Metrics-based Evaluation of Object-Oriented Software Development Methods

Reiner R. Dumke, Erik Foltin

University of Magdeburg
Postfach 4120, D-39016 Magdeburg, Germany
Tel: +49-391-67-18828, Fax: +49-391-67-12810
email: {dumke, foltin}@irb.cs.uni-magdeburg.de
http://irb.cs.uni-magdeburg.de/sw-eng/us/

Preprint Nr. 10, Fakultät für Informatik, 1996

Contents

Abstract

The efficiency of software development (i. e. to produce good software products based on an efficient software process) must be controlled by a quantification of the software development methodologies. The description of object-oriented (OO) methods or comparisons of some of these methods are usually given by a listing of their features. These presentations describe the functionality of a particular development method, but often fail to address quality issues like efficiency, maintainability, portability, maturity etc.
The quantification by means of software measurement needs a unified strategy, methodology or approach as one important prerequisite to guarantee the goals of quality assurance, improvement and controlled software management to be achieved. Nowadays, plenty of methods such as measurement frameworks, maturity models, goal-directed paradigms, process languages etc. exist to support this idea.
This paper describes an object-oriented approach of a software measurement framework aimed at evaluating OO development methods themselves. It reasons the applicability of metrics-based evaluation as indicator for the quality assurance of the OO development process.

Keywords: object-oriented software development, software quality, process quality, measurement framework

1 Introduction

The benefits of the use of the object-oriented software development techniques are widely discussed in many papers ([Brown 96a], [Hitz 95], [Jacobson 95], [Jones 94], [Moser 96] etc.). However, most of these discussions and presentations only enumerate the features of the OO development methods and programming environments, e. g. in [Embley 95] as

Feature Name	***OOSA(Embly et al.)***	***OMT (Rumbaugh et al.)***	***OOSA (Shlaer, Mellor)***	***OOA (Coad, Yourdon)***	***OOA/D (Booch)***	***OORA (Firesmith)***
Objects	Yes	Yes	Yes	Yes	Yes	Yes
Object classes	Yes	Yes	Yes	Yes	Yes	No
Relationships	Yes	Yes	Yes	Yes	Yes	Yes
Relat. Object classes	Yes	Yes	No	No	Yes	Yes
Full integrated submodels	Yes	No	No	Yes	No	No
Aggregation	Yes	Yes	Yes	Yes	Yes	Yes
Gen/Spec	Yes	Yes	Yes	Yes	No	Yes
Interobject concurrency	Yes	Yes	Yes	Yes	Yes	Yes
Intraobject concurrency	Yes	Yes	No	No	No	Yes
Exceptions	Yes	No	No	No	No	Yes
Temporal conditions	Yes	No	No	No	Yes	No
Interaction details	Yes	No	No	No	No	No
Attributes or methods	No	Yes	Yes	Yes	Yes	Yes
Method classification	No	No	No	No	Yes	Yes
etc.						

and in the presentation by Khan et al. [Khan 95] given the following table of OO features.

OOP language feature		**C++**	**Object Pascal**	**Smalltalk**	**CLOS**
Abstraction	Instance variables	Y	Y	Y	Y
	Instance methods	Y	Y	Y	Y
	Class variables	Y	N	Y	Y
	Class methods	Y	N	Y	Y
Encapsulation	Attributes	public,private protected	public,private	private	reader,writer accessor
	Methods	public,private protected	public,private	public	public
Moduls		files	units	none	packages
Inheritance		multiple	single	single	multiple
Polymorphism		single	single	single	multiple
Generic units		Y	N	N	Y
Strongly typed		Y	Y	N	optional
Metaclass		N	N	Y	Y
Class library (# classes)		> 300	< 100	> 300	< 100

Of course, these features are essential with respect to the implementable semantics of an object-oriented system. But the enumeration of feature is often not sufficient to explain about the size, complexity, and quality characteristics of the implemented products or of the development process itself. We do not find enough information about the process maturity and process quality that gives reasons for choosing a specific method. Hence, we will discuss some essential aspects for a metrics-based object-oriented method evaluation [DuFW 95].

