



**WP 8.1 – Define Methodology for
Validation within OATA**

**ARCHITECTURE STRUCTURE
ASSESSMENT PROCESS**

Document Identifier:	OATA-P2-D8.1.2-01
Technical Baseline:	N/A

Edition Number:	0.M
Edition Date:	8-Jun-05
Status:	Proposed Issue
Intended for:	TRG Members

DOCUMENT CHARACTERISTICS

TITLE			
WP 8.1 – Define Methodology for Validation within OATA			
ARCHITECTURE STRUCTURE ASSESSMENT PROCESS			
EATMP Infocentre Reference:			
Document Identifier:	OATA-P2-D8.1.2-0	Edition Number:	0.M
Technical Baseline:	N/A	Edition Date:	8-Jun-05
Abstract			
This document presents a methodology that permits the assessment of the architectural quality of the OATA logical architecture by using suitable metrics.			
Keywords			
Validation ISO 9126	Logical Architecture Quality Framework	Method	Metrics
Contact Person(s)		Tel	Unit
Prepared by:	SWEDAVIA, AerotechTelub, ISDEFE		
Issued by:	Hans Wagemans, WP8.1 Manager	+32 2 729 3334	SD/ESC

STATUS, AUDIENCE AND ACCESSIBILITY					
Status		Intended for		Accessible via	
In progress	<input type="checkbox"/>	General Public	<input type="checkbox"/>	Intranet	<input type="checkbox"/>
Internal Draft	<input type="checkbox"/>	EATMP Stakeholders	<input checked="" type="checkbox"/>	Extranet	<input checked="" type="checkbox"/>
Working Draft	<input type="checkbox"/>	Restricted Audience	<input type="checkbox"/>	Configuration Manager	<input type="checkbox"/>
Proposed Issue	<input checked="" type="checkbox"/>	<i>Printed & electronic copies of the document can be obtained from the EATMP Infocentre (see page iii)</i>			
Released Issue	<input type="checkbox"/>				

ELECTRONIC SOURCE		
Path:	http://pollux.mis.eurocontrol.be/iw/cci/meta/no-injection/iw-mount/default/main/template_web/eatm/valfor/WORKAREA/work/web/gallery/content/public/zz_tst_borkenhu_1_OATA-P2-D8.1.2-01 DMVO Architecture Structure Assessment Process.doc	
File Name:	OATA-P2-D8.1.2-0 (Architecture Structure Assessment Process ed_0.M).doc	
Host System:	Software Application	Size:
Windows XP:	Microsoft Word 10.0	2931 Kb

EATMP Infocentre
 EUROCONTROL Headquarters
 96 Rue de la Fusée
 B-1130 BRUSSELS

Tel: +32 (0)2 729 51 51
 Fax: +32 (0)2 729 99 84
 E-mail: eatmp.infocentre@eurocontrol.int

Open on 08:00 - 15:00 UTC from Monday to Thursday, incl.

DOCUMENT APPROVAL

The following table identifies all management authorities who have successively approved the present issue of this document.

AUTHORITY	NAME AND SIGNATURE	DATE
Author	SWEDAVIA	
WP Manager	HANS WAGEMANS	

DOCUMENT CONTROL

Copyright notice

© 2007 European Organisation for the Safety of Air Navigation (EUROCONTROL).
 All rights reserved.
 "Member States of the Organisation are entitled to use and reproduce this document for internal and non-commercial purpose under their vested tasks. Any disclosure to

third parties shall be subject to prior written permission of EUROCONTROL".

DOCUMENT CHANGE RECORD

The following table records the complete history of the successive editions of the present document.

EDITION NUMBER	EDITION DATE	REASON FOR CHANGE	PAGES AFFECTED
0.A	13/04/04	Creation	All
0.B	Internal Version not delivered	Restructuring	All
0.C	15/04/04	Modification	All
0.D	12/05/04	Modification including review meetings comments	All
0.E	23/6/2004	Modifications based on reviewers comments	All
0.F	28/07/2004	Modifications based on reviewers comments	All
0.G	23/08/2004	Internal release	All
0.H	19/11/2004	Updates after AT internal review	All
0.I	23/12/2004	Modifications including review meeting comments	All
0.J	24/01/2005	Modifications including review comments	All
0.K	09/02/2005	Updates from review	All
0.L	25/02/2005	Updates	All
0.M	02/03/2005	Updates during review	All

TABLE OF CONTENT

DOCUMENT CHARACTERISTICS	II
DOCUMENT APPROVAL.....	III
DOCUMENT CONTROL.....	III
DOCUMENT CHANGE RECORD	IV
TABLE OF CONTENT	V
LIST OF FIGURES.....	VII
EXECUTIVE SUMMARY.....	1
1 INTRODUCTION.....	2
1.1 DOCUMENT BACKGROUND	2
1.2 DOCUMENT STRUCTURE	2
1.3 READING GUIDELINES.....	3
1.4 DEFINITIONS AND TERMS.....	4
1.5 BIBLIOGRAPHY.....	5
2 METHODOLOGY OVERVIEW.....	7
2.1 INTRODUCTION	7
2.2 VALIDATION OF THE OATA ARCHITECTURE STRUCTURE.....	7
2.3 QUALITY MODEL OVERVIEW	7
2.3.1 Quality Characteristics.....	8
2.3.2 Metrics	9
3 PROCESSES	10
3.1 INTRODUCTION	10
3.1.1 Required roles	14
3.2 PREPARATION.....	14
3.2.1 Introduction	14
3.2.2 Required resources	17
3.2.3 Activity: Plan the validation cycle and prepare the validation team and repository.....	17
3.2.4 Activity: Select the OATA architecture version.....	18
3.2.5 Activity: Define validation aims and objectives.....	19
3.2.6 Activity: Select metrics.....	21
3.2.7 Optional Activity: Select OATA architecture subsets.....	22
3.2.8 Optional Activity: Obtain values from legacy architectures	23
3.2.9 Activity: Set thresholds	24
3.2.10 Activity: Prepare tools.....	25
3.3 EXECUTION	27
3.3.1 Introduction	27
3.3.2 Required resources	29
3.3.3 Activity: Perform Measurement	29
3.4 ANALYSIS AND REPORTING.....	30
3.4.1 Introduction	30
3.4.2 Required resources	32
3.4.3 Activity: Analyse the metric measurement results.....	32
3.4.4 Activity: Assess the Validation Aims.....	33
3.4.5 Activity: Write the validation report(s).....	34

4	APPENDIX A: QUALITY MODEL	36
4.1	INTRODUCTION	36
4.2	OVERVIEW OF QUALITY MODELS AND PROPOSED QUALITY MODEL	36
4.3	DESCRIPTION OF ISO/IEC 9126 QUALITY CHARACTERISTICS	37
4.3.1	Definitions of characteristics.....	38
4.4	THE OATA ARCHITECTURE STRUCTURE INTERNAL QUALITY CHARACTERISTICS	41
4.5	WEIGHT TABLE FOR SUBCHARACTERISTICS.....	42
4.6	PROPOSED OATA QUALITY MODEL.....	42
4.6.1	Proposed Quality Model	43
4.7	IMPACT MATRIX OF METRICS ON QUALITY (FOR MAINTAINABILITY).....	45
5	APPENDIX B METRICS	47
5.1	INTRODUCTION	47
5.2	OPERATIONS PER CLASS	47
5.2.1	Description and calculation.....	47
5.2.2	Quality and interpretation	47
5.2.3	Desired value.....	48
5.2.4	OATA Applicability.....	48
5.2.5	References/Sources	48
5.3	COUPLING METRICS	49
5.3.1	Description.....	49
5.3.2	Quality.....	50
5.3.3	Coupling between classes.....	50
5.3.4	Coupling between packages	51
5.4	STABILITY METRICS	51
5.4.1	Description.....	51
5.4.2	Quality.....	52
5.4.3	Efferent and afferent coupling	52
5.4.4	Instability.....	53
5.4.5	Cyclic dependency	54
5.5	INHERITANCE	55
5.5.1	Description.....	55
5.5.2	Quality.....	56
5.5.3	Depth of inheritance	56
5.5.4	Width of inheritance.....	57
5.6	ENCAPSULATION.....	58
5.6.1	Description and calculation.....	58
5.6.2	Quality and interpretation	58
5.6.3	Desired value.....	58
5.6.4	OATA Applicability.....	58
5.6.5	References/Sources	58
5.7	RESPONSE FOR A CLASS	59
5.7.1	Description and calculation.....	59
5.7.2	Quality and interpretation	59
5.7.3	OATA Applicability.....	59
5.7.4	References/Sources	59
5.8	READABILITY	60
5.8.1	Description and calculation.....	60
5.8.2	Quality and interpretation	60
5.8.3	Desired value.....	60
5.8.4	OATA Applicability.....	60
5.8.5	References/Sources	60

6	APPENDIX C METRIC CARDS.....	61
6.1	METHODS PER OBJECT CLASS	61
6.2	WEIGHTED METHODS PER CLASS	62
6.3	DEPTH OF INHERITANCE, DIT.....	63
6.4	NUMBER OF CHILDREN, NOC	63
6.5	COUPLING BETWEEN CLASSES, CBC	64
6.6	COUPLING BETWEEN OBJECT CLASSES, CBO.....	65
6.7	COUPLING FACTOR, CF,COF	65
6.8	AFFERENT COUPLING, CA	66
6.9	EFFERENT COUPLING, CE	67
6.10	INSTABILITY, I	68
6.11	CYCLIC DEPENDENCIES, CYC	69
6.12	RESPONSE FOR A CLASS, RFC	70
6.13	ENCAPSULATION PRINCIPLE, EP.....	70
6.14	LENGTH OF NAMES, LEN.....	71
6.15	NAME UNIQUENESS RATIO, UNIQ	71
6.16	COMMENT DENSITY, DC.....	71
7	APPENDIX D: SCORECARD EXAMPLE.....	72
7.1	INTRODUCTION	72
7.2	SCORECARD.....	73
8	APPENDIX E: AVAILABLE TOOLS LIST.....	74
8.1	INTRODUCTION	74
8.2	OATA SUITABLE TOOL FEATURES.....	74
8.3	ARCHITECTURE AND CODE METRICS TOOLS.....	75
8.4	CODE METRICS TOOLS	75
8.5	OATA TOOL RECOMMENDATIONS	76
8.6	DETAILED DESCRIPTION OF TOOLS	77
8.6.1	Logiscope.....	77
8.6.2	McCabe QA	77
8.6.3	Insure.....	78
8.6.4	CodeWizard.....	78
8.6.5	Imagix 4D.....	78
8.6.6	JMetric.....	79
8.6.7	QSM - SLIM-Metrics.....	79
8.6.8	SDMetrics	81
8.6.9	Metamata Metrics.....	81
8.6.10	Scientific Toolworks, Inc.....	82
8.6.11	ObjectQuest Advisor.....	82

LIST OF FIGURES

Figure 1: Summary of the Architecture Validation Process	2
Figure 2: Process Overview	11
Figure 3: Resource Overview	12
Figure 4: Information Flow Overview.....	13
Figure 5: Preparation	16
Figure 6: Execution	28
Figure 7: Analysis and Reporting.....	31
Figure 8: ISO/IEC 9126 Quality Model for External and Internal Quality.....	37
Figure 9: Quality Meta Model.....	43
Figure 10: Quality Attribute Relationships	44
Figure 11: Aspects of Coupling between Classes.....	49
Figure 12: Package and Class Dependencies.....	54
Figure 13: Tool Metamodel	74

EXECUTIVE SUMMARY

An architecture is a formal description that encompasses a set of design decisions that affects the system behaviour and performance. A validation of the quality of the OATA logical architectural structure is required to increase the confidence of the architecture being developed, and to ensure that it meets Stakeholders' expectations and quality criteria.

The validation of the structure of an architecture focuses on assessing different quality characteristics. The validation is guided by the definition of validation aims. A validation aim is a high-level expectation of the logical architecture and indicates the quality characteristics to be further investigated and assessed in a validation cycle.

The selected Quality Model for the validation of the OATA architecture structure is ISO/IEC 9126. This Quality Model represents an international effort to collect the ideas and experiences related to the quality of a software system over the last decades.

The ISO/IEC 9126 categorises quality into six relevant quality characteristics to be used for the assessment of quality. One key aspect of the characteristics is that they are reasonably independent and cover a wide range of quality aspects. This validation methodology focuses on the quality characteristics, maintainability and understandability. These quality characteristics are further subdivided into subcharacteristics, which are influenced by different quality attributes. A quality attribute is a measurable property that can be quantified by one or several metrics. A metric is a definition of measurement method of a certain property in Architecture. Included in this methodology is a set of suitable metrics for the assessment of the OATA logical architecture.

The analysis and conclusions of the validation results and the recommendations of the validation team (documented in a validation report) can be used for further refinement of the OATA logical architecture.

1 INTRODUCTION

1.1 Document background

The purpose of the validation in OATA is to increase confidence in the architecture being developed, and to ensure that it meets Stakeholders' expectations and quality criteria.

The validation will be performed around four areas: Architecture Compliance with User Needs, Quality of the Architecture Structure, Compliance with Non-Functional Requirements, and Architecture Tactics Assumptions.

This document contains a methodology for the validation of the OATA logical architectural structure that is the second stage in the validation.

Figure 1 shows the four areas of the OATA validation; the area that this document is concerned with is expanded.

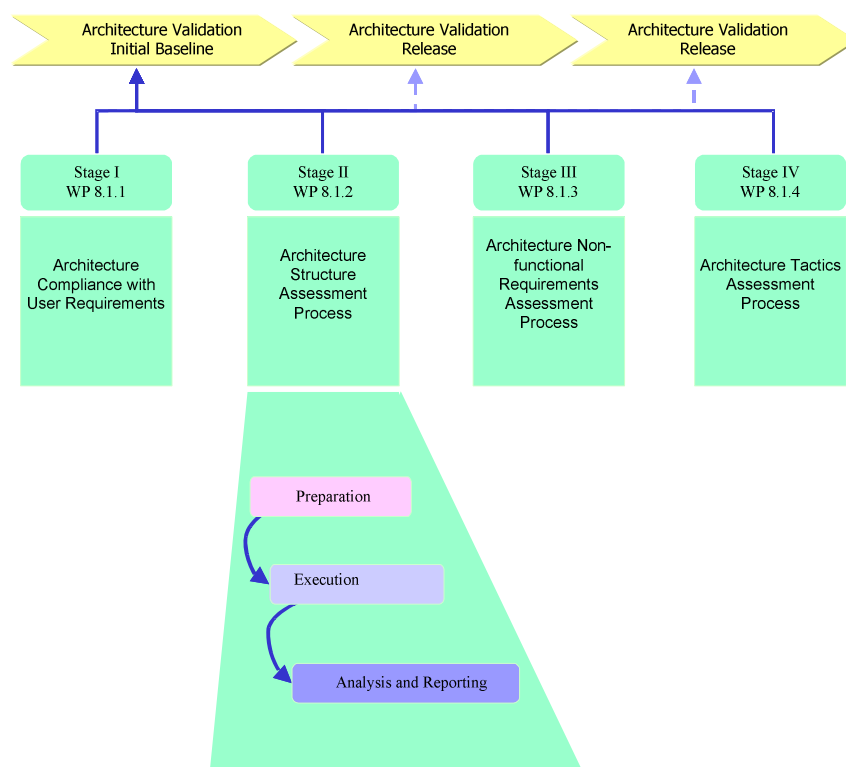


Figure 1: Summary of the Architecture Validation Process

1.2 Document structure

This document has the following main chapters:

- **Introduction**

The introduction contains description of the background of this document, definitions and terms, and a list of references.

- **Methodology Overview**

The methodology overview contains a high level description of the validation process and the proposed quality model to be used in the validation.

- **Processes**

The processes chapter details each part of the validation process and describes the activities that must be undertaken during a validation cycle.

There are also several appendices included in this document that give a more detailed description of elements of the validation methodology:

- Quality Model
- Metrics
- Metric Cards
- Scorecard Example
- Available Tools List

1.3 Reading guidelines

The processes in this methodology are visualised as process diagrams using standard UML Business Modelling syntax, [Eriksson, Penker 00].

The process diagram contains information about:

- **Processes and Activities**

A process is a set of related activities and they are drawn in the centre/middle of the process diagrams with solid arrows linking processes/activities that follow one another.

- **Resources**

Resources are the objects within the business, such as people, material, information, and products, which are used or produced in the business. Resources are manipulated (used, consumed, refined, or produced) through processes.

Controlling resources control or run a process. These resources are drawn above a process with a dashed line from the resource to the process.

Supporting resources participate in a process and are not refined or consumed. These resources are drawn below a process with a dashed line from the resource to the process.

Input information resources are placed to the left of the process and connected to the process with a dashed line.

Output information resources are placed to the right of the process and connected to the process with a dashed line.

- **Activity synchronization**

Activities can be performed in parallel. A need for synchronization is shown in the process diagram by using a vertical synchronization bar. The activities after the synchronization bar must wait for all the activities, before the bar, to complete before they can start.

A detailed process diagram contains activities and their relationships.

For more detailed information see [Eriksson, Penker 00].

1.4 Definitions and Terms

External Quality

The extent to which a product satisfies stated and implied needs when used under specified conditions.

[ISO 9126]

Implied Needs

Needs that may not have been stated but are actual needs when the entity is used in particular conditions.

Note: Implied needs are real needs, which may not have been documented.

[ISO 9126]

Internal Quality

The totality of attributes of a product that determines its ability to satisfy stated and implied needs when used under specified conditions.

[ISO 9126]

Quality

The totality of characteristics of an entity that bear on its ability to satisfy stated and implied needs.

[ISO 9126]

Note: An entity can be a product, service or process.

Quality Attribute

Quality Attribute is a measurable physical or abstract property of an entity (product or service).

[ISO 9126]

Quality Characteristic

Inherent characteristic of a product, process or system related to a requirement.

[ISO 9000:2000]

Quality Subcharacteristic

Quality Subcharacteristic is subdivided quality characteristics.

Quality Model

Set of characteristics and relationships between them, which provide the basis for specifying quality requirements and evaluating quality.

[ISO 9126]

Metric

A defined measurement method and the measurement scale.

[ISO 9126]

Note: A metric -- not to be confused with a measure -- is a quantitative property of software products or processes whose possible values are numbers. A measure is the value of a metric for a certain product or process.

Threshold

A defined value used as an acceptable level (of quality).

UML

Unified Modelling Language, an OMG standard for visual object-oriented modelling.

Validation Aim

Is a sentence describing a high-level expectation that should be reflected in the OATA architecture.

Repository

A (structured) information storage area.

