh, 1999). It uses UML (Unified Modeling Language) as its modeling language (Booch, Rumbaugh & Jacobson, 1999).

The OPEN process uses the OML as modeling language, whereas the Unified Process/Objectory uses the UML modeling language. A critical comparison of both languages is given by Prasse (Prasse, 1998). A modeling language consists of a concrete syntax (graphical and textual notations), and the abstract syntax and semantics (concepts, relations and interpretations). Furthermore, in the comparison are considered the application (perspectives, activities) and the description (language specification, metamodel) of the language. He uses the following criteria in the comparison:

- User-Relevant Criteria: Usability, Clarity, Understandability, Adequacy, Verification, Power

- Model-Relevant Criteria: Unambiguity, Consistency, Formalisation, Integration

- Economic Criteria: Reusability, Extensibility

Prasse concludes that clear improvements are made compared to earlier modeling approaches. OML and UML provide a more precise description of the language constructs and the integration of extension mechanisms. However, both approaches show clear inadequacies. There is still a lack of formality and correctness of descriptions of diagrams and language constructs. In particular, the semantics of object-oriented concepts is not complete. In addition, the large diversity of partly redundant kinds of diagrams complicates the application of both languages. In the following sections, we describe both process models in more detail.

3. The OPEN Process

In this section we present a short introduction to the OPEN process model. Then we discuss a major effort to categorise techniques in the area of object technology in the so-called OPEN Toolbox.

3.1 Introduction

OPEN was initially created as the merger of the following three object-oriented development methods: MOSES by Brian Henderson-Sellers, SOMA by Ian Graham, and The Firesmith Method by Donald Firesmith. OPEN's metamodel has been derived largely from the COMMA project by Brian Henderson-Sellers in 1995 which evaluated 14 previous OO development methods. Much of the impetus for OML was generated by the Object Management Group (OMG) Analysis and Design Task Force Request for Proposal (RFP) for an industry standard metamodel to support upperCASE tool interoperability.

The OPEN process is a generic framework, consisting of large numbers of potential major *activities* which are decomposed into smaller *tasks* which are performed using *techniques* that produce *products*. Because of the large number of potential activities, tasks, techniques, and products it provides, the OPEN Process must be instantiated by selecting the specific activities, tasks, techniques, and products best suited to meet the specific needs of the application and its development organization. The main activities are described below.

3.1.1 Initial Planning and Development

The *Initial Planning and Development* activity occurs once at the beginning of the development of the application. The architecture team prepares for the *Build Development* activity by performing the following tasks:

1. *Document the Background*. During this task, the background of the application including its history and overall purpose is identified and documented. The kind of application (e.g., MIS, embedded), the kind of project (e.g., initial development, update, replacement of one or more existing applications), and the experience level of the development organization are determined.

2. *Initial Training*. During this task, the architecture team is trained in the OPEN Method including process, modeling language, implementation language, guidelines, and metrics.

3. *Tailor the Method*. During this task, the architecture team selects the specific activities, tasks, techniques, and products to be developed from those available within the OPEN framework.

4. *Model the Context*. During this task, the context of the application is developed and documented using the following subtask and techniques: *External Object Identification* and *Semantic Modeling*.

5. *Capture the Initial Requirements*. During this task, the initial requirements are elicited and analyzed using the following techniques: *JAD/RAD Modeling, Interviews*, *Textual Modeling* and *Use Case Modeling*.