2 Evaluation and Metrication of one OO Method - An Example

2.1 The General Approach

The principal ideas of this measurement framework are given in [DFKW 96] and are suited to understand and to quantify the chosen the object-orientated method. A standardized metric set for OOSE does not yet exist (only a metrics definition standard [IEEE 93]). Therefore, it is necessary to define metrics and to analyze them. The validation of this metric set is the main problem in the application of software metrics. The software measurement is directed to three main components in the (object-oriented) software development (see also [Fenton 97])

- the **process measurement** for understanding, evaluation and improvement of the deve-lopment method,

- the **product measurement** for the quantification of the product (quality) characteristics and validation these measures,

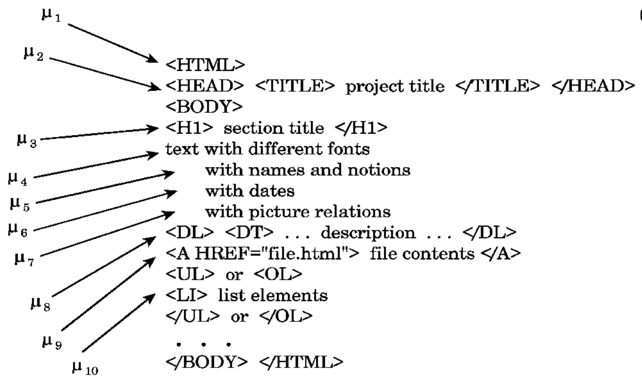
- the **resource measurement** for the evaluation of the supports (CASE tools, measurement tools etc.) and the chosen implementation system.

Some main ideas and some short results of an application of the Software Measurement Laboratory of the University of Magdeburg (SMLAB) is given in the following (see also http://irb.cs.uni-magdeburg.de/ sw-eng/us/).

2.2 The Process Measurement

The chosen OO software engineering method is the Coad/Yourdon approach (described in [Coad 93]). It begins with the transformation of the problem definition into a graphical representation with an underlying documentation. The documentation contains all information that cannot be presented in the drawings. The drawings (which are possible in some variants) and the documentation constitute the OOA model. In a first evaluation of this method we can establish the following goals of the process measurement and the realized activities:

How we can measure the object definition process? This question leads us to the first step of the software development - the problem statement. We need a computational stored problem definition to measure the object definition. The SMLAB problem definition must be accessible to all members



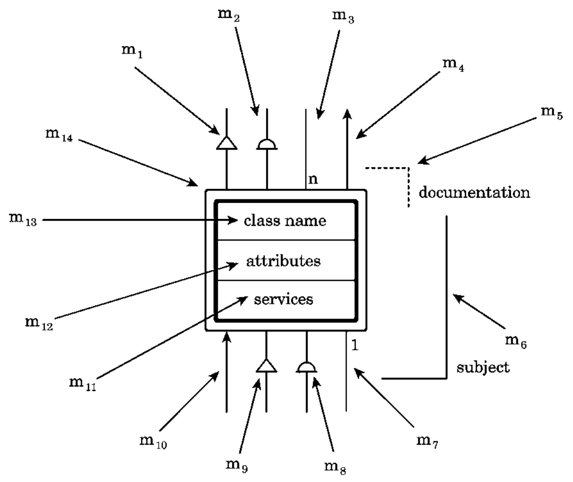
```
<HTML>
<HEAD> <TITLE> project title </TITLE> </HEAD>
<BODY>
<H1> section title </H1>
text with different fonts
      with names and notions
      with dates
      with picture relations
<DL> <DT> . . . description . . . </DL>
<A HREF="file.html"> file contents </A>
<UL> or <OL>
<LI> list elements
</UL> or </OL>
  .  .  .
</BODY> </HTML>
```

e software engineering team and the document itself is an essential source for many outputs such as milestones or an overview for some administrational purposes. Therefore, we decided for a html file set of the World-Wide Web Intranet as a ***living document system.*** The elements of our problem statement are a *list of contents* (as problem description, constraints, given situation, functional requirements, management requirements (controlling and quality)) and a *list of components* (as notions, names, dates, pictures, and (hypertext) relations). An implementation of a ***measurement tool to measure the problem definition*** (PDM) was necessary [Foltin 95]. A more detailed list of life cycle metrics types is given in the following (see also [DFKW 96]).