1.5 Bibliography

Abreu, Melo 96	Abreu, F. B. e. and Melo, W., "Evaluating The Impact Of Object-Oriented Design On Software Quality", Proceedings of 3rd International Software Metrics Symposium, 1996.
Abreu 95	Abreu, F. B. e., "The MOOD Metrics Set," presented at ECOOP '95 Workshop on Metrics, 1995.
Bansiva, Davis 97	Bansiya, J.; Davis, C. "An Object-Oriented Design Quality Assessment Model". 1997.
Basili, Briand, Melo 96	Basili, V. R., Briand, L. C., and Melo, W. L., "A Validation of Object Orient Design Metrics as Quality Indicators," IEEE Transactions on Software Engineering, vol. 21, pp. 751-761, 1996.
Boehm 87	Boehm, B. W., "Improving Software Productivity," IEEE Computer, pp. 43-57, September 1987.
Boehm 78	B.W. Boehm, J.R. Brown, J.R Kaspar et al., "Characteristics of Software Quality", TRW series of Software Technology, Amsterdam, North Holland, 1978
Briand, Ikononovski, Lounis, Wust 81	Briand, L., Ikononovski, S., Lounis, H., and Wust, J., "Measuring the Quality of Structured Designs," Journal of Systems and Software, vol. 2, pp. 113-120, 1981.
Chidamber, Kemerer 94	Chidamber, S. R. and Kemerer, C. F., "A Metric Suite For Object Oriented Design". IEEE Transactions on Software Engineering, Vol. 20, N° 6, June 1994, pp. 467-493.
Churcher, Shepperd 95	Churcher, N. I. and Shepperd, M. J., "Comments on 'A Metrics Suite for Object-Oriented Design'," IEEE Transactions on Software Engineering, vol. 21, pp. 263-5, 1995.
Conte, Dunsmore, Chen 86	Conte, Dunsmore, Chen, "Software engineering metrics and models", 86
Douglass 02	Bruce Powel Douglass, Real-Time Design Patterns (Robust Scalable Architecture for Real-Time Systems) Addison-Wesley, 2002
Dromey 95	G.R. Dromey, "A Model for Software Product Quality", IEEE Trans. Software Engineering, Vol. 21, No. 2, pp. 146-162, Feb. 1995
Elemam 00	Elemam, K., "A Methodology for Validating Software Product Metrics," National Research Council of Canada, Ottawa, Ontario, Canada NCR/ERC-1076, June 2000.
Eriksson, Penker 00	H-E Eriksson, M Penker, "Business Modeling with UML", 2000
Fenton, Pflieger 98	Fenton, N. E. and Pflieger, S. L., Software Metrics: A Rigorous and Practical Approach: Brooks/Cole Pub Co., 1998.
Firesmith	Firesmith, D., http://www.donald-firesmith.com/index.html?Glossary

Glasberg, Emem, Melo, Madhavji 00	Glasberg, D., Emam, K. E., Melo, W., and Madhavji, N., "Validating Object-Oriented Design Metrics on a Commercial Java Application," National Research Council 44146, September 2000.
Gustafsson, Prasad 91	Gustafson, D. A. and Prasad, B., "Properties of Software Measures," in Formal Aspects of Measurement, T. Denvir, Ed. New York: Springer-Verlag, 1991.
Harrison, Counsell, Nithi 98	Harrison, R., Counsell, S. J., and Nithi, R. V., "An Evaluation of the MOOD Set of Object-Oriented Software Metrics," IEEE Transactions on Software Engineering, vol. 24, pp. 491-496, June 1998.
In, Kim, Barry	In, P., Kim, S. and Barry, M., "UML-Based Object-Oriented Metrics for Architecture Complexity Analysis", Proceedings of Ground System Architectures Workshop (GSAW 03), El Segundo, CA, The Aerospace Corporation.
ISO 9126	ISO/IEC 9126-1,-2,-3, "Information Technology – Software Product Quality", International Organization for Standardization. Geneva 2000.
Karlsson 95	Guttorm Sindre, Reidar Conradi, and Even-André Karlsson: "The REBOOT Approach to Software Reuse", Journal of Systems and Software - special issue on software reuse, Vol. 30, No. 3, Sept. 1995, p. 201-212. SU-report 22/95.
Korson, Vaishnavi 86	Korson, T. D. and Vaishnavi, V. K., "An Empirical Study of Modularity on Program Modifiability," Empirical Studies of Programmers, pp. 168-86, 1986.
Lorenz, Kidd 94	Lorenz, M. and Kidd, J., "Object-Oriented Software Metrics: A Practical Guide", Prentice-Hall, 1994.
MacCall 77	J. A. McCall et al., "Factors in software quality," RADC TR-77369, 1977.
McCabe 96	A.H. Watson, T.J. McCabe "Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric", 1996
Morris 89	K.L. Morris, Metrics for object-oriented software development environments, Master Thesis, M.I.T Sloan School of Management, Boston, 1989. http://thesis.mit.edu
MOOSE 03	A.L Baroni, F.B Abreu, An OCL-Based Formalization of the MOOSE Metric Suite, 2003
R.C. Martin 94	Robert C. Martin "OO Design Quality Metrics", 1994
RefactorIT	http://www.refactorit.com
Schneidewind 92	Schneidewind, N. F., "Methodology for Validating Software Metrics," IEEE Transactions on Software Engineering, vol. 18, pp. 410-422, 1992.
Selby, Vasili 91	Selby, R. W. and Vasili, V. R., "Analyzing Error-Prone Systems Structure," IEEE Transactions on Software Engineering, vol. 17, pp. 141-152, 1991.

2 METHODOLOGY OVERVIEW

2.1 Introduction

This methodology is concerned with the validation of the OATA architecture structure. Architecture is a formal description that encompasses a set of design decisions that affects the system behaviour and performance.

The validation of the structure of an architecture focuses on assessing different quality characteristics, such as maintainability and understandability. A quality characteristic is influenced by different quality attributes and a quality attribute may influence several quality characteristics.

2.2 Validation of the OATA Architecture Structure

The validation of the OATA architecture structure is guided by the definition of validation aims. A validation aim is a high-level expectation of the logical architecture and indicates the quality characteristics to be further investigated and assessed in a validation cycle. For example, a validation aim might be *“The OATA architecture structure shall be easy to maintain”*.

The validation aim is decomposed into validation objective(s), which give more detailed information about which quality characteristics or attributes that will be assessed. An example of validation objective might be *“The architecture shall be easy to change”*.

The validation objective is associated to one or more quality attributes that can be quantified and measured in the architecture. The results of the measurements are compared with thresholds, set by a validation team, in order to assess the validation objective.

The assessment of each validation objective is the basis to conclude if the defined validation aim is fulfilled by the logical architecture. The results of the validation are documented in a validation report and can be used to refine the model, if necessary.

This method uses a Quality Model that represents the quality of a product in terms of its relevant characteristics. A Quality Model is useful support during the process of identifying characteristics and metrics from the validation objectives. However, it is important to underline (and realise) that using a Quality Model to define and assess quality is a subjective and iterative process.

An overview of the proposed Quality Model is given in the next section.

2.3 Quality Model Overview

Several different kinds of Quality Models were considered during the elaboration of this methodology but the ISO/IEC 9126 was selected to be used for the validation of the OATA architecture structure. This Quality Model represents an international effort to collect the ideas and experiences related to the quality of a software system over the last decades.

The ISO/IEC 9126 categorises quality into six relevant quality characteristics to be used for the assessment of quality. One key aspect of the characteristics is that they are reasonably independent and cover a wide range of quality aspects.

The quality characteristics have been further subdivided into subcharacteristics. These different subcharacteristics are influenced by different quality attributes. A quality attribute, for example *“coupling”*, is a measurable property that can be quantified by one or several metrics.

A metric is a definition of measurement method of a certain property in Architecture. For example, the metric *“Coupling between classes”* describes a way to count the number of couplings between classes, which provides one measure of the attribute *“coupling”*.

A Quality Meta Model and more details are described in Appendix A: “Quality Model”.

2.3.1 Quality Characteristics

ISO/IEC 9126 has defined the following six relevant characteristics to assess quality:

- **Functionality**

The capability of the software product to provide functions, which meet, stated and implied needs when the software is used under specified conditions. [ISO 9126]

The functionality quality characteristic is assessed in the WP 8.1.1: Define Methodology for Validation within OATA - Architecture Compliance Assessment Process.

- **Reliability**

The capability of the software product to maintain a specified level of performance when used under specified conditions.[ISO 9126]

The reliability quality characteristic is assessed in the WP 8.1.3: Define Methodology for Validation within OATA - Non-Functional Requirements Assessment Process.

- **Usability**

The capability of the software product to be understood, learned, used and attractive to the user, when used under specified conditions. [ISO 9126]

The usability characteristic of the architecture itself is assessed in the method presented here.

- **Efficiency**

The capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions. [ISO 9126]

The efficiency quality characteristic is assessed in the WP 8.1.3: Define Methodology for Validation within OATA - Non-Functional Requirements Assessment Process.

- **Maintainability¹**

The capability of the software to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications. [ISO 9126]

The maintainability characteristic of the architecture is assessed in the method presented here.

- **Portability**

The capability of the software product to be transferred from one environment to another. [ISO 9126]

The portability quality characteristic is assessed in the WP 8.1.4: Define Methodology for Validation within OATA - Architecture Tactics Assessment Process.

¹ ISO 9126 has a different definition of maintainability than ISO 13236. This methodology uses ISO 9126 definition.

2.3.2 Metrics

Metrics quantify a variety of structural aspects of an architecture, which influence qualities of the architecture.

This methodology proposes several metrics that can be used for assessing the structural quality characteristics of the OATA architecture. Most of these metrics have primarily been defined for software code but this methodology proposes that they can be used for assessing a logical system architecture as well.

The relationships between quality (sub)characteristics, quality attributes and the metrics are described in Appendix A. Detailed descriptions of each selected metric can be found in Appendix B.

3 PROCESSES

3.1 Introduction

The methodology consists of three separate processes, which have to be executed in a certain order, since each process produces information that the following process requires.

The first process, "Preparation", is responsible for the planning of a validation cycle and that the Validation Team is well prepared and each team member is given assigned responsibilities. The purpose of the current validation and OATA architecture version that shall be validated must be identified. The results and other information about the validation cycle shall be documented and stored in a repository. Therefore a repository must be identified or created in the beginning of a validation cycle.

The second process, "Execution", handles the actual process of measuring metrics representing the chosen quality aspects in the OATA Logical architecture. The results of these measurements must be well documented and stored in the repository.

And finally, the last process to be executed, "Analysis and Reporting", assesses the quality of the OATA Logical architecture. The measurement results of each quality aspect are assessed and the Validation Team shall recommend necessary corrections. The validation result shall be published in one or several Validation Reports that can be distributed to the Stakeholders and the results can be used to refine the OATA architecture.

An overview of the processes and their activities are shown in Figure 2.

The processes are controlled and supported by roles, information and physical resources. An overview of these resources is shown in Figure 3.

The required information for each process is described in the process diagrams, and these diagrams also include information that is refined or created by the processes. An overview of the information flow is shown in Figure 4.

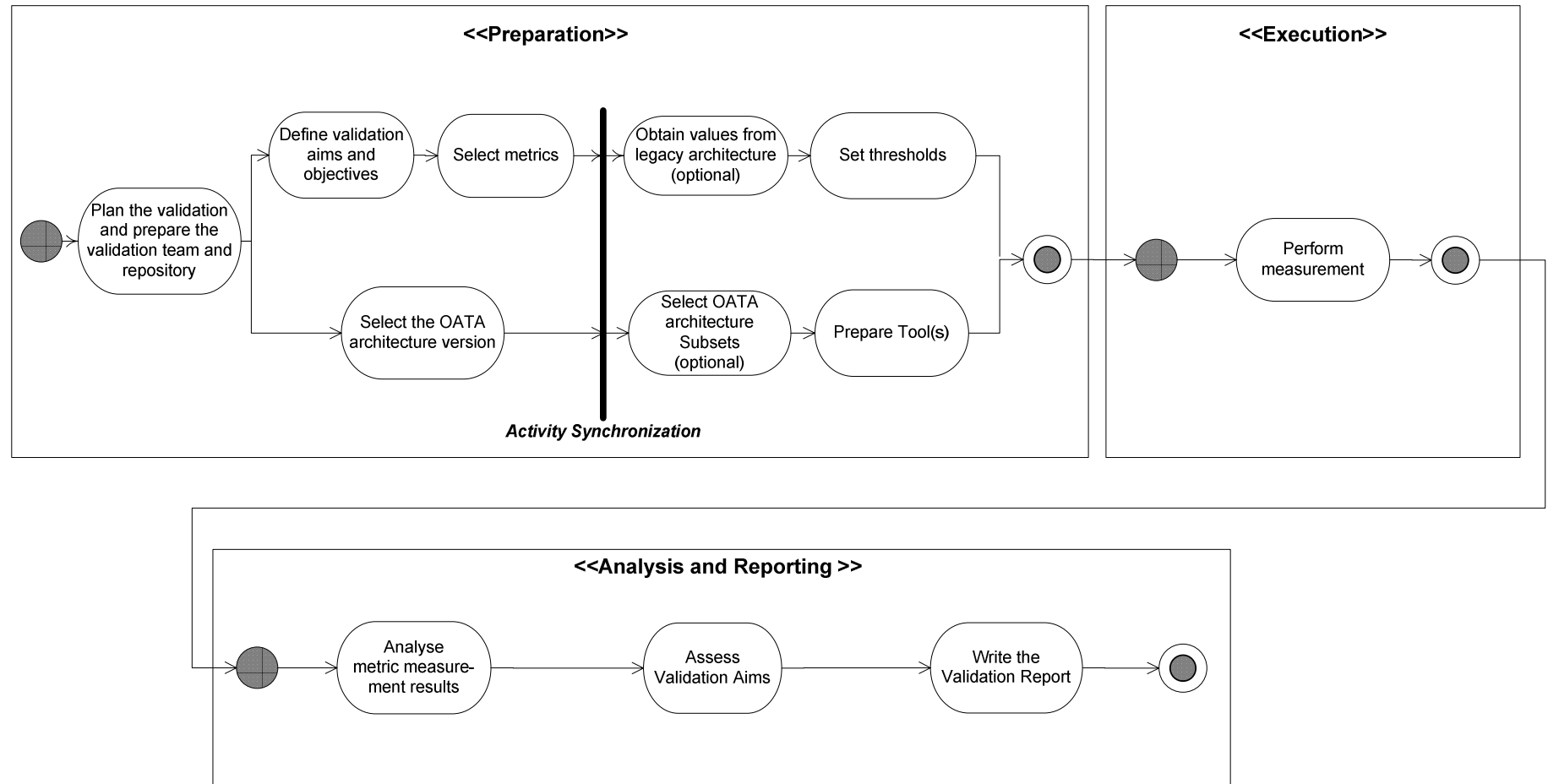


Figure 2: Process Overview

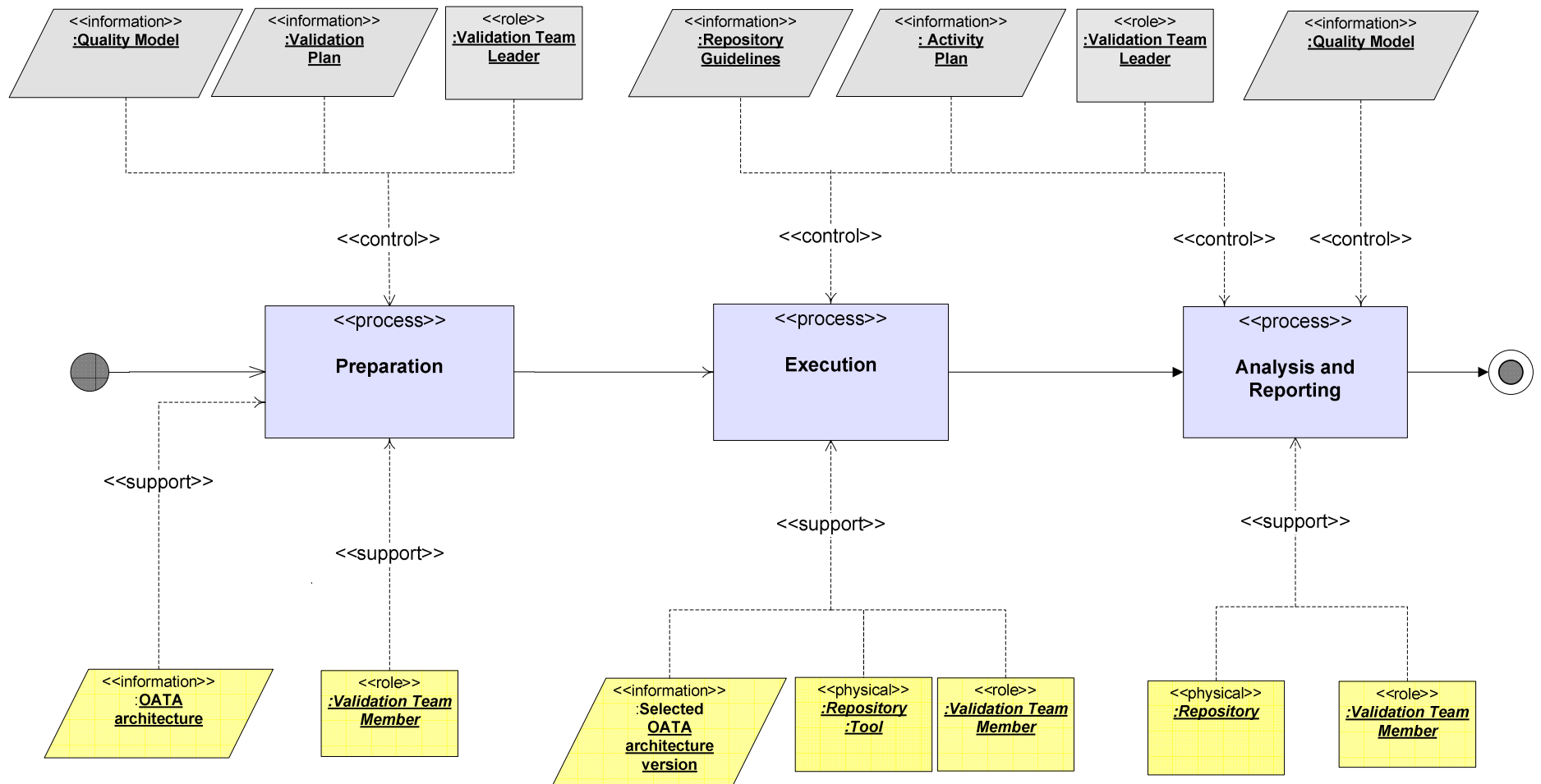


Figure 3: Resource Overview

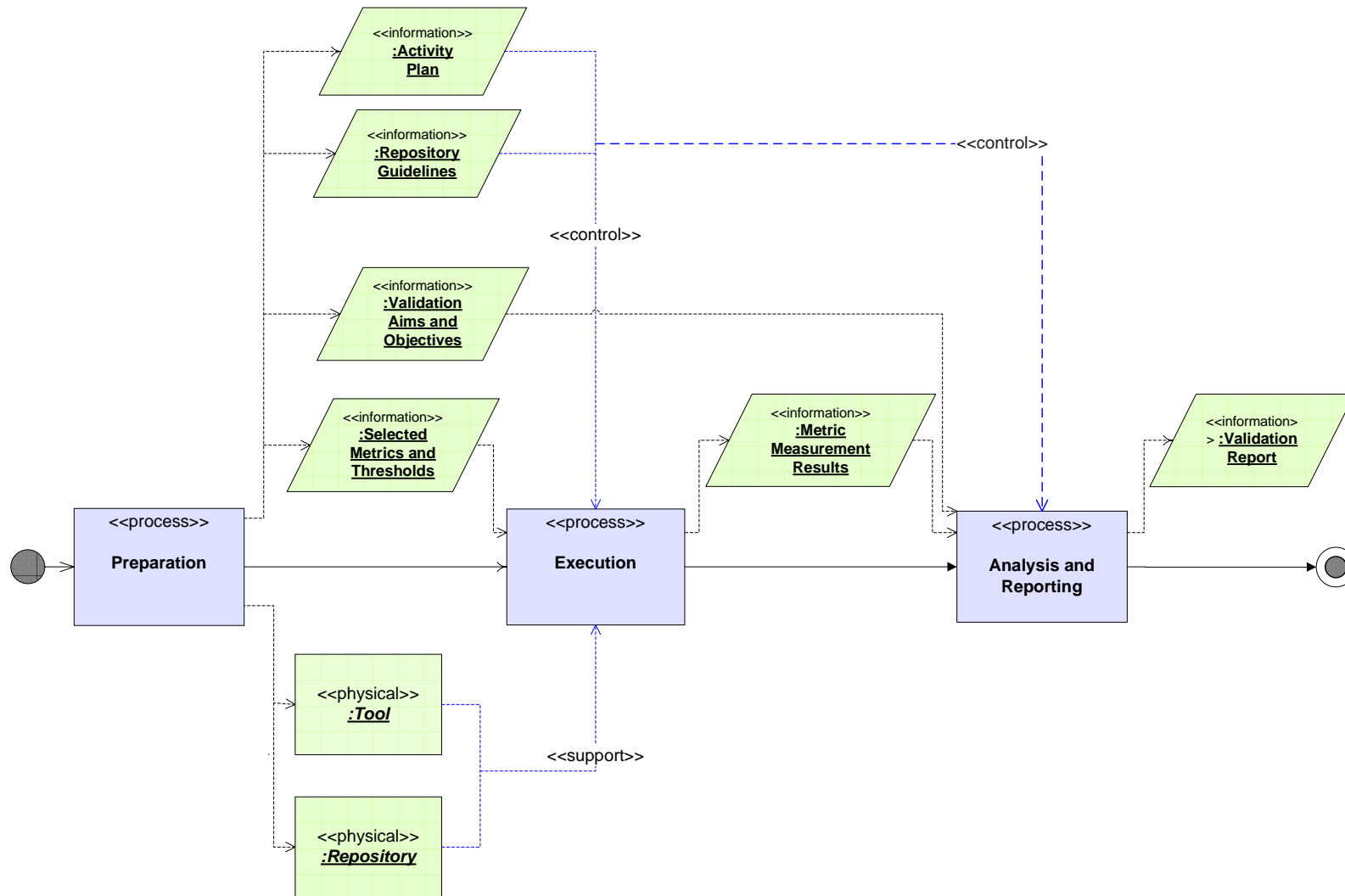


Figure 4: Information Flow Overview

3.1.1 Required roles

The table shows the different roles and their responsibility and requirements for the validation methodology.