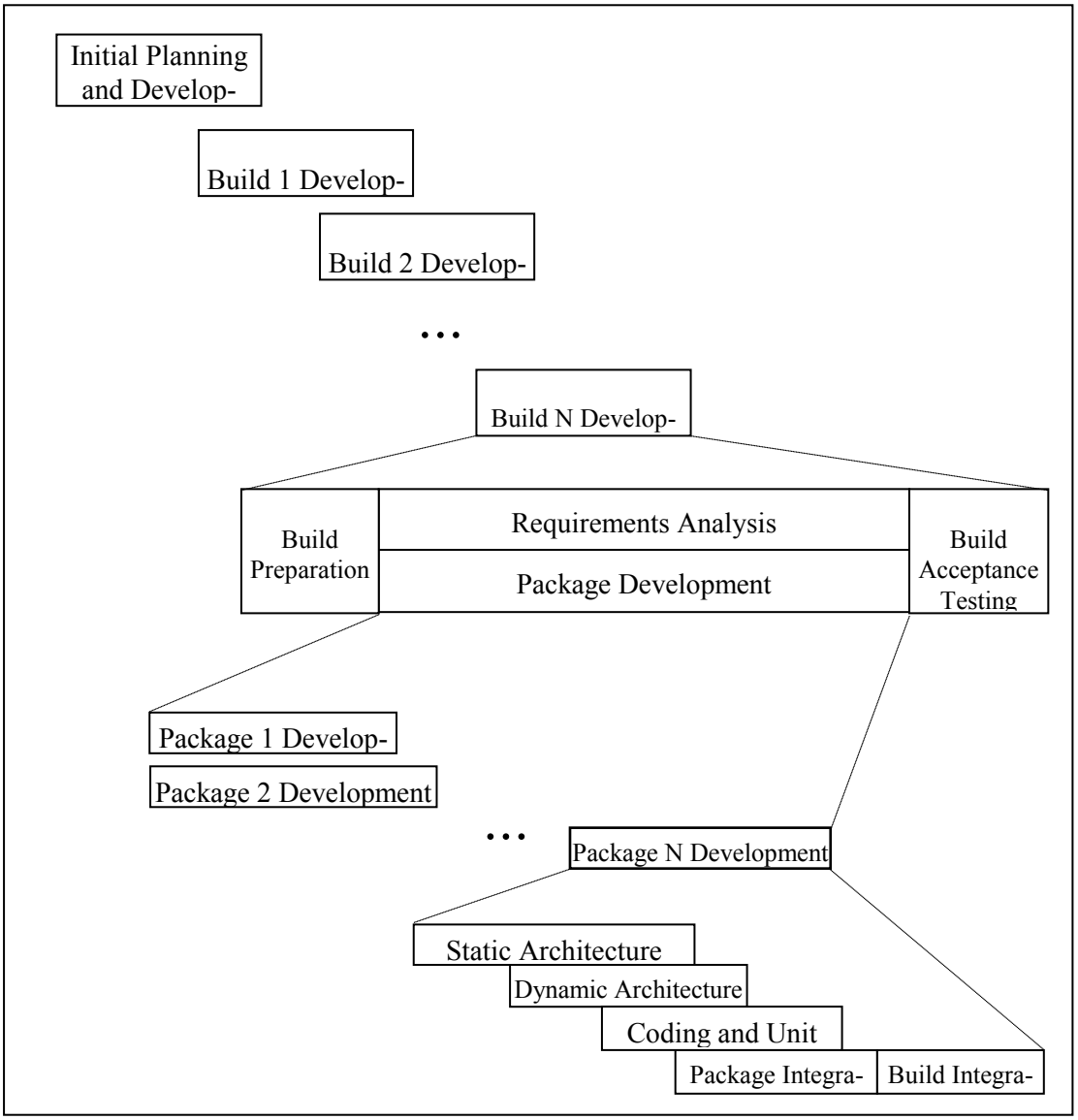

Figure 1: The OPEN Development Process (Firesmith et al.*, 1998)*

6. *Create the Initial Architecture*. The initial architecture forms the foundation for the following *Build Development*. During this task, an initial architecture of the application is developed and documented using the following subtasks and techniques: *Identify Key Abstractions*, *Assign Key Abstractions to Packages*, *Assign Packages to Layers*, *Semantic Modeling, Responsibility-Driven Design, and Interaction Modeling.*

7. *Document the Initial Products*. During this task, the initial requirements and architecture are specified and documented in the following product: *Requirements Specification and Design Document*.

8. *Scope the Application*. During this task, the scope of the application in terms of size (e.g., number of packages, number of classes) and required effort (e.g., number of

person weeks) is estimated based on the initial architecture and the estimated productivity of the development team.

9. *Plan Build Development*. During this task, the initial builds and releases of the application are scheduled and documented in the following product: *Build Plan*.

Although the above tasks are listed in the rough order in which they occur, there is a great amount of overlap and iteration among them. Due to the functional nature of use case modeling, use cases should be used more for verification purposes than for requirements analysis purposes.

3.1.2 Build Development

During the *Build Development* activity, the build development team performs the following tasks and subactivities:

1. *Build Preparation*. During this task, the build development team prepares to develop the build by performing the following subtasks: *Update the Relevant Requirements*, *Schedule the Build*, and *Update the Build Plan*.

2. *Requirements Analysis*. During the *Requirements Analysis* subactivity, the requirements team(s) perform the following tasks:

   - *Requirements Elicitation*. During this task, the relevant requirements are elicited using the following techniques: *JAD/RAD Sessions* and *Interviews*.

   - *Requirements Analysis and Specification*. During this task, the relevant requirements are analyzed and specified using the following techniques: *JAD/RAD Sessions*, *Textual Modeling*, *Use Case Modeling*, *Model Verification*.

3. *Package Development*. During the *Package Development* subactivity, the package teams develop, document, and integrate the packages in the build. The following tasks are performed in a highly iterative, incremental, and parallel manner:

   - *Identify the Packages*. This task is performed on an ongoing basis and consists of the following subtasks: *Identify Key Abstractions* and *Assign Key Abstractions to Packages*.

   - *Model the Package Static Architecture*. This task is performed on a package by package basis, using the following techniques: *Semantic Modeling* and *Responsibility-Driven Design.*

   - *Model the Package Dynamic Architecture*.
     Precondition: Model the Static Architecture has started.
     This task is performed on a package by package basis, using the following techniques: *Collaboration Modeling*, *Class Modeling* and *State Modeling*.

   - *Coding and Unit Testing*.
     Precondition: Model the Dynamic Architecture has started.

This task is performed on a package by package basis and consists of the following subtasks: *Class Coding* and *Class Testing*.

- *Package Integration*.
  Precondition: Coding and Unit Testing has started.
  This task is performed on a package by package basis, using the following technique: *Dependency-Based Testing*.

- *Build Integration*. During this task, the package is integrated into the growing build.

4. *Verify the Packages and Use Cases*. During this task, the consistency of the package and use case models is verified on an ongoing basis.

5. *Build Acceptance Testing*. During this task, acceptance testing of the build is performed using the following technique: *Use Case Testing*.

A complete application using the OPEN process has been described by Firesmith *et al*., 1998.

3.2 The OPEN Toolbox

In a companion book the techniques that can be used in the realisation of tasks in the activities of the software process have been described (Henderson-Sellers, Simons & Younessi, 1998).

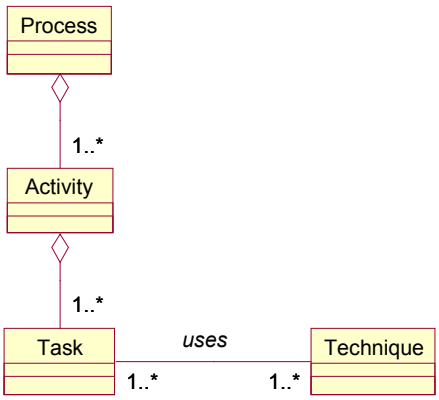

Figure 2. Activities have tasks, which are realized by the use of techniques (Henderson-Sellers, Simons & Younessi, 1998)

A two-dimensional matrix links the tasks (which provide the goals) to the techniques (which provide the way the goal can be achieved. In the OPEN Toolbox a survey is given of a large number of techniques, indicating the focus, typical tasks for which the technique is needed, related techniques, inputs (pre-conditions), and underpinning concepts. Moreover, a star rating is provided: *** well-tried, ** reasonably well validated, *** experimental or not well understood.

We consider the example of the technique Scenario development (p. 310). The star rating for this technique is **. The focus is dynamic modeling; the typical task for which this technique is needed is to construct the object model. A related technique is collaboration analysis. The

input is the user requirements. The underpinning concepts are scenarios, task scripts and use cases. The entry is concluded with a technique description and an outline of the technique usage.

Summary

In this section we discussed the OPEN software development process. OPEN covers the entire development process, which is one of the ISO-IEC primary life cycle processes. OPEN covers all major activities and tasks in this process, including initial requirements elicitation, analysis, and specification, logical and physical design, implementation, and testing. OPEN emphasizes an object-oriented mindset and is responsibility-driven rather than use-case-driven or data-driven.

4. The Unified Process - Objectory

In this section we present a short introduction to the Objectory process model. Then we discuss how software quality is aimed at in the Objectory tool.

4.1 Introduction

In the documentation of the CASE tool, the Rational Objectory Process is described as a full lifecycle software engineering process. It is a controlled iterative process, with strong focus on architecture. It is a use-case driven, object-oriented process, using the Unified Modeling Language (UML) as a notation for its models (Rational, 1997).

The Rational Objectory Process is described in two dimensions. The first dimension *time* represents the dynamic aspect of the process, as it is enacted, and is expressed in terms of cycles, phases, iterations and milestones. The software lifecycle is broken into cycles, each cycle working on a new generation of the product. The Objectory process divides one development cycle in four consecutive phases: the inception phase, elaboration phase, construction phase, and transition phase. The second dimension *process components* represents the static aspect of the process: how it is described in terms of process components, activities, workflows, artifacts, and workers. The Objectory process is composed of seven process components. There are four engineering process components: requirement capture, analysis and design, implementation and test, and three supporting components: management, deployment, and environment.