Role	Responsibilities	Requirements
Validation Team Leader	<p>Is responsible for the control of the validation process and team management.</p> <p>Is responsible for the information flow.</p> <p>Is responsible for the Validation Aims.</p> <p>Is responsible for the Activity Plan.</p> <p>Is responsible for the Validation Reports.</p> <p>Is responsible for the Repository and the guidelines.</p>	<p>Good knowledge of the OATA project and organisation.</p> <p>Good knowledge of the validation methodology described in this document.</p>
OATA Architecture Expert	<p>Is responsible for the OATA architecture.</p>	<p>Expert knowledge about the OATA architecture.</p> <p>Full access to the OATA architecture.</p> <p>Mandate to select version to validate.</p>
Validation Expert	<p>Is responsible for the metric selection.</p> <p>Is responsible for setting the thresholds.</p>	<p>Good knowledge about metrics and their applicability.</p> <p>Good knowledge about the Quality Model.</p>
OO/UML Expert	<p>Is responsible for expert knowledge concerning object-orientation, system design and UML.</p>	<p>Expert knowledge about architectures and UML.</p> <p>Expert knowledge about object-oriented analysis.</p>
Tool Expert	<p>Is responsible for the tools which are used for measurement of metrics.</p>	<p>Good knowledge of tools for measuring metrics in an architecture</p>
Legacy Architecture Expert (Optional)	<p>Is responsible for the knowledge of legacy architecture structure.</p>	<p>Good knowledge of legacy architecture.</p>

3.2 Preparation

3.2.1 Introduction

The objective of the “Preparation” process is to prepare the necessary information and tools required in the “Execution” and the “Analysis and Reporting” processes (see Figure 5 below).

The Preparation process consists of the following activities:

- **Plan the validation cycle and prepare the validation team**

The Validation Team Leader plans the validation cycle and makes sure that the Validation Team members are available and assigns responsibilities.

- **Select the OATA architecture version**

A released version of the OATA architecture is selected to be validated during a validation cycle.

- **Define validation aims**

The Validation Team identifies a high-level expectation of the OATA architecture quality.

- **Select metrics**

The metrics that can be used to quantify the expected quality are selected for measuring.

- **Set thresholds**

For each selected metric an acceptable range is set.

- **Obtain values from legacy architectures (Optional)**

This activity is optional and its purpose is to collect values to be used as basis for setting thresholds.

- **Select OATA architecture subsets (Optional)**

The validation team can select subsets of the OATA architecture, if necessary, to be validated.

- **Prepare tools**

The tools used for measuring are selected and prepared.

During the preparation the basis for a validation cycle is set. The basis is mainly the validation aims, selected metrics and the defined thresholds. The validation aim is the basis on which the conclusions are made and presented in a validation report. The selected metrics represents the attributes that are quantified in order to conclude the validation aims. The thresholds are important because the confidence in the assessment result depends on the confidence in the thresholds. The setting of thresholds is one of the most difficult activities.

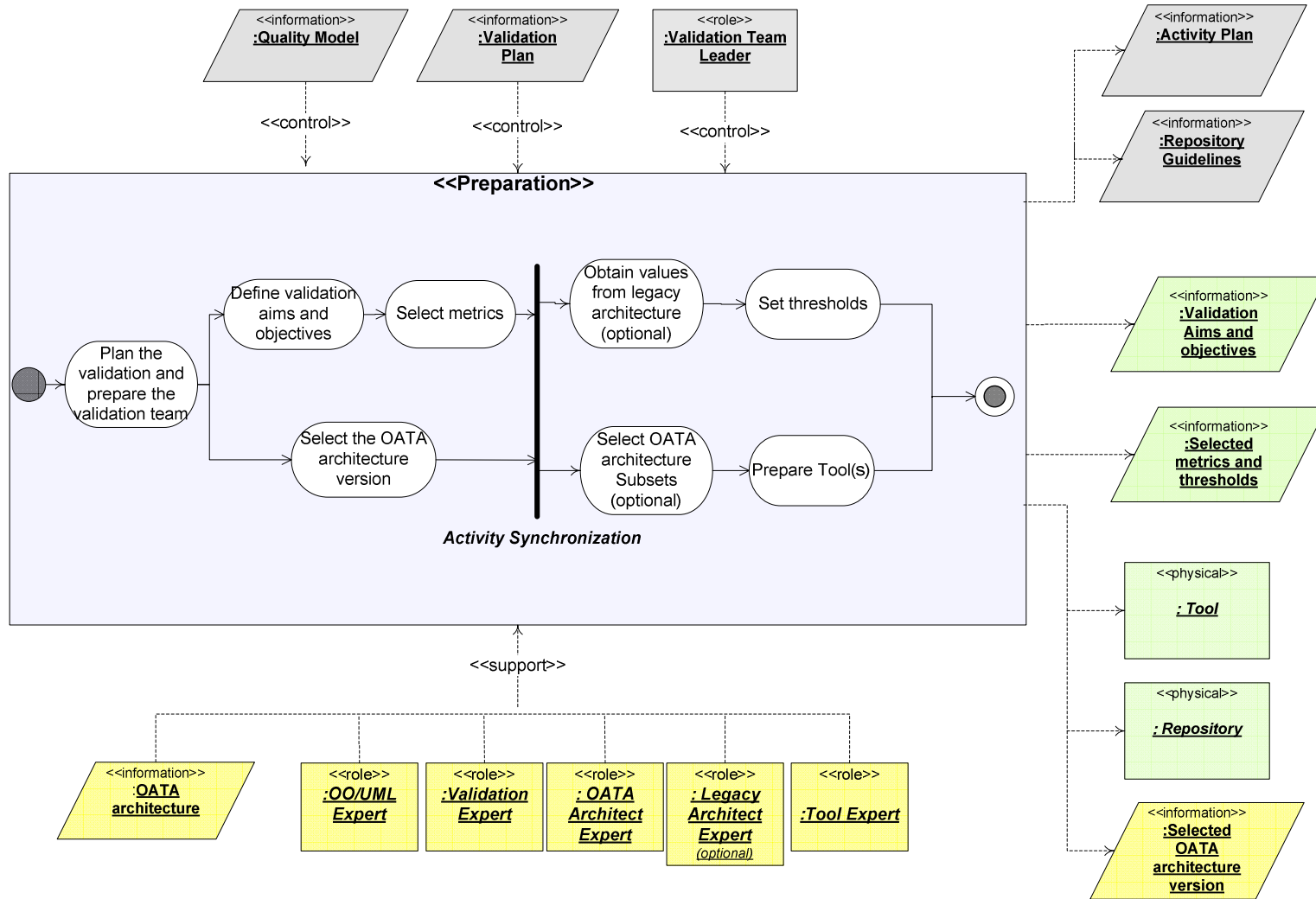


Figure 5: Preparation

3.2.2 Required resources

The table shows the planned participant resources for the activities in “Preparation”. The highlighted markings are the role responsible for the activity.

Activities	Roles					
	Validation Team Leader	OATA Architecture Expert	Validation Expert	OO/UML Expert	Tool Expert	Legacy Architecture Expert (Optional)
Plan the validation cycle and prepare the validation team	<u>X</u>					
Select the OATA architecture version		<u>X</u>				
Define validation aims	<u>X</u>	X	X			
Select metrics		X	<u>X</u>	X		
Select OATA architecture subsets (Optional)		<u>X</u>				
Obtain thresholds from legacy architecture (Optional)			<u>X</u>			X
Set thresholds		X	<u>X</u>	X		
Prepare tools		X			<u>X</u>	

3.2.3 Activity: Plan the validation cycle and prepare the validation team and repository

3.2.3.1 Objective

The objective of this activity is to ensure that required resources are available and a properly trained validation team for the assignment.

3.2.3.2 Rationale and Context

Before the validation can be initiated it is important that the Validation Team Leader ensures that the Validation Team Members are available and well prepared for the validation. To have effective use of the Validation Team Members, the Validation Team Leader should prepare a relatively detailed activity plan.

A repository should also be selected/created to be used for documentation of the validation cycle. Guidelines for its usages should also be prepared and made available for the Validation Team Members.

3.2.3.3 Description

The Validation Team leader is responsible for the planning of the validation cycle and that each team member is well prepared. The validation results must be documented and stored in a repository and the Validation Team leader is responsible for the repository.

The Validation Team Leader shall perform the following tasks:

- Plan the activities in the validation process and document in the Activity Plan.

- Introduce the validation methodology to the team members. Each Validation Team Member shall understand the processes and their activities.
- Assign roles and responsibilities to each Validation Team Member.
- Prepare a repository for validation results and create guidelines to how it shall be used.

3.2.3.4 Examples and Recommendations

A repository should be a well-structured place to store the metric measurements and other important information produced during a validation cycle. It can be for example an Excel-sheet, database or a folder in a file system (or a combination).

Following criteria's should be considered:

- The selected tools ability to store measurements in a repository.
- The results must be easy to identify.
- The measurements should be stored in the repository in order to facilitate analysis.

If a repository is already available (for example from previous validation iterations) for storing validation data, then this should be used. It is essential that the validation data is fully identifiable and traceable.

3.2.3.5 Product

The expected result of this activity is:

- Well prepared Validation Team Members with assigned responsibilities.
- An activity plan for the current validation cycle.
- A prepared repository for storage of the validation data results.
- Repository guidelines.

3.2.3.6 Resources

The required resources for this activity are shown in the table below.

Role	Responsible	Executor	Input Information	Output Information	Physical
Validation Team Leader	X	X	Validation Plan	Activity Plan	Repository
Validation Team Members				Repository guidelines	

3.2.4 Activity: Select the OATA architecture version

3.2.4.1 Objective

The objective of this activity is to select the OATA architecture version for the current validation cycle.

3.2.4.2 Rationale and Context

The development of the OATA architecture is an iterative work process with regular version releases. To ensure that the OATA architecture is not changed during the validation cycle one specific release version must be selected. This is also necessary for maintaining the traceability to the validated version for future analysis and comparisons.

3.2.4.3 Description

The OATA Architecture Expert shall select and document the OATA architecture version which shall be validated. The selected version must be clearly identified and accessible for the Validation Team.

3.2.4.4 Examples and Recommendations

The selected architecture version will refer to a specific OATA baseline, for example: OATA-2003-09-19B.

3.2.4.5 Product

The result of this activity is a selected version of the OATA architecture for the current validation cycle.

3.2.4.6 Resources

The required resources for this activity are shown in the table below.

Role	Responsible	Executor	Input Information	Output Information
OATA Architecture Expert	X	X	OATA architecture	The selected OATA architecture version

3.2.5 Activity: Define validation aims and objectives

3.2.5.1 Objective

The objective of this activity is to define validation aims and objectives for the current validation cycle.

3.2.5.2 Rationale and Context

This activity decides the purpose of the current validation cycle. All validation aims decided here will be analysed and the conclusion will be documented in the validation report.

First the validation aim has to be defined and then decomposed into more detailed objectives. These validation objectives are used to identify the metrics that needs to be measured via the quality attributes or characteristics.

3.2.5.3 Description

A validation aim shall focus on a specific high-level objective that is essential for increasing the knowledge about the OATA architecture structure quality. To understand and comprehend the Validation Aims these are decomposed into validation objectives. These objectives should address terms of quality characteristics, quality subcharacteristics or quality attributes.

The **first task** is to define the validation aims. The validation aims can be defined more or less detailed. They can focus on the whole architecture or a specific subset of the architecture. The validation aim is the high-level objective, which later will be decomposed into validation objectives.

The validation aims can be defined in terms like:

- *“The OATA architecture shall be easy to maintain”*
- *“The OATA architecture shall be understandable”*
- *“The OATA architecture shall be able to adapt to operational changes”*

The **second task** is to decompose the validation aims into smaller objectives. The purpose of validation objectives is to increase the understanding of the validation aim and the details that have to be assessed in order to make a conclusion.

The objectives should be phrased in terms like:

- *“The OATA architecture shall not be too difficult to test”*
- *“The OATA architecture shall be easy to analyse”*
- *“The OATA architecture shall be reasonably easy to change”*

The validation aims and objectives shall be documented in the repository.

3.2.5.4 Examples and Recommendations

The validation objectives should address quality characteristics, subcharacteristics or attributes defined in the Quality Model in Appendix A: “Quality Model”, but this is not mandatory. However it will make it easier to select the appropriate metrics.

The validation objectives shall cover as much as possible of the validation aim in order to increase the possibility to make a good conclusion of the validation aim.

It is recommended that a validation objective address one quality attribute.

3.2.5.5 Product

The result of this activity is a set of validation aims and objectives for the current validation cycle.

3.2.5.6 Resources

The required resources for this activity are shown in the table below.

Role	Responsible	Executor	Input Information	Output Information
Validation Team Leader	X	X	Appendix A: Quality Model	Validation Aims and objectives
OATA Architecture Expert				
Validation Expert				

3.2.6 Activity: Select metrics

3.2.6.1 Objective

The objective of this activity is to choose metrics that quantifies the detailed validation objectives originated from the Validation Aims.

3.2.6.2 Rationale and Context

This activity is required to identify metrics that are required for assessing the validation objectives.

3.2.6.3 Description

The Appendix A: "Quality Model", based on ISO 9126, is used as a tool to identify which metrics are suitable for assessing the validation objectives.

The **first task** is to identify the appropriate quality characteristics, subcharacteristics or attributes in the validation objectives.

If the validation objective addresses quality issues that cannot be identified in the Quality Model then this should be documented for future analysis how to incorporate in the quality model.

The **second** task is to select and describe the metrics that shall be measured in the OATA architecture.

Identify the metric categories that are associated with the identified quality attributes. For each metric group, use Appendix B "Metrics" to select the appropriate metrics. A metric group can include several metrics which are applicable to different levels or aspects of the OATA architecture.

A selected metric can be an original metric (like MPC) or an original metric adjusted especially for the OATA architecture.

The selected metric shall be documented together with a description of the metric calculation method in the OATA architecture. This description will be used as guidance to the tool selection/creation activity.

3.2.6.4 Examples and Recommendations

The definition of the selected metrics should not be changed from iteration to iteration in order to make it possible to make a trend analysis.

3.2.6.5 Product

The product of this activity is a set of selected metrics with a description/guidance to the tool selection activity.

3.2.6.6 Resources

The required resources for this activity are shown in the table below.

Role	Responsible	Executor	Input Information	Output Information
Validation Expert	X	X	Appendix B: Metrics	Selected metrics
OATA Architecture Expert		X	Appendix A: Quality Model	
OO/UML Expert		X		

3.2.7 Optional Activity: Select OATA architecture subsets

3.2.7.1 Objective

The objective of this activity is to select relevant subsets from the OATA architecture, which shall be validated.

If the decision is to validate the complete architecture, this activity does not need to be performed.

3.2.7.2 Rationale and Context

Metrics are applicable to one or more levels in the OATA architecture and the validation aims may consider only a subset of the OATA architecture.

Optionally subsets of the OATA architecture that shall be validated shall be selected since this can make the activity “Prepare tools” more efficient.

The selection of subsets may be restricted by the validation aims/objectives. The metrics may also restrict the selection since metrics are applicable to different levels in the architecture.

3.2.7.3 Description

The OATA Architecture Expert shall for each metric select relevant subset in the OATA architecture. The selected subset must be documented.

The result shall be combined and used as guidance to the tool selection activity.

3.2.7.4 Examples and Recommendations

For example, it may be decided to validate a particular cluster, such as “Avionics”, or a single module, such as “Flight Guidance”.

3.2.7.5 Product

The product of the activity is relevant subsets of the OATA architecture for each metric.

3.2.7.6 Resources

The required resources for this activity are shown in the table below.

Role	Responsible	Executor	Input Information	Output Information
OATA Architecture Expert	X	X	Selected OATA Architecture version	Selected subsets of OATA Architecture version
		X	Selected Metrics	

3.2.8 Optional Activity: Obtain values from legacy architectures

3.2.8.1 Objective

The objective of this activity is to obtain metric measurement values from legacy architecture in order to improve the thresholds for the metrics.

This is a difficult task and when considered not possible, this activity does not need to be performed.

3.2.8.2 Rationale and Context

By obtaining values from legacy architecture the credibility of thresholds will be increased.

3.2.8.3 Description

The task in this activity is to investigate if values from legacy architecture are accessible. If these kinds of values are available then they can be a good basis for setting threshold values. Using values from legacy architecture makes it possible to compare the OATA architecture with an existing architecture.

If values are available, then these shall be documented in the repository and used as input to the set threshold activity.

When the OATA architecture has been validated then these values can be used as basis for setting thresholds for the current validation cycle and therefore be considered as legacy architecture.

3.2.8.4 Examples and Recommendations

If values from legacy architecture are available it is recommended to analyse if these values have been obtained from a similar architectural level as the OATA architecture level. Otherwise the values can be difficult to use.

3.2.8.5 Product

The product of this activity is set of metric measurement values obtained from legacy architecture.

3.2.8.6 Resources

The required resources for this activity are shown in the table below.

Role	Responsible	Executor	Input Information	Output Information
Validation Expert	X	X	Appendix B: Metrics	Legacy values architecture
Legacy Architecture Expert		X	Selected Metrics	
			Results from previous validation cycles	

3.2.9 Activity: Set thresholds

3.2.9.1 Objective

The objective of this activity is to set threshold for each selected metric.

3.2.9.2 Rationale and Context

This activity is one of the most important to gain confidence in the validation result. The thresholds that are decided in this activity will form the base for the analysis process. During the analysis the measured metric values will be compared with the thresholds in order to assess the quality.

The reason why this activity is performed during “Preparation” is to avoid influence and discussion during measuring and analysis of the metric measurements results.

3.2.9.3 Description

In order to gain an assessment result of a metric, a threshold has to be set. The threshold defines the range (or ranges) within the measured value is considered to be acceptable. The metric measurement value will be compared with the set threshold and the resulting value will finally be used for the assessment of validation objectives.

The task is to set one threshold for each metric. Basic data that can be used to set a threshold is found in Appendix B: “Metrics”, chapter “Desired value”, for most of the metrics. A better choice is to use values obtained from legacy architectures if available. As legacy architecture the OATA architecture can be used in future iterations. If such values are not available they can be obtained from:

- Basic data for a threshold (guidelines will be included in Appendix B: “Detailed metric description”).
- Reasonable values (expert knowledge and experience).