4.1.1 Activities

The activities in the engineering process components are the following:

1. Requirement Capture:
    - Capture a Common Vocabulary
    - Find Use Cases and Actors
    - Describe the Use-Case Model
    - Prioritize Use Cases
    - Describe a Use Case
    - Structure the Use-Case Model
    - Review the Use-Case Model
2. Analysis & Design
    - Architectural Analysis

- Architectural Design
- Describe Concurrency
- Describe Distribution
- Review the Architecture
- Use-Case Analysis
- Object Analysis
- Review the Analysis
- Use-Case Design
- Object Design
- Review the Design

3. Implementation
    - Define the Organization of Subsystems
    - Plan System Integration
    - Plan Subsystem Integration
    - Implement Classes
    - Fix a Defect
    - Perform Unit Test
    - Review Code
    - Integrate Subsystem
    - Integrate System
4. Test
    - Plan Test
    - Design Test
    - Implement Test
    - Design Test Packages and Classes
    - Implement Test Subsystems and Components
    - Execute Integration Test

- Execute System Test
- Evaluate Test

For each activity, the purpose, the workers, the input artifacts and the output artifacts are described. As an example, we consider the activity: Find Use Cases and Actors.

- The *purpose* of this activity is: To outline the functionality of the system. To define what will be handled by the system and what will be handled outside the system. To define who and what will interact with the system.
- The *worker* is the Use Case Model Architect.
- The *input artifacts* are: Requirements, Glossary, Vision, Use-Case Modeling Guidelines.
- The *output artifacts* are: Use Case Model, Use Cases, Glossary, Supplementary Specifications
- For this activity the following *steps* have to be performed: Find Actors, Find Use Cases, Describe How Actors and Use Cases interact, Evaluate Your Results.

Summary

In this section we described the Objectory/Unified Process software development process. Objectory covers the entire development process, which is one of the ISO-IEC Primary Life Cycle processes. Objectory emphasizes a use-case centered approach. Furthermore, some of the Supporting Life Cycle processes are provided for, such as the processes Reviews and Quality. We now consider in more detail how software quality is achieved in the Objectory process.

4.2 Software Quality

4.2.1 Introduction

Software quality assurance ought to be an important component of automated software engineering. We discuss how software quality assurance is realised in the Rational Objectory CASE tool. Although much support is given through guidelines and checkpoints, the tool fails to provide clear goals and metrics for quality assessments and it only partially supports the phases in a measurement program. Or to quote Booch: Quality software doesn't happen; rather, it's engineered that way (Booch, 1998)

One of the requirements for integrated CASE environments is the support of both management and technical metrics that can be used to improve the software process and the software artifacts (Forte, 1989). Measurement is an important constituent of software quality assurance as to reach the higher maturity levels in the Capability Maturity Model (CMM) (Paulk *et al.*, 1993). At level 4, the Managed Level, one of the key process areas (KPA) is Software Quality Management. The purpose of this KPA is to develop a quantitative understanding of the quality of the project's software products and achieve specific quality goals. Software Quality Management applies a comprehensive measurement program to the software products (as described in the KPA Software Process Engineering).

The Objectory CASE tool (Rational, 1998) supports the software development process as developed initially by Jacobson (Jacobson *et al.*, 1992). Objectory facilitates reaching CMM level 2 and 3 (Rational, 1997b). Objectory claims to encourage objective on-going quality control: "Quality assessment is built into the process, in all activities, involving all participants, using objective measurements and criteria, and not treated as an afterthought or a separate activity performed by a separate group".

In this document, we discuss how Software Quality Management is realised in the Rational Objectory Process. This evaluation is based on documentation provided along with the Objectory CASE tool Version 4.1. The Objectory Process will be described formally in Jacobson, Booch & Rumbaugh (Jacobson, Booch & Rumbaugh, 1999). However, this book is not yet available at the time of writing this docuemnt.

In the following sections we first describe the Quality Assurance in Objectory, and subsequently we evaluate this Quality Assurance, followed by some conclusions.

4.2.2 Objectory

Quality assurance in Objectory is realised through guidelines and checkpoints. There are four annotated templates for guidelines: Use Case Modeling Guidelines, Design Guidelines, Programming Guidelines and Test Guidelines. For each project these guidelines have to be defined. As an example the proposed outline of a Use-Case Modeling Guidelines document is as follows:

1. Brief Description	A brief description of the role and purpose of the Use-Case Modeling Guidelines
2. References	A description of related or referenced documents
3. General Use-Case Modeling Guidelines	The section describes which notation to use in the use-case model. For example, you may have decided to not use extends-relationships between use cases
4. How to Describe a Use Case	This section gives rules, recommendations and style issues, and how you should describe each use case

In Table 1 we show an overview of the lists with checkpoints as encountered in the Objectory process components Requirement Capture, Analysis and Design, Implementation and Test.

Requirement Capture	
Review the Use-Case Model	Checkpoints for the Use-Case Model Checkpoints for the Actors Checkpoints for Use Cases Checkpoints for Use-Case Descriptions Checkpoints for the Glossary
Analysis & Design	

Review the Architecture	Checkpoints for the Software Architecture in General Checkpoints for the Use-Case View Checkpoints for the Logical View Checkpoints for the Process View Checkpoints for the Deployment View
Review the Analysis	Checkpoints for the Design Model Checkpoints for Use-Case Realisations Checkpoints for Classes Checkpoints for Classes - Detailed
Review the Design	Checkpoints for the Design Model Checkpoints for Use-Case Realisations Checkpoints for Classes Checkpoints for the Design Guidelines
Implementation	
Review Code	Checkpoints for Source Code
Test	
Evaluate Test	Evaluate Test-Case Coverage Evaluate Code Coverage Analyse Defects Determine if Test Completion and Success Criteria Are Met

Table 1. Overview of checkpoints for steps in the Objectory process components

As exemplification we will quote some concrete checkpoints related to use cases. First use cases are introduced briefly. A use case is a specification of actions, including variants, which a system can perform, interacting with an actor of the system. A use case is a specific way of using the system by performing some part of the functionality. A use case instance - also called a scenario - is a specific sequence of actions as specified in a use case carried out under certain conditions. A use case model or diagram contains a collection of related use cases. The flow of control within each use case can be derived from the sequence diagrams (Jacobson *et al.*, 1992; Rational, 1997). Some checkpoints in Objectory related to use cases are given in Table 2.

Checkpoints for the Use-Case Model

1. Have you found all the use cases? Those you have found must be able to perform all system behaviours; if not, some use cases are missing.

2. Does the use-case model contain any superfluous behaviour; that is, does it present more functions than were called for in the requirements?

3. Is the division of the model into use-case packages appropriate? Does the packaging make the model more simple and intuitive to understand and maintain?

4. Does the Introduction section of the use-case model contain all the necessary information?
Checkpoints for Use Cases 5. Is each concrete use case involved with at least one actor? If not, something is wrong: a use case that does not interact with an actor is superfluous, and you should remove it. 6. Has part of the flow of events already been modelled as another use case? If so, you can have the new use case use the old one. 7. Do the use cases have unique, intuitive, and explanatory names so that they cannot be mixed up at a later stage? If not, you change their names.
Checkpoints for Use-Case Descriptions 8. Does the use case meet all the requirements that obviously govern its performance? You must include any (non-functional) requirements to be handled in the object models in the use-case Special Requirements. 9. Are any use cases overly complex? If you want your use-case model to be easy to understand, you might have to split up complex use cases. 10. Does the brief description give a true picture of the use case?
Checkpoints for the Use-Case View 11. Does the Use-Case View describe all the scenarios and use cases that were the input to the current Iteration Plan?
Checkpoints for Use-Case Realisations in Analysis 12. Has each flow of the corresponding use case been handled? 13. Does the flow of the use case proceed naturally over the participating objects, or are there things that need to be made smoother? 14. If there are several collaboration diagrams for the use-case realisation, is it easy to understand which collaboration diagrams relates to which flow of events?
Checkpoints for Use-Case Realisations in Design 15. Have you found all the required objects and classes for each use-case realisation? 16. Do the sequence diagrams correctly describe the use case's flow of events? 17. Does the use case flow naturally over the participating objects, or do some places need smoothing?

Table 2 Some checkpoints in Objectory related to use cases

Furthermore, there are in Objectory two key measures of a testing process (see step Evaluate Test in Table 1). The first measure is testing completeness and is based on the coverage of testing, expressed either by the coverage of test requirements and test cases, or the coverage

of executed code. The second measure is of reliability, and based on analyses of defects discovered during the testing. For each of these measures reports can be generated.

We now continue with an evaluation of the quality assurance in the Objectory process.

4.2.3 Evaluation

Many software quality models have been proposed in the past, e.g. the McCall model and the Boehm model (see Pfleeger, 1998). The ISO 9126 is an international standard for measuring software product quality. Six major characteristics contribute to quality of a software product: functionality, reliability, usability, efficiency, maintainability and portability. Each characteristic is determined by a number of attributes. The Factor-Criteria-Metric model of McCall considers three categories of quality factors according to product operation, product revision and product transition (Fenton & Pfleeger, 1996). Quality factors are for example reliability, correctness, maintainability, testability and reusability. For each factor there are a number of criteria as completeness, consistency, simplicity, conciseness, expandability, self-descriptiveness and modularity. Next, criteria are associated with sets of low-level, directly measurable attributes (both product and process).

In Table 3 we relate some quality criteria with the checkpoints from Table 2. Moreover we mention some candidate metrics.

checkpoint	**criteria**	**metric**
1	completeness	% system behaviours covered by use cases
3	modularity	cohesion coupling
7	self-descriptiveness	# non-unique names in glossary % names of three or more syllables
9	simplicity	McCabe complexity of sequence diagram
12	completeness	% paths of control flow in sequence diagram covered in realisation

Table 3 Example quality criteria and metrics for checkpoints

For checkpoint 3, the division of the model into use-case packages, it is suggested to elaborate metrics for cohesion and coupling (Fenton & Pfleeger, 1996). These metrics have to be adapted to use-cases. Fowler (Fowler & Scott, 1997b) presents a heuristic to minimise the dependencies between packages. A dependency between two packages exists if any dependency exists between any two classes in the packages. Dependencies between classes may exist for various reasons: One class sends a message to another; one class has another as part of its data; one class mentions another as a parameter to an operation. No further details are given for the minimization of dependencies.