3.2.9.4 Examples and Recommendation

The threshold value can be set as a range within the metric value will be considered as acceptable. Otherwise the metric value will be considered as not accepted. For example a threshold can be set to the range 3-15 and if the metric value is within this range it is considered as “accepted/pass”, otherwise it is considered as “not accepted/fail”.

The threshold values can also be set to several ranges. One range that is considered as “Good”, another range considered as “Moderate”, otherwise it is considered as “Poor”. For example the following ranges can be set:

- Range: 5-10 Good or 3
- Range: 3-15 Moderate or 2
- Otherwise Poor or 1

If statistics are elaborated and these statistical values shall be assessed then thresholds must also be set for these kinds of values. For example following ranges can be set:

- Range for average per module: 5-10
- Range for average per cluster: 0-5

The thresholds can be adjusted when one or several validation cycles have been performed.

3.2.9.5 Product

The product of this activity is one threshold for each selected metric.

An optional product is thresholds for statistical values.

3.2.9.6 Resources

The required resources for this activity are shown in the table below.

Role	Responsible	Executor	Input Information	Output Information
OATA Architecture Expert		X	Appendix B: Metrics	Set threshold for each selected metric
Validation Expert	<u>X</u>	X	Selected metrics	Optional: Thresholds for statistical values
OO/UML Expert		X	Optional: Values from legacy architecture	

3.2.10 Activity: Prepare tools

3.2.10.1 Objective

The objective is to select and configure appropriate tools to be used for measuring metrics.

3.2.10.2 Rationale and Context

Depending on the type of metric a tool has to be either created or purchased. The problem is that there isn't one tool that is suitable to measure all kinds of metrics therefore there probably will be a combination of several tools or scripts.

3.2.10.3 Description

It is recommended to make a list of requirements for the tool. These requirements shall consider type of metric to measure and the selected OATA architecture version. The OATA architecture is modelled in Rose and therefore the Rose model shall either work as input to the tool or the model will have to be exported to a suitable format.

The **first** task for the Validation Team is to select a suitable tool for each metric. A tool can be:

- An application for measuring metrics in UML
A list of suggested tools are found in Appendix E: “Available Tools List”
- A script
- Manually measuring

In order to select the most effective tool the Validation Team must estimate the time required for learning, installing, testing and executing each suitable tool for the selected metrics. The number of different tools should also be minimised as far as possible. It is also important to consider that the tool and its configuration should be repeatable in future iterations.

The **second** task is to acquire or create the selected tool(s) using the tool requirements list.

The **third and optional** task is to prepare the OATA architecture for usage by the selected tool. This task is dependent on the input requirements for the tool/script.

Each tool with a specific configuration shall be uniquely identifiable and documented.

3.2.10.4 Examples and Recommendations

The tool should support the measurement of absolute values of metrics. If the selected tool also supports the elaboration of statistics it will be useful during the analysis.

Another important requirement of tools that should be considered is the possibility to make automatic comparisons between the metric measurements values and the set thresholds.

The OATA Architecture expert is responsible for the preparation of the OATA architecture.

UML 2.0 specifies a scripting language, OCL, to be used on UML. Examples are given in ref [MOOSE 03] for some metrics.

3.2.10.5 Product

The product of this activity is installed and tested tools suitable to measure the selected metrics and the prepared OATA architecture. The selected or created tools and the prepared OATA architecture shall be documented.

3.2.10.6 Resources

The required resources for this activity are shown in the table below.

Role	Responsible	Executor	Input Information	Output Information	Physical
Tool Expert	X	X	Appendix E: Available Tools List	List of tools	Tools
OATA Architecture Expert		X	Selected OATA Architecture version		Prepared OATA architecture (Optional)

3.3 Execution

3.3.1 Introduction

The objective of the Execution process is to measure the selected metrics. The Execution process consists of the activity “Perform measurement” which includes the following tasks (see Figure 6 below):

- **Perform measurement**

When the tools and the OATA architecture have been prepared the tools are run to collect the metric measurement values.

- **Save the result in repository**

The results of the measurements are stored in the repository.

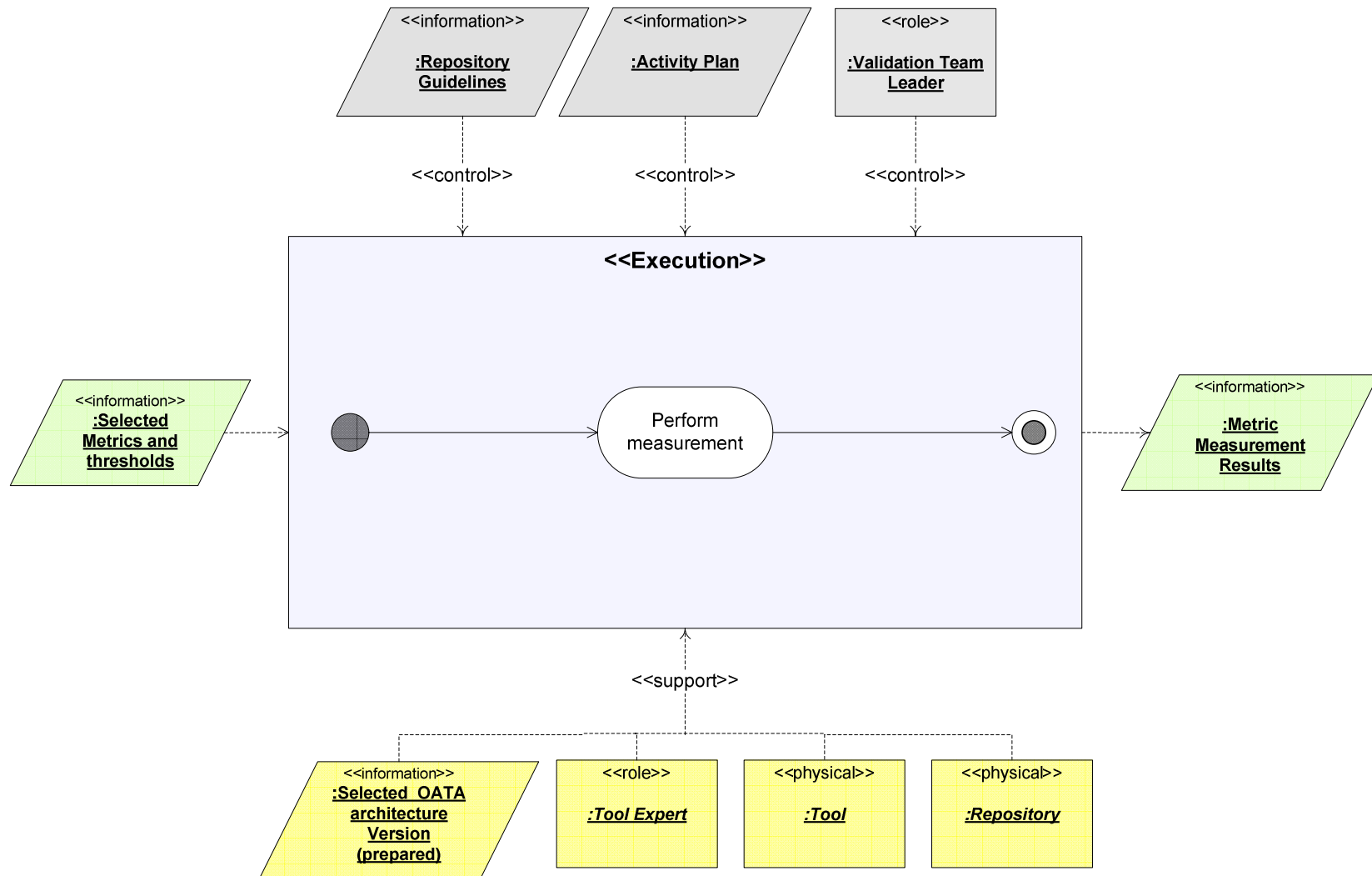


Figure 6: Execution

3.3.2 Required resources

The table shows the planned participating resources for the activity in “Execution”.
The highlighted marking is the role responsible for the activity.

Activities	Roles					
	Validation Team Leader	OATA Architecture Expert	Validation Expert	OO/UML Expert	Tool Expert	Legacy Architecture Expert
Perform Measurement					X	

3.3.3 Activity: Perform Measurement

3.3.3.1 Objective

The objective of this activity is to collect the metric values from the selected OATA architecture version.

3.3.3.2 Rationale and Context

This activity is required to collect the metric values for later analysis.

3.3.3.3 Description

The task is to perform measurement of the selected metrics.

For each metric;

- Execute the tool to collect measurements for the metric. See manual/help for the selected tool.
- Save the results in the repository and follow the guidelines for the repository use created in “Preparation”. All measured values, both from a tool/script and from manually measurements, shall be stored in the repository.

3.3.3.4 Examples and Recommendations

No information.

3.3.3.5 Product

The product from this activity is measured metric values uniquely identified and stored in repository.

3.3.3.6 Resources

The required resources for this activity are shown in the table below.

Role	Responsible	Executor	Input Information	Output Information	Physical
Tool Expert	X	X	Selected Metrics	Metric Measurements Values	Tools
			Repository Guidelines		Repository

3.4 Analysis and Reporting

3.4.1 Introduction

The objective of the “Analysis and Reporting” process is to analyse the measurements in order to assess the quality of OATA architecture. The process consists of the following activities (see Figure 7 below):

- **Analyse metric measurement results**

The measurements results are analysed by the Validation Team Members.

- **Assess Validation Aims**

The conclusions of the validation results are made.

- **Write the Validation Reports**

The results and conclusions of the validation cycle are documented in a Validation Report(s).

The analysis begins with the comparison between measured metrics and the thresholds. Next step is to assess the quality referred by the validation objectives, to assess the validation aims and to make the conclusions.

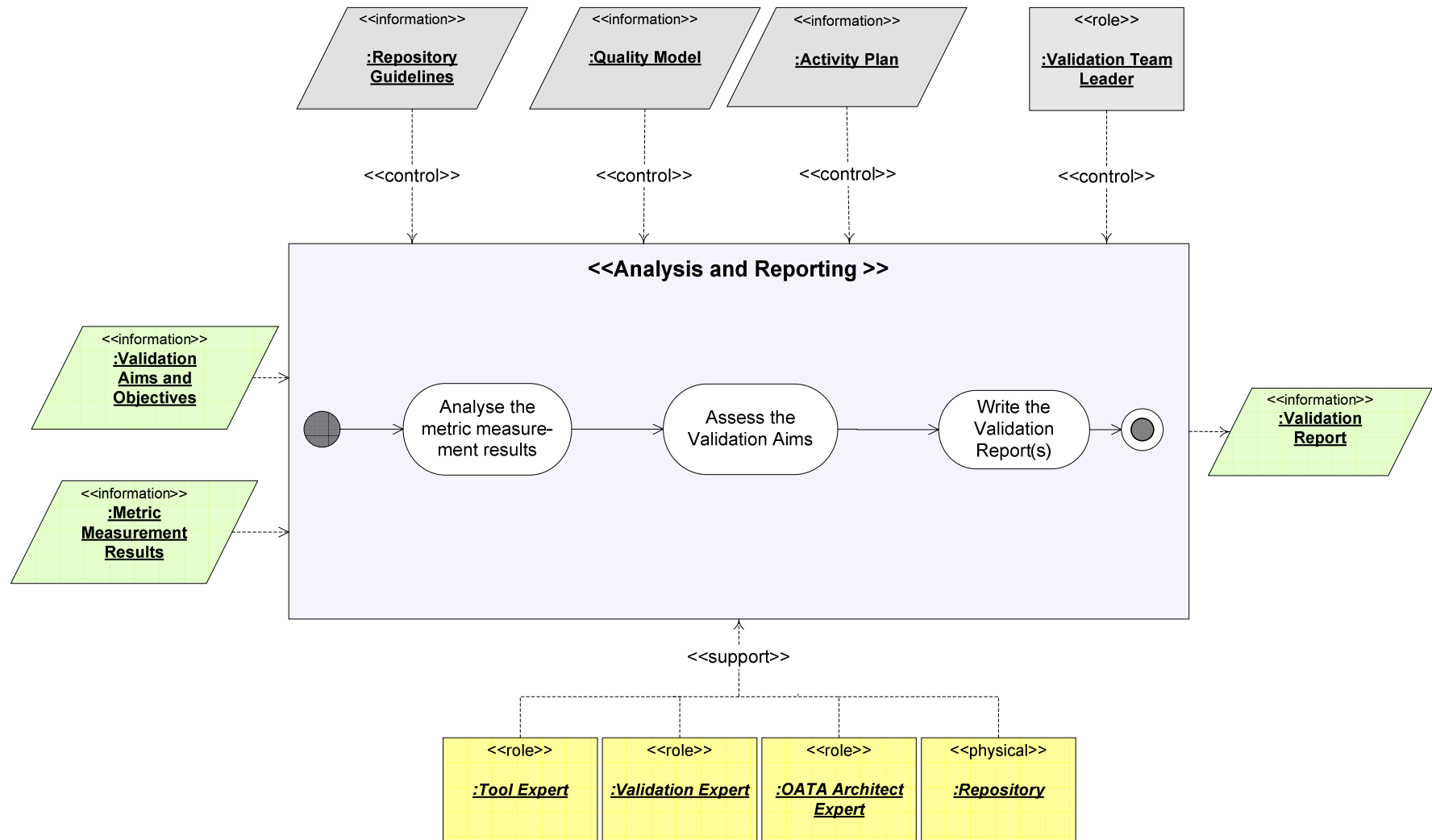


Figure 7: Analysis and Reporting

3.4.2 Required resources

The table shows the planned participating resources for the activities in “Analysis and Reporting”.

The highlighted markings are the role responsible for the activity.

Activities	Roles					
	Validation Team Leader	OO/UML Expert	OATA Architecture Expert	Validation Expert	Tool Expert	Legacy Architecture Expert
Analyse the metric measurement results				X	X	
Assess the Validation Aims	X			X		
Write the validation report(s)	X			X		

3.4.3 Activity: Analyse the metric measurement results

3.4.3.1 Objective

The objective of this activity is to compare each measured metric to the thresholds.

3.4.3.2 Rationale and Context

This activity is required to decide if the measured value of the metric is within the threshold or not. This is the first analysis step in order to assess the validation objectives.

3.4.3.3 Description

The task is to analyse each measured value of a metric by comparing the value with the decided threshold.

This can be done manually, but for increased efficiency, a tool should be used for this task. The tool can be the same tool used for the measuring or a separate analysis tool.

3.4.3.4 Examples and Recommendations

If the tool can support thresholds and compare them with the measurements results, this activity could very well be combined with the measuring of metrics.

The credibility of the result from this activity is completely depending on the credibility of the thresholds. This credibility should be considered when the conclusion of the validation aim is made.

3.4.3.5 Product

The product of this activity is the result from comparison between measurement and threshold for each metric.

3.4.3.6 Resources

The required resources for this activity are shown in the table below.

Role	Responsible	Executor	Input Information	Output Information	Physical
Tool Expert or Validation Expert	X	X	Metric measurements	Comparison between measurements and thresholds	Repository
			List of thresholds per metric		

3.4.4 Activity: Assess the Validation Aims

3.4.4.1 Objective

The objective of this activity is to assess and conclude the Validation Aims.

3.4.4.2 Rationale and Context

When the metric measuring and analysis are done, it is possible to assess the validation aims.

This activity is where the actual validation of the OATA architecture is performed.

3.4.4.3 Description

The **first** task is to assess the quality referred by each validation objective. Assessing the metric assessment results for those metrics that influences the referred quality does this task. For each validation objective there will be one assessment result. To assist this task, use the documentation from activity "Select metrics" in preparation.

The defined subcharacteristics have different impact on characteristics and ISO 9126 includes a suggested weight for each quality subcharacteristics, see Appendix A. This can be used as guidance during the assessment of the quality.

The **second** task is to make conclusion for each validation aim. Using the assessment results of each validation objective that are related to the validation aim does these conclusions.

Recommendations and comments should be documented together with the conclusions. When the conclusion is that the validation aim is not fulfilled then validation team should recommend actions to be done.

3.4.4.4 Examples and Recommendations

The recommendation is to have the same weight for all metrics for the first iteration, but this could be refined for future iterations. In reality, different metric categories have different impact on quality attributes. Weighting of metric categories for each attribute might be considered in the future as knowledge increases. For the first iteration it's recommended that weigh the metric categories equal.

The validation team can use the results of previous validation cycle in order to make a trend analysis whether the architecture has improved or not.

3.4.4.5 Product

The product of this activity is the assessments and conclusions of the validation aims documented with recommendations and comments.

3.4.4.6 Resources

The required resources for this activity are shown in the table below.

Role	Responsible	Executor	Input Information	Output Information
Validation Team Leader	X			
Validation Expert		X	Comparison between measurements and thresholds	Conclusion of Validation Aims

3.4.5 Activity: Write the validation report(s)

3.4.5.1 Objective

The objective of this activity is to finalize the validation process of the OATA architecture by writing a validation report (or several).

3.4.5.2 Rationale and Context

The Validation Reports can be at different levels for different readers and they can cover different parts of the quality model.

3.4.5.3 Description

The validation team members shall decide to whom the validation report is aimed to and the context of each report.

The Validation Team Members shall report about the performed validation, the validation results and the made conclusions. The reports should also include information about "lessons-learned" during the validation in order to improve the future validation activities.

The final task in the validation iteration is to create and distribute the reports.

3.4.5.4 Examples and Recommendations

A Validation Report should include;

- Information about the Validation Team
- Validation Aims and their objectives
- Selected OATA architecture version and tool(s)
- Selected metrics and the defined thresholds for each metric
- The assessment of the metric measurements results
- Conclusion of validation objectives and validation aims
- Recommendation of corrective actions
- A lessons-learned section

3.4.5.5 Product

The product of this activity is one or more Validation Reports.

3.4.5.6 Resources

The required resources for this activity are shown in the table below.

Role	Responsible	Executer	Input Information	Output Information
Validation Team Leader	X	X	Conclusions on the Validation Aims	Validation Reports
Validation Expert		X		

4 APPENDIX A: QUALITY MODEL

4.1 Introduction

It is necessary to clearly state the context of quality and metrics before the selection and definition of metrics. This context is usually known as the Quality Model. A Quality Model is a model that considers a set of quality characteristics that can be assessed through a set of quality attributes. These quality attributes are quantified through a set of metrics. Not all the available metrics are suitable for all quality attributes and related characteristics and it is therefore necessary to make a decision on the appropriate quality characteristics and metrics, according to the state-of-the-art or previous experience in the field.

4.2 Overview of Quality Models and Proposed Quality Model

A Quality Model is a model that represents different kinds of quality characteristics and attributes of a product.

Several Quality Models were initially taken into account but only one of them has finally been proposed for use in the assessment of OATA architecture structure. The considered Quality Models are listed below:

- Factor-Criteria-Metrics Model [MacCall 77], [Boehm 78]
- REBOOT [Karlsson 95]
- Dromey's Quality Model Framework [Dromey 95]
- QMOOD (Quality Model for Object-Oriented Design) [Bansiva,Davis 97]
- ISO/IEC 9126 Quality Model [ISO 9126]

The ISO/IEC 9126 Quality Model represents an international effort to collect the ideas and experiences related to the quality of a software system over the last decades. This is the main reason why it was finally proposed as basis for the validation of the quality of OATA architecture structure. All the other Quality Models that were initially taken into account are included directly or indirectly within the ISO/IEC 9126 standard.

The ISO/EIC 9126 Quality Model is defined as a two-part model for software product quality:

1. internal and external quality, and
2. quality in use.

The first part of the quality model describes quality characteristics, which can be used for assessing internal and external quality of a software product. These quality characteristics are subdivided into quality subcharacteristics that are influenced by one or more internal and external quality attributes. The quality attributes can be quantified by internal or external metrics.

- Internal metrics can be applied to a non-executable software product (such as a specification or source code) during designing and coding. Internal metrics have the benefit that they are able to evaluate software product quality and address quality issues early in the lifecycle, before the software product becomes executable.
- External metrics use measures of a software product derived from measures of the behaviour of the system of which it is a part, by testing, operating and observing the executable software or system.

The second part of the quality model addresses quality in use, which is the user's view of the quality of a system. It is measured in terms of the result of using the software, rather than properties of the software itself.

Since OATA WP 8.1.2 assesses the quality of an architecture structure (i.e. non-executable) the proposed Quality Model considers the quality characteristics and subcharacteristics appropriate for assessing the internal quality of an architecture structure.

4.3 Description of ISO/IEC 9126 Quality Characteristics

ISO/IEC 9126 specifies six quality characteristics for internal and external quality, which are further divided into subcharacteristics. The six relevant characteristics to define and assess the internal and external quality of a software product are:

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

One key aspect of these characteristics is that they are reasonably independent. For this reason they cover a wide range of the quality aspects of the productive process of a software product. The standard also identifies subcharacteristics to be taken into account while assessing each characteristic.

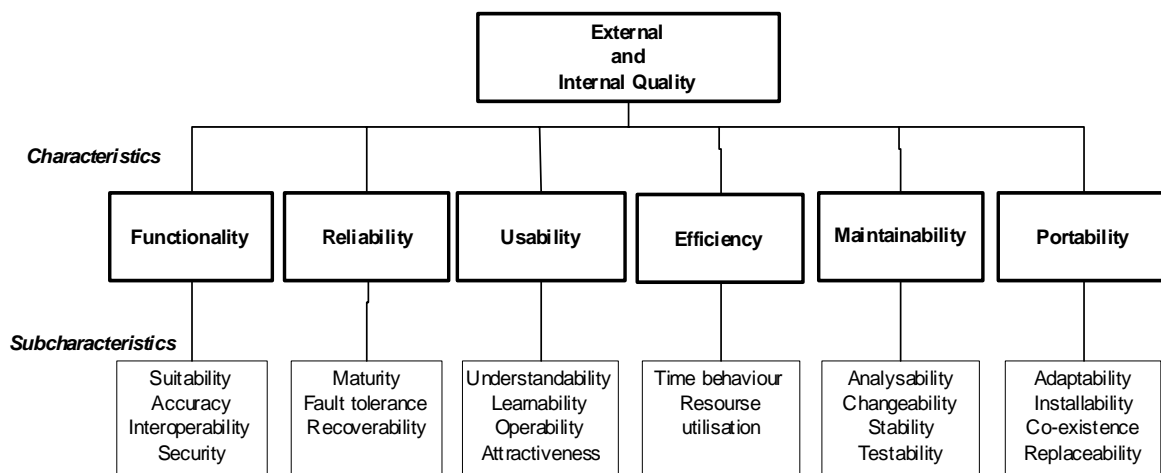


Figure 8: ISO/IEC 9126 Quality Model for External and Internal Quality

4.3.1 Definitions of characteristics

Quality Characteristics and Subcharacteristics [ISO/IEC 9126-1]	
<p>1. FUNCTIONALITY</p> <p>Characteristics relating to achievement of the basic purpose for which the software is being engineered.</p> <p>Definition:</p> <p>The capability of the software product to provide functions which meet stated and implied needs when the software is used under specified conditions.</p> <p>Note 1: This characteristic is concerned with what the software does to fulfil its needs, whereas the other characteristics are mainly concerned with when and how it fulfils needs.</p> <p>Subcharacteristics and their definitions:</p> <ul style="list-style-type: none"> • Suitability: The capability of the software to provide an appropriate set of functions for specified tasks and user objectives. • Accuracy: The capability of the software product to provide the right or agreed results or effects with the needed degree of precision. • Interoperability: The capability of the software product to interact with one or more specified systems. • Security: The capability of the software to protect information and data so that unauthorised persons or systems cannot read or modify them and authorised persons or systems are not denied access to them. 	
<p>2. RELIABILITY</p> <p>Characteristics relating to capability of software to maintain its level of performance under stated conditions for a stated period of time.</p> <p>Definition:</p> <p>The capability of the software product to maintain a specified level of performance when used under specified conditions.</p> <p>Subcharacteristics and their definitions:</p> <ul style="list-style-type: none"> • Maturity: The capability of the software product to avoid failures as a result of faults in the software. • Fault tolerance: The capability of the software product to maintain a specified level of performance in cases of software faults or of infringement of its specified interface. • Recoverability: The capability of the software product to re-establish a specified level of performance and recover the data directly affected in the case of a failure. 	

Quality Characteristics and Subcharacteristics**[ISO/IEC 9126-1]****3. USABILITY**

Characteristics relating to the effort needed for use, and on the individual assessment of such use, by a stated or implied set of users.

Definition:

The capability of the software product to adhere to standards, conventions or regulations relating to reliability.

Subcharacteristics and their definitions:

- **Understandability:** The capability of the software product to enable the user to understand whether the software is suitable, and how it can be used for particular tasks and conditions of use.
- **Learnability:** The capability of the software product to enable the user to learn its application.
- **Operability:** The capability of the software product to enable the user to operate it and control it.
- **Attractiveness:** The capability of the software to be attractive to the user.

4. EFFICIENCY

Characteristic related to the relationship between the level of performance of the software and the amount of resources used, under stated conditions.

Definition:

The capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions.

Subcharacteristics and their definitions:

- **Time Behaviour:** The capability of the software product to provide appropriate response and processing times and throughput rates when performing its function, under stated conditions.
- **Resource Utilization:** The capability of the software product to use appropriate amounts and types of resources when the software performs its function under stated conditions.

Quality Characteristics and Subcharacteristics	
[ISO/IEC 9126-1]	
5. MAINTAINABILITY	<p>Characteristics related effort needed to make modifications, including corrections, improvements or adaptation of software to changes in environment, requirements and functional specifications.</p> <p>Definition: The capability of the software product to be modified. Modifications may include corrections, improvements or adaptation of the software to changes in environment, and in requirements and functional specifications.</p> <p>Subcharacteristics and their definitions:</p> <ul style="list-style-type: none"> • Analysability: The capability of the software product to be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified to be identified. • Changeability: The capability of the software product to enable a specified modification to be implemented. • Stability: The capability of the software product to avoid unexpected effects from modifications of the software. • Testability: The capability of the software product to enable modified software to be validated.
6. PORTABILITY	<p>Characteristics related to the ability to transfer the software from one organisation or hardware or software environment to another.</p> <p>Definition: The capability of the software product to be transferred from one environment to another.</p> <p>Subcharacteristics and their definitions:</p> <ul style="list-style-type: none"> • Adaptability: The capability of the software product to be adapted for different specified environments without applying actions or means other than those provided for this purpose for the software considered. • Installability: The capability of the software product to be installed in a specified environment. • Co-existence: The capability of a software product to co-exist with other independent software in common environment sharing common resources. • Replaceability: The capability of the software product to be used in place of another specified software product for the same purpose in the same environment.

Table 1: ISO/IEC 9126 Quality Concepts Definitions

4.4 The OATA Architecture Structure Internal Quality Characteristics

Depending on the nature of the OATA architecture and the purpose of WP 8.1.2 not all characteristics are considered to be relevant to assess. The following table describes the ISO/IEC 9126 quality characteristics relevancy to WP 8.1.2.

Quality characteristic	Quality subcharacteristic	Relevancy
Functionality	Suitability	The functionality quality characteristic is included in the WP 8.1.1: Define Methodology for Validation within OATA - Architecture Compliance Assessment Process. <i>WP 8.1.2 will not include this quality characteristic.</i>
	Accuracy	
	Interoperability	
	Security	
Reliability	Maturity	The reliability quality characteristic is included in the WP 8.1.3: Define Methodology for Validation within OATA - Non-Functional Requirements Assessment Process. <i>WP 8.1.2 will not include this quality characteristic.</i>
	Fault tolerance	
	Recoverability	
Usability ²	Understandability	WP 8.1.2 will assess the subcharacteristic understandability with the definition that it is the architecture's understandability that will be assessed. WP 8.1.2 will not assess Learnability, Operability, Attractiveness since these characteristics are user-oriented and no suitable metrics have been found.
	Learnability	
	Operability	
	Attractiveness	
Maintainability	Analysability	WP 8.1.2 shall assess maintainability and its entire set of subcharacteristics.
	Changeability	
	Stability	
	Testability	
Efficiency	Time Behaviour	The efficiency characteristic will be included in WP 8.1.3: Define Methodology for Validation within OATA - Non-Functional Requirements Assessment Process. <i>WP 8.1.2 will not include this quality characteristic.</i>
	Resource Utilization	
Portability	Adaptability	The portability characteristic will be included in WP 8.1.4: Define Methodology for Validation within OATA - Architecture Tactics Assessment Process. <i>WP 8.1.2 will not include this quality characteristic.</i>
	Installability	
	Co-existence	
	Replaceability	

Table 2: Internal Quality characteristics

² Assumes that the OATA architecture is the product

4.5 Weight table for subcharacteristics

ISO/IEC 9126 proposes relative weights to be assigned to the subcharacteristics (see Table 2 below). This allows the evaluators to focus their efforts on the most important subcharacteristics.

External & Internal Quality		
CHARACTERISTIC	SUBCHARACTERISTIC	WEIGHT (High/Medium/Low)
Functionality	Suitability	H
	Accuracy	H
	Interoperability	L
	Security	L
	Compliance	M
Reliability	Maturity (hardware/software/data)	L
	Fault tolerance	L
	Recoverability (data, process, technology)	H
	Compliance	H
Usability	Understandability	M
	Learnability	L
	Operability	H
	Attractiveness	M
	Compliance	H
Efficiency	Time behaviour	H
	Resource utilization	H
	Compliance	H
Maintainability	Analyzability	H
	Changeability	M
	Stability	L
	Testability	M
	Compliance	H
Portability	Adaptability	H
	Installability	L
	Co-existence	H
	Replaceability	M
	Compliance	H

Table 3: Subcharacteristics weight table

4.6 Proposed OATA Quality Model

ISO/IEC 9126 defines that, for each characteristic and subcharacteristic, the capability of the software is determined by a set of attributes that can be measured by metrics. See Figure 9: Quality Meta Model.

In WP 8.1.2 the product is the OATA architecture itself and WP 8.1.2 addresses the structure of that architecture. Therefore the selected metrics will measure the internal quality attributes relating to the structure of an architecture only.

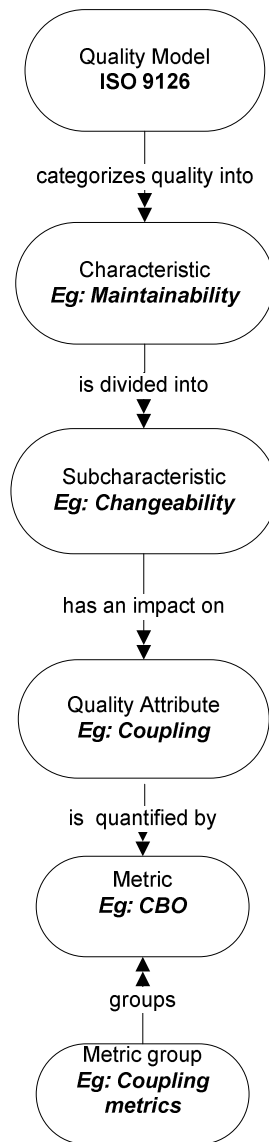


Figure 9: Quality Meta Model

4.6.1 Proposed Quality Model

The quality characteristics and subcharacteristics defined in the ISO 9126 shall be used for assessing the quality in the OATA Logical architecture.

Quality attributes are used to help identify which metrics can quantify the quality (sub)characteristics. However, the quality attributes, and associated metrics to be used, are not detailed in the ISO 9126 standard. WP 8.1.2 has examined best practices in this area and developed a proposal of the association between the characteristics, attributes and the different metrics, shown in Figure 10 below.

Note that a quality subcharacteristic has an impact on several quality attributes; a quality attribute may in turn be impacted by several quality subcharacteristics – i.e. there is many to many relationship. This is also true of the relationship between quality attributes and metrics.

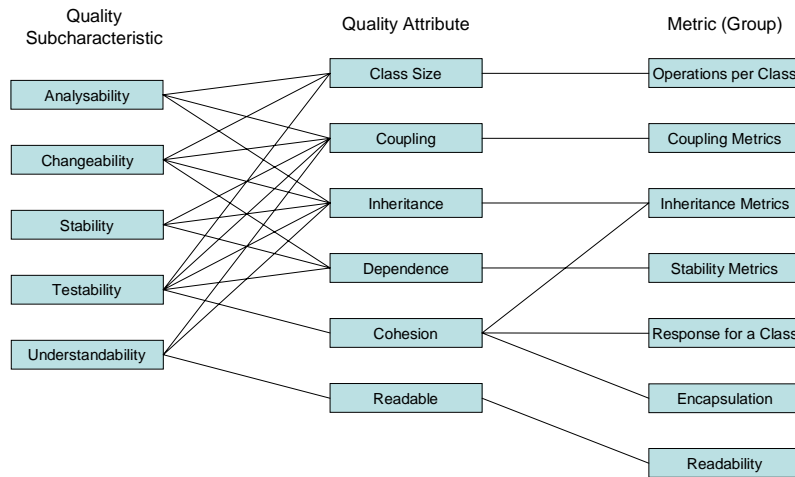


Figure 10: Quality Attribute Relationships

Table 4: Proposed Quality Model uses these relationships to identify for each quality characteristic and subcharacteristic the most relevant metric group(s). This table can be used as guidance during the activity of selecting metrics.

Quality characteristic	Quality subcharacteristic	Quality attribute	Metric group
Maintainability	Analysability	Class size	Operations per class
		Coupling	Coupling metrics
		Inheritance	Inheritance metrics
	Changeability	Class size	Operations per class
		Coupling	Coupling metrics
		Inheritance	Inheritance metrics
		Dependence	Stability metrics
	Stability	Coupling	Coupling metrics
		Inheritance	Inheritance metrics
		Dependence	Stability metrics
	Testability	Class size	Operations per class
		Coupling	Coupling metrics
		Inheritance	Inheritance metrics
		Dependence	Stability metrics
Cohesion		Response for a class	
		Inheritance metrics	
	Encapsulation		
Usability	Understandability ³	Readable	Readability metrics
		Coupling	Coupling metrics
		Inheritance	Inheritance metrics

Table 4: Proposed Quality Model

³ Understandability of the OATA architecture itself

Note that the above figure and table are only a proposal. During validation iterations the validation team may discover other significant quality attributes that are not included. These quality attributes should be added and associated with appropriate metric groups (Activity – Preparation: Select Metrics).

In order to get an overall validation assessment for each quality characteristic, the validation team should also investigate the appropriate weighting to be applied to:

- the quality subcharacteristics that are to be assessed (using Table 3 for initial guidance),
- the different quality attributes that they impact, and
- the different metrics within a metric group that will be used to quantify them.

4.7 Impact Matrix of Metrics on Quality (for Maintainability)

Analysis of existing metrics work related to the Maintainability quality characteristic identified the following sets of metrics (not all of which are applicable to OATA). The table below shows how each of these metrics may impact the quality of the Maintainability subcharacteristics and desirable values of the metrics in this context.

Note that the metric column contains original metrics from various sources. Some of these have been selected for assessment of the OATA Logical Architecture, but have in some cases been adapted to suit the specific nature of OATA.

Metric	Quality Subcharacteristics				Desirable Values	
	Analyzability	Changeability	Stability	Testability		
Morris Metrics Set	Methods per Class	Negative	Negative	Negative	Negative	Low
	Inheritance Dependencies	Negative	Negative	Negative	Negative	Low
	Degree of Coupling between Classes	Negative	Negative	Negative	Negative	Low
	Degree of Cohesion of Classes	-	Positive	-	-	High
	Class Library Effectiveness	-	-	-	-	High
	Factoring Effectiveness	Positive	Positive	Positive	Positive	High
	Degree of Reuse of Inheritance Methods	-	-	-	-	High
	Average Method Complexity	Negative	Negative	Negative	Negative	Low
	Application Granularity	Positive	Positive	Positive	Positive	High
Chidamber & Kemerer's Metrics Set	Weighted Methods per Class (WMC)	Negative	Negative	Negative	Negative	20 or 50 (1)
	Depth of Inheritance Tree (DIT)	Negative	Negative	Negative	Negative	<=5
	Number of Children (NOC)	Negative	Negative	Negative	Negative	Low (TBC)
	Coupling between classes (CBO)	Negative	Negative	Negative	Negative	Low (TBC)
	Response for a Class (RFC)	Negative	Negative	Negative	Negative	Low (TBC)
	Lack of Cohesion in Methods (LCOM)	Negative	Negative	Negative	Negative	Low (TBC)
MOOD Metrics Set	Method Hiding Factor (MHF)	Positive	Positive	Positive	Positive	>8% <25% (2)
	Attribute Hiding Factor (AHF)	Positive	Positive	Positive	Positive	100% ID >90%
	Method Inheritance Factor (MIF)	Negative	Negative	Negative	Negative	>20% <80%
	Attribute Inheritance Factor (AIF)	Negative	Negative	Negative	Negative	0% <50% (3)
	Polymorphism Factor (PF)	-	-	-	-	>5% <20% (4)
	Coupling Factor (CF)	Negative	Negative	Negative	Negative	>0% <12%

- (1) As maximum 10% of classes can have more than 24 methods (reference value).
- (2) Found in literature but not contrasted.
- (3) Contradictory ranges found, but low values are recommended.
- (4) Not justified.

Table 5: Impact of high values on the Quality Subcharacteristics

5 APPENDIX B METRICS

5.1 Introduction

This appendix gives a structured overview of the metrics that have been identified as applicable to the OATA logical architecture and a suggestion of how to group the applicable metrics.

A metric group contains the metrics that measure the same aspect of an architecture, for example inheritance or coupling. The different metric groups can be used to provide guidance for the selection of which metrics shall be used for the measurement of the OATA Architecture.

The description of each group contains the following information:

- **Description and calculation**
Contains a description of the metric group and an explanation of how to calculate the metric.
- **Quality and interpretation**
Contains a description of the quality characteristics that is associated with the metric and a description of how to interpret the metric measurement.
- **Desired value**
Contains example and recommendations regarding desired value of the metric.
- **OATA Applicability**
Identifies the levels in the OATA architecture at which the metrics in the group can be applied.
- **References/Sources**
Contains a reference to the original definition of the metric. For a summary, read about the metric in Appendix C “Metric Cards”. For details, see reference for the original metric.

Some of the metric groups are also divided into more specific subgroups, for example “Coupling metrics” has been divided into “Coupling between classes” and “Coupling between packages”.

Different common statistics, for example standard deviation, mean, average, maximum and minimum, can be calculated using the results from the basic metric measurements and are not described in this appendix.

5.2 Operations per class

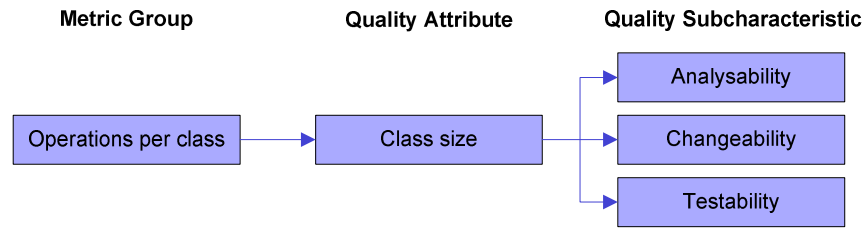
5.2.1 Description and calculation

Classes and operations are used to encapsulate data and responsibility in object-oriented development.