There are checkpoints for which it appears difficult to define a metric, such as in Table 2 checkpoint 10 (true picture) and checkpoints 13 and 17 (naturally, smoother).

The simplicity (or oppositely the complexity) of use cases, i.e. checkpoint 9, could be quantified with convential metrics such as the McCabe cyclomatic complexity number derived for the control flow in the sequence diagram of a use case. The complexity could also be expressed with simple size metrics, such as the number of uses-relationships and the number of extends-relationships.

There are several exposés on other OO-metrics (Lorenz & Kidd, 1994; Henderson-Sellers, 1996). Also in his first book on Objectory, Jacobson (Jacobson *et al.*, 1992) discusses the use of metrics. The Objectory checkpoints could be improved by an explicit description of the metrics to be used.

Metrics should be part of a quality assurance plan, in which they are related to software quality goals. A software quality characteristic may be set as goal to be achieved in the software development process. Metrics should be selected to enable the assessment of successfully achieving these goals. For example, the Hewlett-Packard FURPS-model provides measurable objectives for each life cycle phase (Grady, 1992).

Goals may relate to the software development process, such as productivity and schedules. Goals may be set from different viewpoints (user/customer, development engineer, support engineer) or from different levels (engineer, project, division, company) (Pfleeger, 1998).

In order to relate goals with metrics, the Goal-Question-Metric (GQM) paradigm has been introduced by Basili & Rombach (Basili & Rombach, 1988) (discussed by Fenton & Pfleeger, 1996). A goal raises several questions, each with a set of metrics. Grady (Grady, 1992) gives a detailed description of the following project management goals (with related questions and metrics): Maximise customer satisfaction, minimise engineering effort and schedule, and minimise defects. There may be conflicting goals. For example, the goal to reach maintainability/adaptability of the software may slow down productivity in the design phase. In Objectory there is no explicit statement of various goals to be achieved.

As stated in the introduction, Software Quality Management involves the application of a comprehensive measurement program to the software products. There are several phases in such a measurement program (e.g. the AMI approach in Pulford *et al.*, 1996, i.e. Application of Metrics in Industry; Roche & Jackson, 1994; Park *et al.*, 1996):

1. assessment of the context for the measurement

2. the formulation of the measurement goals and the metrics (as in GQM)

3. the collection and storage of measurement data

4. the analysis of the data

5. the interpretation in order to give feedback and facilitate quality improvement of process and products.

The goals can be defined with a template, which include the object of measurement, the purpose, the quality focus, the viewpoint and context of the measurement (see Fenton & Pfleeger, 1996). The interpretation of data should to be based on baseline values and threshold values for metrics. Furthermore, there must be models with hypotheses which describe the relation between the measured attributes and the quality factors, both for defining the goals and metrics and for the analysis and interpretation of the data (Latum *et al*., 1996).

For example, a goal is to minimise the maintainability effort in a project, i.e. the quality focus, from the viewpoint of the support engineer. Object of measurement is the set of use cases used in the development of the software. One of the questions is checkpoint 9 in the Review of the Use Case model (from Table 2). A metric is McCabe cyclomatic complexity of the control flow in the sequence diagram. A hypothesis is that use cases with complexity larger than 10 should be split into sub use cases in order to reduce the maintainability effort. The hypotheses and the thresholds should be validated in the context of the actual improvement of the ongoing project.

The Objectory tool only provides partial support for the phases in a measurement plan. The use of most given checkpoints is rather subjective. Models for the interpretation of measurements are not provided. Although the checkpoints are integrated in the Objectory process, the collection, the analysis and interpretation of data related with the use of the checkpoints is neither integrated nor automated.

We will summarise the evaluation given above. The Rational Objectory CASE tool provides comprehensive support to the software development process. Quality assurance is an integral part of this process through templates for guidelines and many checkpoints to check the quality of artifacts to be delivered in each phase. However, there are no clear goals defined for which metric values are to be collected. The Goal-Question-Metric paradigm could provide a framework for making goals and interpretation models explicit as part of the measurement plan. The Objectory CASE tool is already linked with a number of other tools, such as tools for visual modeling and requirements management. Objectory could be improved by a linkage to quality assessment tools that supports – if possible automated - data collection, storage, analysis and interpretation of quality metrics.