For each selected class, count the number of operations in the class (see 5.2.4 below for details).

5.2.2 Quality and interpretation

This metric contributes to the assessment of class size, which provides feedback on the Analysability, Changeability and Testability of the Architecture.



A large number of operations per class are usually an indication that the class is complex and application-specific, thus difficult to reuse. However, a large total number of operations per class may be desirable because subclasses tend to inherit a larger number of operations and inheritance increases code reuse.

The more operations a class has, the larger it will be and the more complicated the testing required.

5.2.3 Desired value

No exact value can be specified.

A similar metric, WMC (Weighted Methods per Class), is indicated by Chidamber & Kemerer to be low, (0 to 10). [Chidamber, Kemerer 94]

5.2.4 OATA Applicability

Operations per class are calculated for classes found in the Application part of the Specification Model in the OATA Logical View.

The table below describes operations to count in the OATA architecture.

Operations	OATA Architecture Operations	Count
Inherited	Yes	No
Private	Some	Yes
Friend	Probably not	Yes
Public	Yes	Yes
get/set operations	Yes	Decision has to be made before activity "Prepare Tools"

5.2.5 References/Sources

Metric name	Description	Source
MPC	Calculates an average of methods per class in a specified subset	Morris Metrics Set [Morris 89]
WMC	Weighted Methods per Class Calculates an average of weighted methods per class in a specified subset. (Weighting set to same value for all methods counts number of methods per class.)	[Chidamber, Kemerer 94]

5.3 Coupling metrics

5.3.1 Description

Coupling is a measure of the strength of association between different parts of the architecture. It is normally counted between classes and result calculated for classes, packages and (in the OATA case) clusters.

Three kinds of coupling between classes can be considered:

- Coupling between classes in the same package;
- Coupling between classes in different packages but same cluster; and
- Coupling between classes in different clusters.

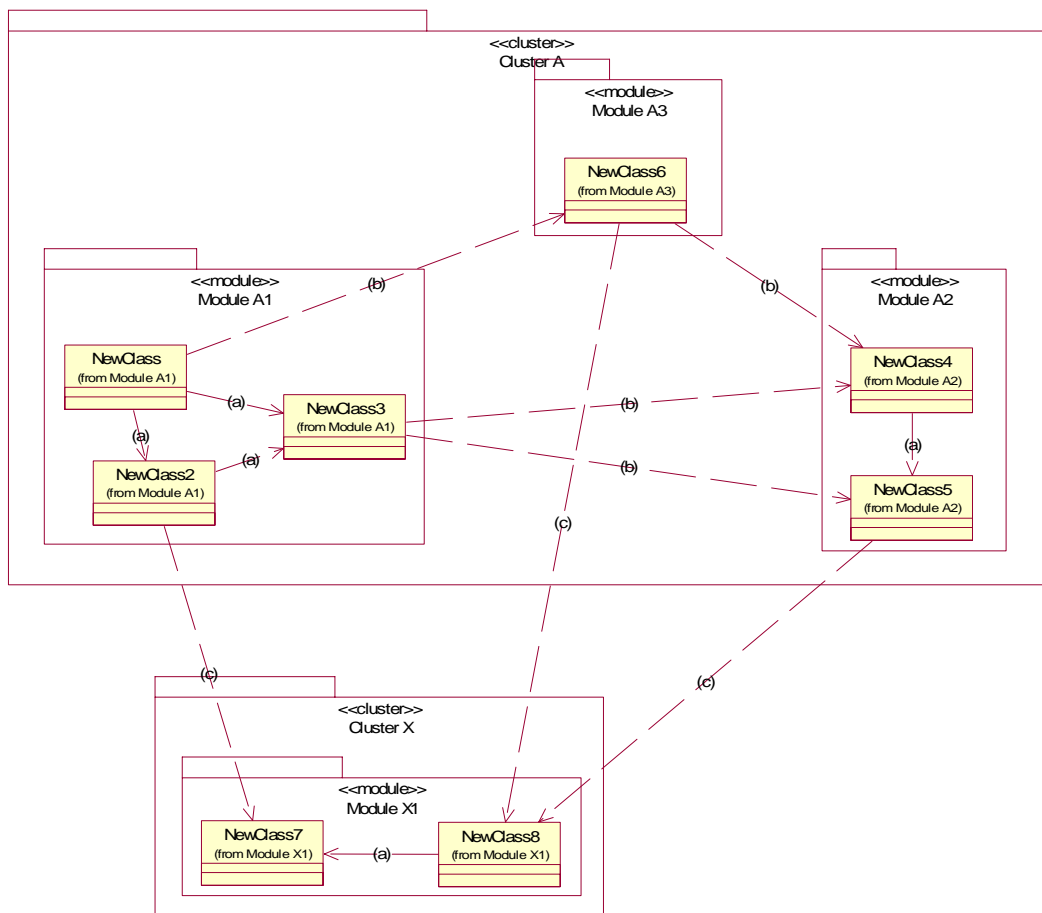
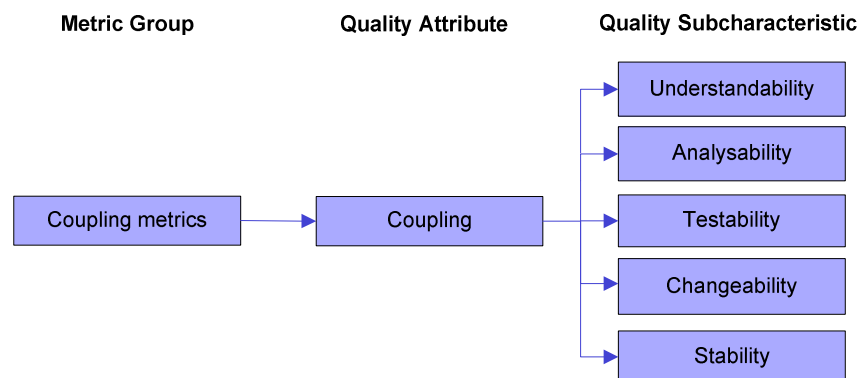


Figure 11: Aspects of Coupling between Classes

To get a full view of the coupling of a class, all three aspects of coupling need to be considered. Coupling between packages can be calculated using the last two aspects alone, while coupling between clusters considers just the third aspect. Note, however, that coupling between packages and clusters can also be counted directly by considering their dependencies.

5.3.2 Quality

This metric group contributes to the assessment of Coupling which provides feedback on the Understandability, Analysability, Changeability, Testability and Stability of the Architecture.



The lower the degree of coupling between classes, the more they will be suitable for reuse. The more independent a class is, the easier it is to reuse. Excessive coupling between classes is detrimental to modular design and prevents reuse.

Uncoupled classes are usually easier to augment than those with a high degree of coupling, due to the lower degree of interaction.

A measure of coupling is useful to determine how complex the testing of various parts of a design is likely to be. The higher the inter-class or package coupling, the more rigorous the testing needs to be (low testability).

Class interaction complexity associated with coupling can lead to increased error generation during development.

In order to improve modularity and promote encapsulation, inter-class couples should be kept to a minimum. The larger the number of couples, the higher the sensitivity to changes in other parts of the design, and therefore maintenance is more difficult (low maintainability).

5.3.3 Coupling between classes

5.3.3.1 Description and calculation

Coupling is a measure of the strength of association established by a connection from one class to another.

Classes are coupled in three ways:

- **Message passing**

When a message is passed between objects, the corresponding classes are coupled.

- **Shared data**

Classes are coupled when there is an association between them or a class is used to define the type of a class's attributes or parameters of its operations.

- **Inheritance**

Inheritance introduces a significant tight coupling between superclasses and their subclasses.

5.3.3.2 Desired value

The recommendation is a low value.

The exact range has to be decided in activity “Set thresholds” during validation preparation.

5.3.3.3 OATA Applicability

Coupling between classes are calculated between classes found in the Application part of the Model in the OATA Logical View.

5.3.3.4 References/Sources

Metric name	Description	Source
CBC	Coupling between Classes The average number of couplings per class in a subset	Morris Metrics Set [Morris 89]
CBO	Coupling between Objects Total number of couplings of a class	Chidamber & Kemerer object-oriented metrics [Chidamber, Kemerer 94]
CF	Coupling Factor: The number of couplings divided by the number of classes in the evaluated subset	MOOD Set [Abreu 95]

5.3.4 Coupling between packages

5.3.4.1 Description and calculation

Coupling between packages is not normally described in object-oriented metrics. Instead it arises for the calculation of statistics on the couplings between classes when the classes are in different packages.

5.3.4.2 Desired value

The recommendation is a low value.

The exact range has to be decided in activity “Set thresholds” during validation preparation.

5.3.4.3 OATA Applicability

Coupling between packages shall be counted for each Module in the Application part of the Specification Model in the OATA Logical View. It can also be calculated for the Clusters.

5.3.4.4 References/Sources

No standard metric exists.

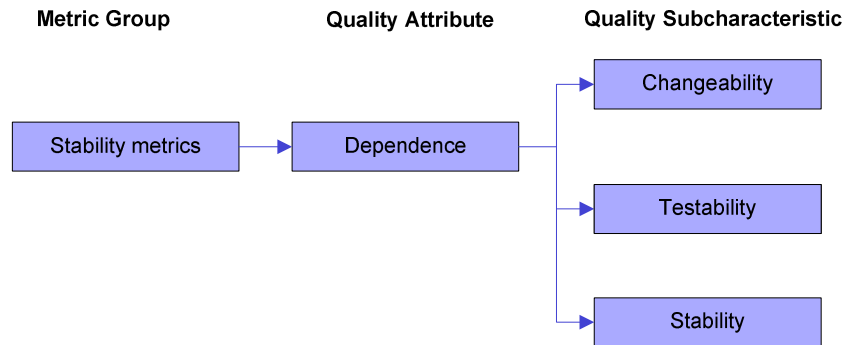
5.4 Stability metrics

5.4.1 Description

This metric group contains metrics that measure the level of stability in the architecture.

5.4.2 Quality

This metric group contributes to the assessment of Dependence, which provides feedback on the Changeability, Testability and Stability of the Architecture.



5.4.3 Efferent and afferent coupling

5.4.3.1 Description and calculation

Afferent coupling (incoming) counts the number of classes outside the analysed package that depend upon classes within the analysed package.

Efferent coupling (outgoing) counts the number of classes inside the analysed package that depend upon classes outside the analysed package.

5.4.3.2 Quality and interpretation

The number of packages that depend upon the analysed package is an indication of its level of responsibility. If the package is relatively abstract then a large number of incoming (afferent) dependencies are acceptable.

The larger the number of couples, the higher the sensitivity to changes in other parts of the design, and therefore the more difficult the maintenance with respect to changeability.

A large outgoing (efferent) coupling indicates that a class is unfocused and may also indicate that it is unstable, since it depends on the stability of all the types to which it is coupled.

5.4.3.3 Desired value

The recommendation is a low value.

The exact range has to be decided in activity "Set thresholds" during validation preparation.

5.4.3.4 OATA Applicability

Afferent and efferent coupling shall be counted for each Module in the Application part of the Specification Model in the OATA Logical View. It can also be calculated for the Clusters.

5.4.3.5 References/Sources

Metric name	Description	Source
Ca	Afferent Coupling: The number of incoming couplings to the analysed package	Chidamber & Kemerer's Metrics Suite (derivative) - 1994. Some attributes to Robert C. Martin. [Chidamber, Kemerer 94] [R Martin 94]
Ce	Efferent Coupling: The number of outgoing couplings from the analysed package	Chidamber & Kemerer's Metrics Suite (derivative) - 1994. Some attributes to Robert C. Martin. [Chidamber, Kemerer 94] [R.C. Martin 94]

5.4.4 Instability

5.4.4.1 Description and calculation

This metric was proposed by Robert C. Martin and measures the instability of packages, related to the effort to change one package without impacting other packages.

The instability of a package is its number of outgoing couplings divided by the sum of its incoming and outgoing couplings.

Instability = Outgoing couplings (Ce) / (Incoming couplings (Ca) + Outgoing couplings (Ce))
If $(Ca+Ce) = 0$ then *Instability=0*

See also metric description for Afferent Coupling, Ca and Efferent Coupling, Ce.

5.4.4.2 Quality and interpretation

The number of incoming and outgoing couplings is one indicator determining the changeability of a package, where:

- Packages that contain many outgoing but few incoming couplings are less stable because they may be impacted by changes in these packages.
- Packages containing many incoming couplings but few outgoing ones are more stable because they are more difficult to change without impacting other packages.

A stable package is a package that has no couplings with other packages, i.e. it is completely independent.

5.4.4.3 Desired value

Zero (0) indicates a maximum stable package and 1 indicates a maximum unstable package.

Designs of packages should intentionally be made as stable (range 0.0 to 0.3) or unstable (range 0.7 to 1.0) as possible.

The recommendation for OATA is a low value.

The exact range has to be decided in activity "Set thresholds" during validation preparation.

Robert C. Martin's Stable Dependencies Principle states that the dependencies between packages in a design should be in the direction of the stability of the packages; i.e. a package should depend only on packages that are more stable than itself.

5.4.4.4 OATA Applicability

Instability should be calculated for each Module in the Application part of the Specification Model in the OATA Logical View. It can also be calculated for the Clusters.

5.4.4.5 References/Sources

Metric name	Description	Source
I	Instability: Measures the instability/changeability of architecture	Robert C. Martin [R.C. Martin 94]

5.4.5 Cyclic dependency

5.4.5.1 Description and calculation

Packages have relationships that are dictated by the classes contained within them, and the relationships that these classes possess with classes in other packages. While time is usually spent designing the relationships between classes, it's unfortunate that little time is given to the relationships between packages.

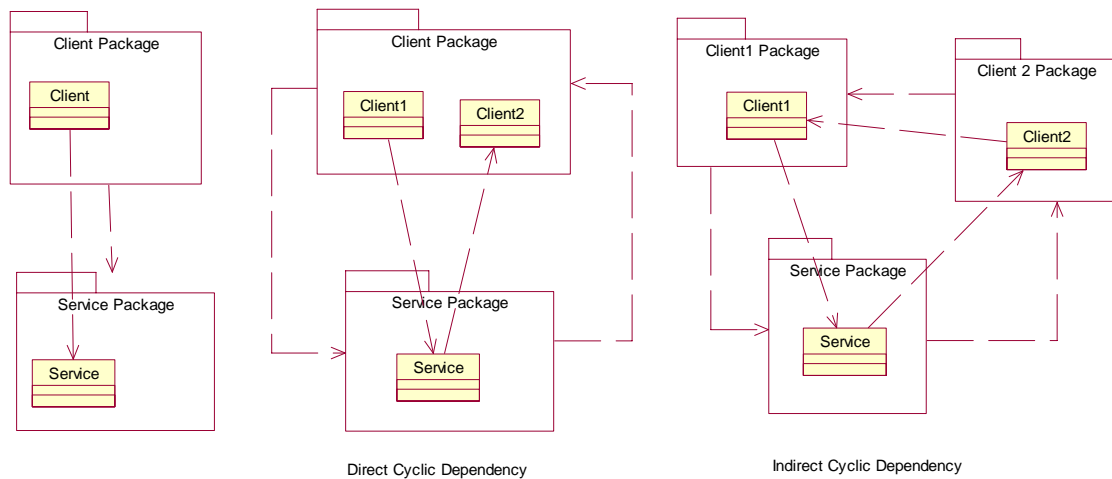


Figure 12: Package and Class Dependencies

The diagram illustrates no cyclic dependency, direct and indirect cyclic dependency. At left, we see a client package that contains a Client class. This Client class has a relationship to a Service class in the service package. In this scenario, we can easily reuse the service package and it's containing classes, because we know the classes in the service package have no other dependencies. At the centre and the right, however, we see a cycle in the dependency structure. In centre, the Client and Service packages have a bi-directional relationship, implying that the client package has a relation to the service package, and the service package has a relation to the client package. In the right, the Client1, Client2 and Service packages have circular relationship. In these scenarios, neither the client nor the service packages can be used without the other.

The metric is calculated by counting the number of cyclic dependencies of a package (direct and indirect).

5.4.5.2 Quality and interpretation

Cyclic dependencies are difficult to maintain since cyclically dependent packages are harder to comprehend individually. They cannot be packaged, versioned, and distributed independently. Consequently, this metric is useful to catch cyclic package dependencies before they make it into a software baseline.

5.4.5.3 Desired value

The exact range has to be decided in activity “Set thresholds” during validation preparation.

5.4.5.4 OATA Applicability

The metric is counted in a dependency graph generated from packages in the Application part of the Specification Model in the OATA Logical View.

5.4.5.5 References/Sources

Metric name	Description	Source
CYC	Cyclic Dependency The number of mutual package couplings for one package	Robert C. Martin [R.C. Martin 94]
DCYC	Direct Cyclic Dependency The number of mutual package couplings between two packages	Robert C. Martin [R.C. Martin 94]

5.5 Inheritance

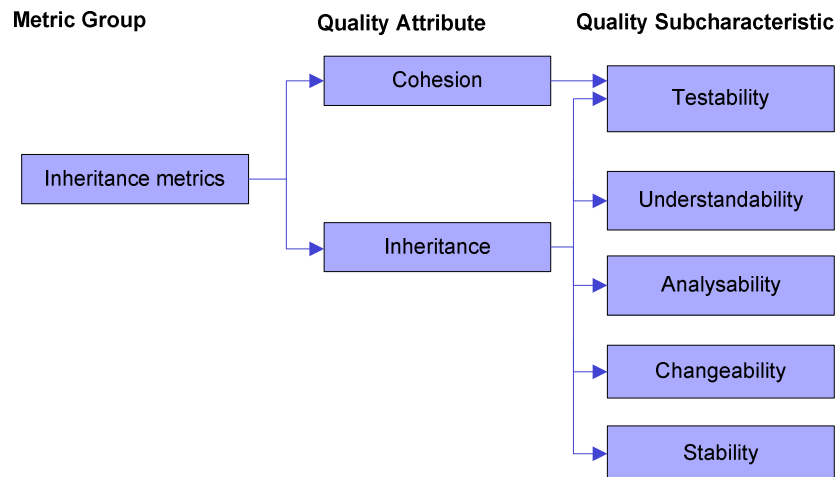
5.5.1 Description

One design abstraction in object-oriented design is the use of inheritance. Inheritance is a relationship between classes that enables designers to reuse previously defined elements, including attributes, operations and associations.

Inheritance metrics primarily evaluate the amount of reuse in the architecture.

5.5.2 Quality

This metric group contributes to the assessment of Cohesion, and Inheritance, which provide feedback on the Understandability, Analysability, Changeability, Stability and Testability of the Architecture.



Inheritance decreases complexity by reducing the number of operations, but this abstraction of objects can make maintenance and design difficult.

5.5.3 Depth of inheritance

5.5.3.1 Description and calculation

Depth of inheritance measures the depth of a class within the inheritance hierarchy. It is the maximum length from the class node to the root of the tree, measured by the number of ancestor classes.

5.5.3.2 Quality and interpretation

The deeper a class is in the hierarchy, the greater the number of methods it is likely to inherit, making it more complex to predict its behaviour.

Deeper trees constitute greater design complexity.

A correlation between inheritance depth and the number of faults found in the final implementation has been discovered both in research and real-life implementations.

5.5.3.3 Desired value

The exact range has to be decided in activity “Set thresholds” during validation preparation.

[RefactorIT] suggests that the desired value of depth of inheritance should not be more than 5.

5.5.3.4 OATA Applicability

Depth of inheritance is calculated for classes found in the Application part of the Specification Model in the OATA Logical View.

5.5.3.5 References/Sources

Metric name	Description	Source
DIT	Depth of inheritance Tree: The depth of a class within the inheritance hierarchy is the maximum length from the class node to the root of the tree, measured by the number of ancestor classes.	Chidamber & Kemerer object-oriented metrics [Chidamber,Kemerer 94]

5.5.4 Width of inheritance

5.5.4.1 Description and calculation

Width of inheritance is the measure of inheritance via the number of direct subclasses of a class. It is also known as “Number of Children” or “Number of Direct Subclasses of a Class”.

To calculate, count the direct subclasses of a class.

5.5.4.2 Quality and interpretation

The number of children approximately indicates how an application reuses code and is an indicator of the potential influence a class can have on the testability and design of the system. It is assumed that the more children a class has, the more responsibility there is on the maintainer of the class not to break the children's behaviour. As a result, it is harder to modify the class and it requires more testing.

Hence:

- The greater the number of children, the greater the reuse of code, since inheritance is a form of reuse.
- The greater the number of children, the greater the likelihood of improper abstraction of the parent class. If a class has a large number of children, it may be a case of misuse of sub-classing.
- The number of children gives an idea of the potential influence a class has on the design. If a class has a large number of children, it may require more testing of the methods in that class, thus increase the testing time.

5.5.4.3 Desired value

The exact range has to be decided in activity “Set thresholds” during validation preparation.

However, the initial recommendation is that width of inheritance should be between 0 and 10. If width of inheritance exceeds 10 children for a class, this may indicate a misuse of sub-classing because more reuse implies poor design and increased testing requirements.

5.5.4.4 OATA Applicability

Width of inheritance is calculated for classes found in the Application part of the Specification Model in the OATA Logical View.

5.5.4.5 References/Sources

Metric name	Description	Source
NOC	Number of children: Count number of direct child classes derived from a class via the Inherits statement.	Chidamber & Kemerer object-oriented metrics [Chidamber, Kemerer 94]

5.6 Encapsulation

5.6.1 Description and calculation

Encapsulation refers to the concept of making an object a "black box". An object should expose only the absolutely necessary information needed to interface with it. Encapsulation also means that an object includes everything it needs: both the data and the operations on it (methods). In a good modular design, each of the modules encapsulates implementation details that are not visible to the user of the module.

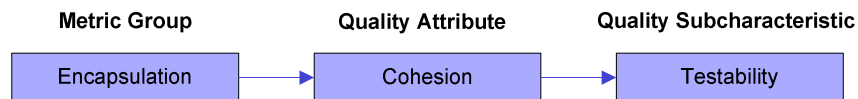
To measure this correctly for a class, it is necessary to have both public and private declarations of operations and attributes of classes. However, by measuring the number of classes used outside a package, it's possible to have an indication on encapsulation of a package.

Count the number of classes from outside the package that the package has an efferent coupling to, and divide with the number of classes in the package.

$$EP = \text{Number of used classes outside this package} / \text{Total number of classes in this package}$$

5.6.2 Quality and interpretation

This metric contributes to the assessment of Cohesion, which provides feedback on the Testability of the Architecture.



5.6.3 Desired value

The recommendation is a low value. A low value indicates "good" encapsulation of data/algorithms within a package.

The exact range has to be decided in activity "Set thresholds" during validation preparation.

5.6.4 OATA Applicability

EP should be calculated for each Module in the Application part of the Specification Model in the OATA Logical View. It can also be calculated for the Clusters.

5.6.5 References/Sources

Metric name	Description	Source
EP	Encapsulation Principle: Number of used classes outside this package divided by total number of classes in this package	

5.7 Response for a class

5.7.1 Description and calculation

Response for a class is the number of distinct methods and constructors invoked by a class. This is the cardinality of the set of *all* operations and constructors that can be invoked (i.e. operations in the class, inheritance hierarchy, and operations located in other objects) as a result of any message that could be sent to an object of the class.

The response set for the class can be expressed as:

$$RS = \{ M \} \cup \text{all } i \{ Ri \}$$

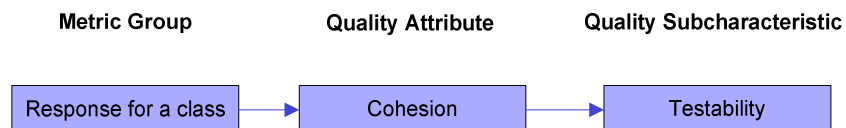
where $\{ Ri \}$ = set of methods called by method i and $\{ M \}$ = set of all methods in the class

For each remote operation, (each operation that can receive a message from another class) count the number of distinct operations that can be invoked through the entire call tree.

A given operation is counted only once.

5.7.2 Quality and interpretation

This metric contributes to the assessment of Cohesion, which provides feedback on the Testability of the Architecture.



If a large number of methods can be invoked in response to a message, the testing and debugging of the class becomes more complicated since it requires a greater level of understanding on the part of the tester. If the “response for a class” is large, it can be difficult to comprehend the behaviour and detect problems.

5.7.2.1 Desired value

The recommendation is a low value.

The exact range has to be decided in activity “Set thresholds” during validation preparation.

5.7.3 OATA Applicability

The metric shall be counted for classes found in the Application part of the Specification Model in the OATA Logical View considering sequence diagrams in the Application part of the Specification Model and (where necessary) the Use Case Realisation of the OATA Logical View.

5.7.4 References/Sources

Metric name	Description	Source
RFC	Response for class: The set of all operations and constructors that can be invoked as a result of a message sent to an object of the class.	Chidamber & Kemerer object-oriented metrics [Chidamber, Kemerer 94]

5.8 Readability

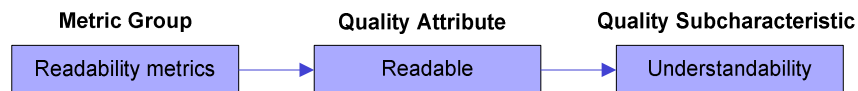
5.8.1 Description and calculation

There are 3 metrics in this group that help assess the readability of architecture, addressing:

- the length of the names of the architecture elements;
- the uniqueness of the names (count number of times each name is used); and
- the density of documentation (e.g. number of words)

5.8.2 Quality and interpretation

This metric contributes to the assessment of Readability, which provides feedback on the Understandability of the Architecture.



The longer the names of actors, use-cases, modules/packages, functions, attributes etc, the more descriptive they probably are, therefore increasing understandability.

It is acceptable within UML to use the same name at many locations. However, understandability is increased if the name always refers to the same logical thing. For example, an attribute Username could be expected always to contain the same type of username with the same data type (string). If Username can mean one thing in one class and another thing somewhere else, the likelihood of confusion increases.

The more description contained in the model, generally the easier the architecture is to read and understand - as long as the information is relevant.

5.8.3 Desired value

The exact range has to be decided in activity “Set thresholds” during validation preparation.

5.8.4 OATA Applicability

All elements in the parts of the model that is included in the validation cycle for assessment.

5.8.5 References/Sources

Metric name	Description	Source
LEN	Length of names: The average length of element names in a subset	
UNIQ	Name Uniqueness Ratio: The average number of unique names in a subset	
DC	Density of Comments: The average length of descriptions in a subset	

6 APPENDIX C METRIC CARDS

6.1 Methods per object class

Metric Name	Methods per object class	MPC
Original metric information		
Description	<p>Mr Morris describes three different metrics related to methods per object class.</p> <p>“Average number of methods per class” is the average number of methods per class in a subset and measures both size and complexity.</p> <p>“Max number of methods per class” is the maximum number of methods per class in the subset.</p> <p>“Min number of methods per class” is the minimum number of methods per class in the subset.</p>	
Method of Calculation	<p>Average number of methods per class = Total number of methods / Total number of classes</p> <p>Max number of methods per class = Max number of methods of any class in the subset</p> <p>Min number of methods per class = Minimum number of methods of any class in the subset</p>	
Source	[Morris 89]	

6.2 Weighted methods per class

Metric Name	Weighted methods per class	WMC
Original metric information		
Description	<p>The number of methods and the complexity of methods involved are indicators of how much time and effort is required to develop and maintain the object.</p> <p>The larger the number of methods in an object, the greater the potential impact on the children, since, children will inherit all the methods in the object. A large number of methods can result in a too application specific object, thus limiting the possibility of reuse.</p> <p>Since WMC can be described as an extension of the Cyclomatic Complexity⁴ (CC) [McCabe 96] metric (if CC is used to calculate WMC) that applies to objects, its recommended threshold value can be compared with the upper limit of the CC metric. Take the calculated WMC value and divide it with the number of methods, this value can then be compared with the upper limit of CC. One disadvantage of using CC in order to measure an objects complexity is that the WMC value cannot be collected in early design stages, e.g. when the methods in a class have been defined but not implemented.</p> <p>To be able to measure WMC as early as possible one could use the number of methods in a class as a complexity value, but then the WMC metric is no longer a complexity measure but a size measure, also known as the methods per class metric (MPC).</p>	
Method of Calculation	<p>Consider a class C1, with methods M1, M2, ...Mn. Let c1, c2, ...cn be the complexity⁵ of each method. Then:</p> $WMC = \sum_i^n C_i, \text{ where } n \text{ is the number of methods in the class.}$ <p>If all complexities are considered to be unity, $WMC = n$, the number of methods.</p>	
Source	[Chidamber, Kemerer 94]	

4 Cyclomatic Complexity [McCabe 96] provides a measure of the number of loops and complexity of the code implementing a method.

5 Complexity is deliberately not defined more specifically here in order to allow for the most general application of this metric. It can be argued that developers approach the task of writing a method as they would a traditional program, and therefore some traditional complexity metric may be appropriate. This is left as an implementation decision, as the general applicability of any existing complexity metric has not been generally agreed upon. Any complexity metric used in this manner should have the properties of an interval scale to allow for summation. [Chidamber, Kemerer 94] One possible complexity metric is Cyclomatic Complexity, CC [McCabe 96].

6.3 Depth of inheritance, DIT

Metric Name	Depth of inheritance	DIT
General metric information		
Description	<p>The purpose is to measure the inheritance structure complexity.</p> <p>Depth of inheritance of the class is the DIT metric for the class. In cases involving multiple inheritances, the DIT will be the maximum length from the node to the root of the tree.</p> <ul style="list-style-type: none"> • The deeper a class is in the hierarchy, the greater the number of methods it is likely to inherit, making it more complex to predict its behaviour. • Deeper trees constitute greater design complexity, since more methods and classes are involved. • The deeper a particular class is in the hierarchy, the greater the potential reuse of inherited methods. 	
Method of Calculation	<p>The depth of a class within the inheritance hierarchy is the maximum length from the class node to the root of the tree, measured by the number of ancestor classes.</p> <p>DIT = maximum inheritance path from the class to the root class</p>	
Source	[Chidamber, Kemerer 94]	

6.4 Number of children, NOC

Metric Name	Number of children	NOC
General metric information		
Description	<p>The number of children (NOC) is the number of immediate sub-classes subordinated to a class in the class hierarchy.</p> <ul style="list-style-type: none"> • The greater the number of children, the greater the reuse, since inheritance is a form of reuse. • The greater the number of children, the greater the likelihood of improper abstraction of the parent class. If a class has a large number of children, it may be a case of misuse of sub-classing. • The number of children gives an idea of the potential influence a class has on the design. If a class has a large number of children, it may require more testing of the methods in that class. 	
Method of Calculation	<p>Count number of direct child classes derived from a class via the Inherits statement.</p> <p>NOC = number of direct sub-classes of a class</p>	
Source	[Chidamber, Kemerer 94]	

6.5 Coupling between classes, CBC

Metric Name	Coupling between classes	CBC
General metric information		
Description	<p>CBC is a measure of the control interconnections between objects. Two possible object coupling metrics might be <i>the average number of “uses” dependencies per object</i> and <i>the maximum number of “uses” dependencies per object</i>.</p> <p>The greater the number of <i>use</i> interactions between objects, the greater the amount of coupling between those objects.</p> <p>Ideally, to maximize objects’ encapsulation, coupling between objects should be minimized.</p> <ul style="list-style-type: none"> • A higher degree of coupling between objects is likely to complicate application maintenance because the objects’ interconnections and interactions are more complex. • The higher the degree of object independence (i.e. the more “uncoupled” objects are from each other) the more likely it is that the objects will be suitable for reuse within the same applications and within other applications. • Uncoupled objects should be easier to augment than those with high degree of “uses” dependencies, due to the lower degree of interaction. • Testability is likely to degrade with a more highly coupled system of objects. Coupling is likely to lead to testing complexity. • A highly coupled “uses” network is more complex and, therefore, likely to be less comprehensible. • Object interaction complexity associated with coupling can lead to increased error generation during development. • Uncoupled objects should be easier to replace with customized objects than those with a high degree of “uses” dependencies, due to the lower degree of interaction. <p>For more details see [Morris 89].</p>	
Method of Calculation	<p>Count the number of relationships within the selected area. Then calculate the number of classes and calculate the division.</p> <p>The types of relationships that will be considered (separately) are:</p> <ul style="list-style-type: none"> • Association • Dependencies (recommended) <p>$CBC = \text{Total number of couplings} / \text{Total number of classes}$</p>	
Source	[Morris 89]	

6.6 Coupling between Object Classes, CBO

Metric Name	Coupling between Object	CBO
General metric information		
Description	<p>CBO for a class is a count of the number of other classes to which it is coupled.</p> <p>CBO relates to the notion that an object is coupled to another object if one of them acts on the other, i.e., methods of one use methods or instance variables of another. Since objects of the same class have the same properties, two classes are coupled when methods declared in one class use methods or instance variables defined by the other class.</p> <ul style="list-style-type: none"> Excessive coupling between object classes is detrimental to modular design and prevents reuse. The more independent a class is, the easier it is to reuse it in another application. In order to improve modularity and promote encapsulation, inter-object class couples should be kept to a minimum. The larger the number of couples, the higher the sensitivity to changes in other parts of the design, and therefore maintenance is more difficult. A measure of coupling is useful to determine how complex the testing of various parts of a design is likely to be. The higher the inter-object class coupling, the more rigorous the testing needs to be. 	
Method of Calculation	CBO = Number of classes to which the class is coupled	
Source	[Chidamber, Kemerer 94]	

6.7 Coupling Factor, CF, COF

Metric Name	Coupling Factor	CF, COF
General metric information		
Description	<p>The Coupling Factor (CF) metric was proposed as a measure of coupling between classes excluding coupling due to inheritance.</p> <p>[Harrison, Counsell, Nithi 98]</p>	
Method of Calculation	<p>CF is calculated by considering all possible pair wise sets of classes, and asking whether the classes in the pair are related, either by message passing or by semantic association links (references by one class to an attribute or method of another class).</p> <p>$CF = \text{Actual couplings} / \text{Number of classes in the evaluated subset}$</p> <p>[Harrison, Counsell, Nithi 98]</p>	
Source	<p>[Fernando Brito e Abreu, 95]</p> <p>[Harrison, Counsell, Nithi 98]</p>	

6.8 Afferent Coupling, Ca

Metric Name	Afferent Coupling	Ca
General metric information		
Description	<p>Afferent Coupling (also known as Incoming Dependencies and Number of Types outside a Package that Depend on Types of the Package) indicates the number of classes and interfaces from other packages that depend on classes in the analysed package.</p> <p>The number of packages that depend upon the analysed package is an indication of the analysed package's level of responsibility. In order to improve modularity and promote encapsulation, inter-object class couples should be kept to a minimum. If the package is relatively abstract then a large number of incoming dependencies is acceptable but the larger the number of couples, the higher the sensitivity to changes in other parts of the design, and therefore maintenance is difficult. Excessive coupling between concrete object classes is detrimental to modular design and prevents reuse.</p> <p>Summary by RefactorIT [RefactorIT]</p>	
Method of Calculation	<p>Ca = cardinality ({classes from other packages that depend on classes in the analysed package})</p> <p>Preferred values range between [0,500]</p> <p>[RefactorIT]</p>	
Source	<p>[RefactorIT]</p> <p>[Chidamber, Kemerer 94]</p> <p>[R.C. Martin, 94]</p>	

6.9 Efferent Coupling, Ce

Metric Name	Efferent Coupling	Ce
General metric information		
Description	<p>Ce (also known as Outgoing Dependencies or the Number of Types outside a Package those Types of the Package Depend on) indicates the number of classes and interfaces in other packages that classes and interfaces in the analysed package depend upon. In short, this measure includes all the types referred to anywhere within the source of the measured package.</p> <p>RefactorIT recommends an upper limit of 20 since excessive coupling is detrimental to modular design since classes are not independent. A large efferent coupling indicates that a package is unfocussed and may also indicate that it is unstable, since it depends on the stability of all the types to which it is coupled. This prevents reuse since a high coupling possibly indicates a package is poorly designed and difficult to understand/maintain.</p> <p>Summary by RefactorIT [RefactorIT]</p>	
Method of Calculation	<p>Ce = cardinality({classes from other packages that the current package depends upon})</p> <p>Preferred values range between [0,20]</p> <p>[RefactorIT]</p>	
Source	<p>[RefactorIT]</p> <p>[Chidamber, Kemerer 94]</p> <p>[R.C. Martin ,94]</p>	

6.10 Instability, I

Metric Name	Instability	I
General metric information		
Description	<p>This metric was proposed by Robert C. Martin and measures the instability of packages, where stability is measured by calculating the effort to change a package without impacting other packages within the application. The number of incoming and outgoing dependencies is one indicator determining the stability and instability of a package, where:</p> <ul style="list-style-type: none"> • Packages that contain multiple outgoing but few incoming dependencies are less stable because of the consequences of changes in these packages. • Packages containing more incoming dependencies are more stable because they are more difficult to change. • 0.0 indicates a maximally stable package and 1.0 indicates a maximally unstable package. • Designs of packages should intentionally be made as stable [0.0,0.3] or unstable [0.7,1.0] as possible. <p>Robert C. Martin's Stable Dependencies Principle states that the dependencies between packages in a design should be in the direction of the stability of the packages; i.e. a package should depend only on packages that are more stable than itself.</p> <p>Summary by RefactorIT [RefactorIT]</p>	
Method of Calculation	$I = Ce / (Ca + Ce)$ <p>Preferred values for I range between [0.0, 0.3] or [0.7, 1.0].</p> <p>[RefactorIT]</p>	
Source	[R.C. Martin, 94]	

6.11 Cyclic Dependencies, CYC

Metric Name	Cyclic Dependencies	CYC
General metric information		
Description	<p>CYC is an estimate of how many cycles a package is involved with by determining the number of mutual coupling dependencies between packages. That is, the number of other packages the package depends upon and which in turn depends on that package.</p> <p>Cyclic dependencies are difficult to maintain and indicate potential code to apply refactoring changes, since cyclically dependent packages are not only harder to comprehend/compile individually, but they cannot be packaged, versioned, and distributed independently. Thus, they violate the idea that a Java package is the unit of release. Consequently, this metric is useful to catch cyclic package dependencies before they make it into a software baseline.</p> <p>Summary by [RefactorIT]</p>	
Method of Calculation	The metric is calculated by counting the number of cyclic dependencies of a package (direct and indirect). ⁶	
Source	Robert C Martin [R.C. Martin, 94]	

⁶ The calculation of this metric is not described in detail by R C Martin.