5. Conclusion

In this document we discussed two major software object-oriented process models. The OPEN process model has a stronger academic bias, whereas the Unified Process model has a stronger industrial momentum through its reliance on the UML effort of the Rational company.

Both approaches mainly focus on the primary life cycle process *Development* of the ISO-IEC-12207 standard. The organisational life cycle processes are only touched in a very limited way. Furthermore, they provide for some of the supporting life cycle processes. In this document we described how the life cycle process Quality is aimed at in Objectory. Mainly guidelines are given, but hardly any support for metrics is provided.

For further research, it is recommended to investigate the combination of the ISO-IEC life cycle processes and the CMM model with the Key Process Areas, and subsequently analyse both Objectory/Unified Process and the OPEN process model in that framework.

References

[Aliee & Warboys, 1995] Aliee, F.S. & Warboys, B.C. (1995). *Applying Object-Oriented Modeling to Support Process Technology*. Proceedings 1st World Conference on Design and Process Technology IDPT-Vol. 1 A. Ertag *et al.*, Eds., Austin Texas.

[Basili & Rombach, 1988] Basili, V.R., & Rombach, H.D. (1988). The TAME Project: Towards Improvement-Oriented Software Environments. *IEEE Trans. Softw. Eng., SE 14*, 758-773.

[Booch, 1998] Booch, G. (1998). *Quality Software and the Unified Modeling Language*. See: http://www.rational.com/support/techpapers/soft_uml.html

[Booch, Rumbaugh & Jacobson, 1999] Booch, G., Rumbaugh, J. & Jacobson, I. (1999). The Unified Modeling Language User Guide. Addison Wesley Longman

[Christie *et al.*, 1996] Christie, A.M., Earl, A.N., Kellner, M.I. & Riddle, W.E. (1996). *A Reference Model for Process Technology.* In: Montangero (1996), pp. 3-17.

[Conradi *et al.*, 1994] Conradi, R., Fernström, C. & Fuggetta, A. (1994). *Concepts for Evolving Software Processes*. In: Finkelstein *et al.* (1994), pp 9-31.

[Curtis *et al.*, 1992] Curtis, B., Kellner, M.I. & Over, J. (1992). *Process Modeling*. Comm. ACM, Vol. 35 No 9, pp. 75-90.

[EWSPT-1, 1991] *Proceedings of the First European Workshop on Software Process Modeling* (1991), Milan, Italy.

[Feiler & Humphrey, 1992] Feiler, P.H. & Humphrey, W.S. (1992). *Software Process Development and Enactment*, (CMU/SEI-92-TR-04, ADA258465). Pittsburgh, PA: SEI, Carnegie Mellon University.

[Fenton & Pfleeger, 1996] Fenton, N.E. & Pfleeger, S.L. (1996), Software Metrics, A Rigorous & Practical Approach. 2nd edition. Thomson, London.

[Finkelstein *et al.*, 1994] Finkelstein, A., Kramer, J. & Nuseibeh, B. (1994). *Software Process Modeling and Technology*. Wiley, New York.

[Firesmith *et al.*, 1997] Firesmith, D., Henderson-Sellers, B. & Graham, I. (1997). OPEN Modeling Language (OML) Reference Manual. Sigs, New York

[Firesmith *et al*., 1998] Firesmith, D., Krutsch, S., Stowe, M. & Hendley, G. (1998). Documenting a Complete Java Application Using OPEN. Addison-Wesley.

[Forte, 1989] Forte, G. (1989). In Search of the Integrated Environment. *CASE Outlook*, March/April 1989, 5-12. Quoted in Pressman (1997), p. 845.

[Fowler & Scott, 1997b] Fowler, M. & Scott, XXX (1997). UML Distilled. Applying the Standard Object Modeling Language. Addison-Wesley, Reading

[Fugetta & Wolf, 1996] Fuggetta, A. & Wolf, A. (1996). *Software Process*. Wiley, Chichester.

[Grady, 1992] Grady, R.B. (1992). *Practical Software Metrics for Project Management and Process Improvement*. Prentice Hall.

[Graham, Henderson-Sellers & Younessi, 1997] Graham, I., Henderson-Sellers, B. & Younessi, H. (1997). The OPEN Process Specification. Addison-Wesley, Harlow

[Henderson-Sellers, 1996] Henderson-Sellers, B. (1996). *Object-Oriented Metrics: Measures of Complexity*, Prentice-Hall.

[Henderson-Sellers, Simons & Younessi, 1998] Henderson-Sellers, B., Simons, A. & Younessi, H. (1998). The OPEN Toolbox of Techniques. Addison-Wesley, Harlow

[Huff, 1996] Huff, K.E. (1996). *Software Process Modeling*. In: Fuggetta & Wolf (1996), pp. 1-24.