6.12 Response for a class, RFC

Metric Name	Response for a class	RFC
General metric information		
Description	<p>The response set of a class is a set of methods that can potentially be executed in response to a message received by an object of that class. The cardinality of this set is a measure of the attributes of objects in the class. Since it specifically includes methods called from outside the class, it is also a measure of the potential communication between the class and other classes.</p> $RFC = RS \text{ where } RS \text{ is the response set for the class.}$ <p>The response set for the class can be expressed as:</p> $RS = \{ M \} \cup \text{all } i \{ Ri \}$ <p>where $\{ Ri \}$ = set of methods called by method i and $\{ M \}$ = set of all methods in the class</p> <ul style="list-style-type: none"> • If a large number of methods can be invoked in response to a message, the testing and debugging of the class becomes more complicated since it requires a greater level of understanding required on the part of the tester. • The larger the number of methods that can be invoked from a class, the greater the complexity of the class. • A worst-case value for possible responses will assist in appropriate allocation of testing time. 	
Method of Calculation	<p>RFC = number of methods that are directly invoked in response to a message RFC' = number of methods that are invoked recursively in response to a message (extension of [Chidamber, Kemerer 94])</p>	
Source	[Chidamber, Kemerer 94]	

6.13 Encapsulation Principle, EP

Metric Name	Encapsulation Principle	EP
General metric information		
Description	<p>The Encapsulation Principle can be paraphrased by 'A substantial part of a package should not be used outside of the package.' The EP metric is the ratio of objects that are used outside of the package, thus a low value indicates "good" encapsulation of data / algorithms within a package.</p>	
Method of Calculation	<p>Count the number of classes outside the package that the package uses and divide with the total number of classes in the package.</p> $EP = \text{Number of used classes outside this package} / \text{Total number of classes in this package}$	
Source		

6.14 Length of names, LEN

Metric Name	Length of names	LEN
General metric information		
Description	The purpose is to measure the average length of names.	
Method of Calculation	Count total length of element names in a subset and divide with number of elements. LEN = Average length of names	
Source		

6.15 Name Uniqueness Ratio, UNIQ

Metric Name	Name Uniqueness Ratio	UNIQ
General metric information		
Description	The purpose is to measure the uniqueness of all names.	
Method of Calculation	Calculate number of unique names and divide with total number of names. UNIQ = Number of unique names / Total number of names	
Source		

6.16 Comment density, DC

Metric Name	Comment density	DC
General metric information		
Description	The purpose is to measure the density of comments or descriptions.	
Method of Calculation	Count the length of descriptions and divide with number of elements. DC = Length of description / Number of elements	
Source		

7 APPENDIX D: SCORECARD EXAMPLE

7.1 Introduction

A scorecard is a layered and categorised instrument that helps to identify important statistics, whilst ensuring a proper balance. A scorecard is typically used to organise multiple statistics into an intelligible framework. The first columns are filled in as the corresponding activities are performed. Once the metrics have been calculated, they are entered into the final column (Assessment result). The other Assessment result columns can then be derived, from right to left, aggregating the results and taking into account weightings as required - eventually leading to the overall result per quality characteristic.

7.2 Scorecard

Scorecard											
Validation Aim	Validation Objective	Quality characteristics	Assessment result (Quality char.)	Quality sub-characteristics	Weight	Assessment result (Quality subchar.)	Quality Attribute	Assessment result (Quality Attributes)	Metric	Assessment result (Metrics)	

8 APPENDIX E: AVAILABLE TOOLS LIST

8.1 Introduction

This appendix presents some information needed to identify and select a metrics tool. The information includes a list of tool features recommended for an OATA metrics tool, a market survey of available metrics tools (sections 8.3 and 8.4), and a recommendation for a specific metrics tool (section 8.5).

8.2 OATA Suitable Tool Features

Even though it is recommended to create a detailed check-list with all the features considered relevant for an OATA metrics tool, this appendix presents a minimum tool requirements set.

The main proposed requirements for the OATA metrics tool are summarized as follows:

- UML-based automatic OO metrics counter.
- Report generation capabilities.
- Input format: As compatible as possible with the OATA toolset.
- Output format: Standard output (tool's own HMI), XML, CSV, MS-Excel, MS-Word, etc.
- Interface with other CASE tools (ad-hoc synchronization mechanisms, static communication through XML, CSV, interface with the Microsoft Office toolset, etc.).
- Capabilities for statistical analysis (regression, cluster, Kiviat graphs, six-sigma analysis, etc.).
- Multi-level metrics organisation (application/system, component/subsystem, class/unit).
- Support for the MS-Windows environment.
- Customization capabilities: adaptable formulas for metrics, Quality Criteria, Factors and Attributes.

In general, an OO metrics tool should show a structure like the one in the diagram below:

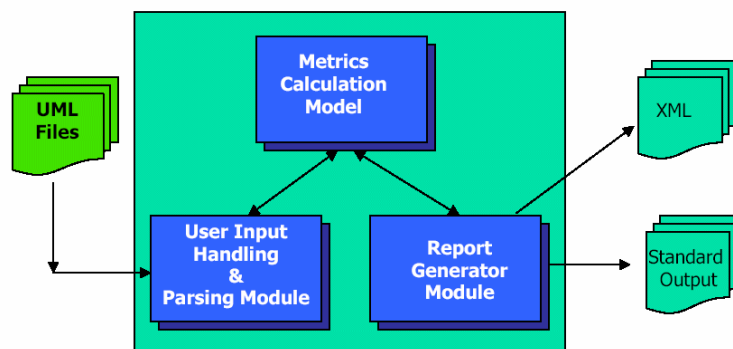


Figure 13: Tool Metamodel

One of the essential requirements is that the tool be able to capture and handle UML (including models and diagrams). This also includes the capability to parse the files to identify relevant elements (classes, attributes, methods, associations, etc.)

It is also required that the tool be able to calculate the values for the metrics identified as applicable to OATA. And finally, the tool should be able to generate reports and outputs in order to easily interpret the results or even to exploit the raw data provided through further treatment or analysis (i.e. statistical or time-evolution analysis).

8.3 Architecture and Code Metrics Tools

Name TAU/Logiscope
Vendor Telelogic
Website <http://www.telelogic.com/products/tau/logiscope/>

Name McCabe QA
Vendor McCabe & Associates
Website <http://www.mccabe.com/>

Name SDMetrics
Vendor SDMetrics
Website <http://www.sdmetrics.com/>

Name ObjectQuest Advisor
Vendor Hallogram
Website <http://www.hallogram.com/objectquest/>

Name MAISA
Vendor Non-commercial tool
Website <http://www.cs.helsinki.fi/group/mais/>

Name OSMAT
Vendor Non-commercial tool
Website <http://faculty.cs.tamu.edu/hohin/>

8.4 Code Metrics Tools

Name Insure++
Vendor CERN Software Development Tools Department
Website <http://sdt.cern.ch/Insure/>

Name CodeWizard
Vendor ParaSoft
Website <http://www.parasoft.com/>

Name Imagix 4D
Vendor Imagix Power Software
Website <http://www.imagix.com>

Name Jmetric
Vendor Jmetric Inc.
Website <http://www.it.swin.edu.au/projects/jmetric/products/jmetric/>

Name QSM-SLIM-Metrics
Vendor QSM Inc.
Website http://www.qsm.com/slim_metrics.html

Name Metamata Metrics
Vendor MetaMata
Website <http://www.metamata.com>

Name Scientific Toolworks
Vendor Scientific Toolworks, Inc.
Website <http://www.scitools.com/>

8.5 OATA Tool Recommendations

According to the requirements specified previously, an overview of tool market in the field has been performed. The first conclusion of this preliminary study is that no existing tool satisfies the whole set of requirements. Nevertheless, the SDMetrics toolset is the one that comes closest to it.

The most common feature found in such these tools is the support of metrics at code source level. Very few tools were found supporting architectural level metrics.

An additional problem is that, even for those tools supporting architectural level metrics (i.e. Telelogic's TAU/Logiscope), the input data are very often the source code files (generated from UML models).

The most suitable tools found can be classified into two categories:

- Well-known commercial tools, with international support, but featuring poor support to architectural metrics (i.e. Telelogic's TAU/Logiscope).
- Academic or research tools, with a reasonable architectural metrics coverage, but little or no commercial/technical support (i.e. OSMAT or MAISA).

According to these conclusions and prior to a more detailed tool market analysis, we recommend:

1. to create an OATA specific metrics tool, or
2. to adapt an existing commercial tool, through its extensibility mechanism, to fulfil the OATA requirements.

Some CASE tools, such as IBM/Rational Rose or Telelogic TAU, provide API's and script languages to facilitate the creation of add-ons to the default tool.

NB: No economical or maintenance issues have been considered in this evaluation. No systematic trade-off analysis has been performed, only a quick overview of the existing commercial and academic/research tools in the field of OO metrics.

8.6 Detailed description of tools

8.6.1 Logiscope

Vendor: Telelogic

Web-site: <http://www.telelogic.com/products/tau/logiscope/>

Logiscope supports the following programming languages: Ada, C, C++ and Java. It addresses two kind of problems:

1. Static Analysis: for metrics of the source code,
2. Dynamic Analysis: for test coverage on instrumented program.

The analysis can be performed both from the command line and from graphical user interfaces.

Logiscope organises the available metrics according to the following information structure:

- Programming Language.
 - Factor.
 - List of factors (ISO 9126 Quality Attributes).
 - Criteria.
 - List of criteria (ISO 9126 Quality Factors).
 - Metric.
 - List of metrics.

Each programming language has a specific set of the available factors, criteria and metrics.

The implementation of numeric calculations for high-level quality characteristics is as follows:

- Factor = Linear combination of (Subset of Criteria).
- Criterion = Linear combination of (Subset of Metrics).

The usual linear combination formula is just a simple addition of lower level quality characteristics.

The tool help does not provide any rationale about the selection of the subsets of metrics associated to the criteria. They are treated just as definitions in this documentation.

8.6.2 McCabe QA

Vendor: McCabe & Associates

Website: <http://www.mccabe.com/>

The vendor provides a toolset to cover different aspects of the quality and complexity of SW systems.

- **Objective Quality Assessment**

McCabe QA mitigates software risk by calculating insightful metrics and thereby identifying complex and error-prone code.

These metrics include:

- McCabe Cyclomatic Complexity.
- McCabe Essential Complexity.
- Module Design Complexity.

- Integration Complexity.
- Lines of Code.
- Halstead.
- **Visualized Quality Improvement**

McCabe QA's Metric Trend reports track a system's metric values over time to document the progress made in improving the overall stability and quality of the project.

8.6.3 Insure

Vendor: CERN Software Development Tools Department

Website: <http://sdt.cern.ch/Insure/>

Insure is a Source level run-time debugger for C and C++. It produce more robust, better optimised, higher quality software.

Insure++ includes three distinct tools that work together to improve the development process. All the results are graphics.

- Insure automatically detects large classes of programming and run-time errors. It quickly pinpoints algorithmic anomalies, bugs, and deficiencies.
- Inuse visualises in real-time the memory manipulation of the program, helping developers spot bugs and inefficiencies in memory handling.
- TCA also performs coverage analysis of programs, providing necessary feedback to programmers about which parts of the code were actually tested.

8.6.4 CodeWizard

Vendor: ParaSoft

Website: <http://www.parasoft.com/>

CodeWizard can be customized to a particular project or user; the analyst can suppress coding standards that do not pertain to a specific exceptional case. The analyst can create rules using CodeWizard's RuleWizard feature. CodeWizard implements RuleWizard coding standards along with existing ParaSoft coding standards.

Benefits of CodeWizard:

- Reduces bugs by 25%.
- Enforces over 70 coding standards automatically.
- Provides a full, detailed explanation for each error found.

8.6.5 Imagix 4D

Vendor: Imagix Power Software.

Website: <http://www.imagix.com>

Imagix 4D generates a series of quality metrics about the source code to help the analyst to identify potential problems in the development and maintenance of the software. You can use source checks to find exceptions to generally agreed upon design and coding standards. You can compare the software metrics to specific norms for the organisation to insure that the software meets the development criteria.

Initially, the analyst can use the software metrics to identify potential problem areas in the code, such as where the size, design or complexity of the code makes understanding,

enhancement or testing difficult. The use of source checks complements the use of software metrics to improve the overall quality of the software.

And having this functionality integrated with the reverse engineering aspects of Imagix 4D leverages the efficiency in both the quality analysis and program understanding tasks.

Software Metrics -- The Metrics Tool is the primary display window for analysing the extensive set of quantitative measurements that Imagix 4D makes about the software. Within the Metrics Tool, the analyst can list, sort, compare and rank all the symbols of a specific type, by whichever set of software metrics the analyst are interested in analysing.

The software metrics are also integrated in other Imagix 4D displays. For example, the analyst can choose to have symbols in the Graph window coloured to indicate their value for software metric, so that the analyst is able to view the metric information within the context of the software's structure.

By tracking the metrics over time, during a development effort, the analyst is able to measure the progress of the development effort and to identify potential quality problems early, when they more easily corrected.

8.6.6 JMetric

Vendor: JMetric Inc.

Website: <http://www.it.swin.edu.au/projects/jmetric/products/jmetric/>

JMetric is a fully functional metrics collection and analysis tool. JMetric collects information from java source files and compiles a metrics model. This model is then populated with metrics information such as Lines of Code, Statement Count, LCOM, and Cyclomatic Complexity. From the model other metrics can then be calculated including number of classes, packages, methods etc. JMetric also provides statistical information in the form of drill down and cross section tables, charts, and raw metrics text.

- 100% Java Application.
- Metrics View in Tables and Charts.
- Capable of batch operation.
- Text file exports.
- Automatic Java Code Metrics Analyser.
- Cross Platform Support.
- Detailed Metrics at Project, Package, Class, Method and Variable Level.
- Provides quick metrics analysis via the Chart View.
- In depth analysis with the Project Tree Viewer.
- Easy to use tool and menu bars, or a faster Command Line.
- Supports Inner Classes and Abstract Classes.
- Coloured Highlighting in Tables and Tree View for better data visualisation.
- Java 1.1 support.

8.6.7 QSM - SLIM-Metrics

Vendor: QSM Inc.

Website: http://www.qsm.com/slim_metrics.html

SLIM-Metrics: Powerful Query Builder and Unlimited

Customizable Analysis Views

QSM ships SLIM-Metrics with a generous selection of predefined queries and analysis views, but the easy-to-use query builder also lets the programmer construct the own custom data sets without having to master complicated SQL statements. The flexible analysis views are designed to allow the analyst to modify the chart/report outline or swap chart metrics with a few simple mouse clicks.

SLIM-Metrics and SLIM-DataManager: Serious Tools for Serious Work

Some organisations are just getting measurement programs started. Others have plenty of data, but no easy way to organise and analyse it. SLIM-Metrics and SLIM-DataManager work together to support both types of organisations. SLIM-DataManager helps the analyst create a corporate database, then teams up with SLIM-Metrics to analyse the data and uncover key relationships and trends.

SLIM-Metrics helps answer the questions software professionals ask every day:

- How does our organisation stack up against the competition?
- Have our investments in process improvement improved our productivity?
- How is the quality of our delivered software changing over time?
- What impact are requirements churn and staff turnover having on our productivity?
- What kind of cost and schedule performance can we expect from different divisions?

SLIM-Metrics Features

- **Powerful query capability**

Create very specific subsets of the data and compare them to one another. Example: What is the average productivity for Client Server, C++ projects using 5 to 10 person teams compared to Web-based, Java applications?

- **Supports multiple views**

Analyse any measure or metric against any other(s). SLIM-Metrics lets the analyst create unlimited of charting/report views within a single workbook. The graph and report functions are highly customizable.

- **Industry benchmarks**

SLIM-Metrics includes recent industry trend lines for each of the standard measures and metrics. It allows to benchmark the project against QSM's database of over 6000 projects, or create own trend lines from the completed project data.

- **Statistical analysis tools**

Uncover the relationships hidden in the data. SLIM-Metrics contains a host of statistical and regression analysis features, including four different curve-fitting algorithms.

Compare and contrast data from up to five different data sets on a single chart. This is especially useful for internal benchmarking and competitive positioning.

Track a single project through multiple graphs. This is useful for visualizing trade-offs and cause/effect relationships.

- **SLIM-DataManager: Flexible, Expandable Data Collection**

SLIM-DataManager builds on a set of essential core measures. A larger set of predefined and user-defined metrics helps the analyst achieve increasing levels of process maturity. With SLIM-DataManager, the analysts have the freedom to collect the metrics that are most meaningful to the organisation.

SLIM-DataManager Features

- **Historical data capture**

Store all of the organisation's historical data (including the SEI Core Metrics, plus custom measures and metrics the analyst defines) in a single, open, relational database. Customizable database

Quickly extend the collected data to include any measures and/or metrics defined by the user, then use the powerful graphing and statistical tools in SLIM-Metrics to analyse the data.

SLIM-DataManager lets the analyst set up default values to describe the environment. The tabbed dialog interface makes data entry fast, consistent, and reduces time spent on error-checking. ODBC compliant file format.

The SLIM-DataManager database is OPEN. You can create seamless interfaces to other ODBC-compliant tools.

8.6.8 SDMetrics

Vendor: SDMetrics

Website: <http://www.sdmetrics.com/>

SDMetrics analyses the structural properties of the UML designs. Use object-oriented measures of design size, coupling, and complexity to

- establish quality benchmarks to identify potential design problems early on,
- predict relevant system qualities such fault-proneness or maintainability to better focus the review and testing efforts,
- increase system quality and quality assurance effectiveness, find more faults earlier and save development cost,
- refine the LOC or effort estimates for implementation and testing.

8.6.9 Metamata Metrics

Vendor: MetaMata

Website: <http://www.metamata.com>

Metamata Metrics provides an easy interface for developers to calculate meaningful object-oriented metrics for Java on partially written applications or portions of Java source code. Examples of such global complexity and quality metrics include:

- cyclomatic complexity,
- lines of code,
- weighted methods per class,
- coupling between classes, and
- depth of inheritance trees.

Metamata Metrics performs static analysis and does not require the developer to compile and run the source code. Metrics are displayed in an organised table, as well as in intuitive graphs. Reports can be generated in text format for easy import into spreadsheets or in HTML format for viewing through a web browser. Metamata Metrics is geared toward enhancing the software development process and quality of team development, optimising testing and maintenance resources and improving project planning activities. Metamata Metrics works with Java source code generated through any other tool or IDE. Metamata Browse, Metamata Debug, Metamata Audit and Metamata Metrics are part of an integrated set of tools for significantly improving the quality and productivity of advanced Java development. All of the suite components are written in Java 1.1, and are available for any Java compliant platform.

<http://www.metamata.com>

8.6.10 Scientific Toolworks, Inc.

Vendor: Scientific Toolworks, Inc.

Website: <http://www.scitools.com/>

Understand for Ada is an interactive development environment (IDE) tool offering reverse engineering, automatic documentation, code navigation and understanding, metrics, maintenance and cross reference tool for Ada 83 and Ada 95 source code. It is designed to help engineers who have inherited large amounts of Ada legacy code, or those whose Ada projects have grown to immense size or complexity.

The tool offers incremental analysis (parsing) of only source code that has changed, detailed graphical reverse engineering, highly interactive cross referencing, and detailed automatic documentation via HTML output. All parsing is compiler independent - Understand for Ada uses it's own Ada parser designed especially for its kind of analysis.

Views include call (invocation) trees, callby trees, generic instantiation trees, with trees, with-by trees, Ada Structure Graphs (ASG), coloured source browsing/editing, quick and smart project wide string searching and detailed HTML/text documentation of analysis information.

8.6.11 ObjectQuest Advisor

Vendor: Hallogram

Website: <http://www.hallogram.com/objectquest/>

ObjectQuest Advisor is an object-oriented metrics tool, which reverse engineers C++ source code into an architectural model and generates metrics for the application.

The architectural model, or blueprint, will help the analyst assess the structural integrity of an application. Eliminate "spaghetti" code forever!

The metrics will help the analyst assess the robustness of a design and measure progress toward achieving organisation's standards.

The interactive Object Relationship Diagram is aware of class relationships and allows the analyst to arrange the classes, set colours, relationship links, and layout of the diagram. You can select an individual class and the tool will highlight all other classes which use or are used by that class.

The Encapsulation Chart measures how well each class is encapsulated and highlights classes that exceed object-oriented standards. You will know immediately where the weak links in the application's architecture are and be able to monitor progress toward truly robust software.