[ICSP-1, 1991] *Proceedings of the First International Conference on the Software Process* (1991), IEEE Computer Society, Washington, DC.

[IEEE-SE, 1993] *Evolution of Software Processes* (1993). Special Section in IEEE Transactions on Software Engineering, Vol. 19, No. 12.

[ISO/IEC 12207, 1995] ISO/IEC 12207 (1995). Standard for Information Technology-software life cycle processes. http://www.software.org/Quagmire/descriptions/iso-iec-12207.html

[ISWP-1, 1984] *Proceedings of the First International Software Process Workshop* (1984), ISPW-1, Runnymede, UK, IEEE Computer Society.

[Jacobson *et al.*, 1992] Jacobson, I., Christerson, M. Jonsson, P. & Övergaard, G. (1992). *Object-Oriented Software Engineering, A Use Case Driven Approach*. Addison-Wesley, Wokingham.

[Jacobson *et al.*, 1997] Jacobson, I, Griss, M. & Jonsson, P. (1997). Software Reuse. Architecture, Process and Organization for Business Success. Addison Wesley Longman

[Jacobson, Booch & Rumbaugh, 1999] Jacobson, I., Booch, G. & Rumbaugh, J. (1999). The Unified Software Development Process. Addison Wesley Longman, 0-201-57169-2 (not yet published).

[Kellner *et al.*, 1990] Kellner, M.I., Feiler, P.H., Finkelstein, A., Katayama, T., Osterweil, L.J., Penedo, M.H. & Rombach, H.D. (1990). *Software Process Example*, Proceedings ISPW-6, T. Katayama Ed., IEEE Computer Society Press.

[Latum *et al*., 1996] Latum, F. van, Oivo, M., Hoisl, B. & Ruhe, G. (1996). No Improvement without Feedback: Experiences from Goal-Oriented Measurement at Schlumberger. In C. Montangero (Ed.). *Software Process Technology*, Proceedings EWSPT'96, LNCS 1149, Springer, pp. 167- 182.

[Lorenz & Kidd, 1994] Lorenz, M. & Kidd, J. (1994). *Object-Oriented Software Metrics*. Prentice-Hall.

[Montangero, 1996]Montangero, C. (1996). *Software Process Technology*. Proceedings 5th European Workshop, EWSPT'96, LNCS 1149, Springer, Berlin.

[Park *et al.*, 1996] Park, R.E., Goethert, W.B., & Florac, W.A. (1996). *Goal-Driven Software Measurement – A Guidebook.* Handbook CMU/SEI-96-HB-002, Software Engineering Institute, Carnegie Mellon University.

[Paulk *et al.*, 1993] Paulk, M. C. *et al.* (1993). *Capability Maturity Model for Software*, Version 1.1, (CMU/SEI-93-TR-24), Pittsburgh, PA: SEI, Carnegie Mellon University.

[Paulk *et al.*, 1993b] Paulk, M. C. *et al.* (1993). *Key Practices of the Capability Maturity Model*, Version 1.1, CMU/SEI-93-TR-25, Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University.

[Pfleeger, 1998] Pfleeger, S.L. (1998). Software Engineering. Prentice Hall

[Prasse, 1998] Prasse, M. (1998). Evaluation of Object-Oriented Modeling Languages: A Comparison Between OML and UML. In: M. Schader & A. Korthaus (Eds.) (1998). The Unified Modeling Language. Physica-Verlag, pp. 58-75.

[Pressman, 1997] Pressman, R.S. (1997). Software Engineering, A Practitioner's Approach. 4th Edition, European Adaptation by D.Ince, McGraw-Hill

[Pulford *et al.*, 1996] Pulford, K., Kuntzmann-Combelles, A. & Shirlaw, S. (1996). *A Quantitative Approach to Software Management, The ami handbook*. Addison-Wesley.

[Rational, 1997] Rational (1997). UML Summary, Semantics, Notation Guide, Version 1.1, Rational Software Corporation. See http://www.rational.com/uml/resources.html

[Rational, 1997b] Rational (1997b). *Reaching CMM Levels 2 and 3 with the Rational Objectory Process. Rational Software Corporation*.
http://www.rational.com/support/techpapers/OBJ98002.pdf

[Rational, 1998] *Rational Objectory Process*, Version 4.1, Rational Software Corporation. See http://www.rational.com/demos/o_process/

[Roche & Jackson, 1994] Roche, J. & Jackson, M. (1994). Software Measurement Methods: Recipes for Success? *Information and Software Technology*, 36(3), 173-189.

[SEJ, 1991] *Software Process & Support* (1991). Special Issue of the Software Engineering Journal, Vol. 6, No. 5.

[Sommerville, 1996] Sommerville, I. (1996). Software Engineering, 5th Edition, Addison-Wesley