PROCESS LIFE CYCLE METRICS:

- ***Problem definition metrics***
  - kinds of problem definitions
  - used standards for problem definitions
  - tool-based level
  - stability metrics
- ***Requirement analysis and specifi-cation metrics***
  - flow level from the problem definition
  - average participatory level
  - team structure
  - development methods metrics
  - level of (cost) estimation methods
  - integration level
  - test cases metrics
- ***Design metrics***
  - automatization level
  - knowledge-based level
  - (class) library metrics
  - reusability level
- ***Implementation metrics***
  - generation level
  - average code quality level
  - test metrics
  - performance metrics
  - distribution level
- ***Maintenance metrics***
  - error management metrics
  - changeability metrics
  - extendibility metrics
  - tuning metrics
  - reliability metrics
  - configuration control metrics

How we can measure the OOA/OOD model itself? The OOA model must be ‘open’ for measurement. This is the case because the models of the used CASE tool - the ObjecTool - are

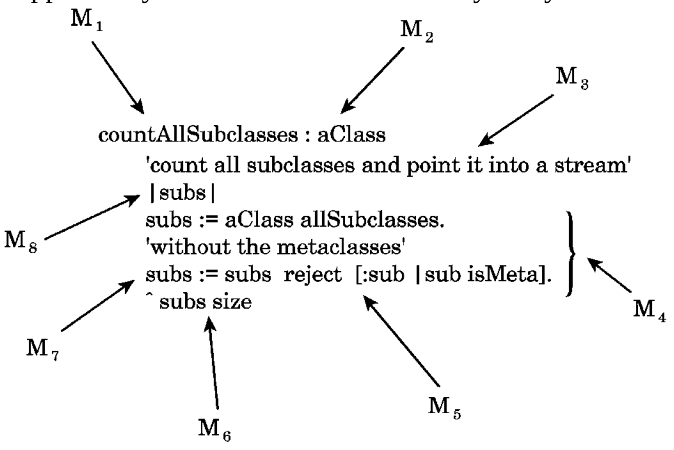


stored in a set of files in an interpretable descriptive language. So, the ***measurement tool*** OOM [Papritz 93] was implemented to measure the OOA model. The evaluation of the OOA step proved a missing inheritance documentation and a rather small and not very helpful critique generated by the tool that is only directed to an object/class symbol. Further, the estimation of effort, costs and quality is not possible in this development phase without prior knowledge about similar projects (a general problem in the OO software engineering). The OOD step ensures a full continuity with the OOA step. It extents (or updates) the OOA model with respect to the chosen implementation environment, i. e. by including libraries for the realization of the user interface or data storage engines. The resulting OOD model is the primary model used later in the maintenance phase. Hence we do not have a method independent specification. There is also no mechanism provided to relate the design to the object-oriented implementation (programming) system. Therefore, some form of browsing the OOP system is required in the OOD phase. To support this activity we have implemented the ***OOC tool for browsing in the Smalltalk*** class library [Lubahn 94]. In general it is necessary to quantify the management activities based on the following metrics [DFKW 96].

PROCESS MANAGEMENT METRICS:

- ♦ *Project Management Metrics:*
  - ***milestone metrics***
    - ∗ number of milestones
    - ∗ number of proved requirements per milestone
    - ∗ controlling level metrics
  - ***risk metrics***
    - ∗ probability of resources availability
    - ∗ probability of the requirements validity
    - ∗ risk indicators (long schedules, inadequate cost estimating, excessive paperwork, error-prone modules, canceled projects, excessive schedule pressure, low quality, cost overruns, greeting user requirements, excessive time to market, unused or unusable software, unanticipated acceptance criteria, hidden errors)
    - ∗ application risk metrics
  - ***workflow metrics***
    - ∗ walkthrough metrics
    - ∗ traceability metrics
    - ∗ variance metrics
  - ***controlling metrics***
    - ∗ size of control elements
    - ∗ structure of control elements
    - ∗ documentation level
    - ∗ tool application level
  - ***management database metrics***
    - ∗ data quality metrics
    - ∗ management data complexity
    - ∗ data handling level (performance metrics)
    - ∗ visualization level
    - ∗ safety and security metrics
- ♦ *Quality Management Metrics:*
  - ***customer satisfaction metrics***
    - ∗ characteristics size metrics
    - ∗ characteristics structure metrics
    - ∗ empirical evaluation metrics
    - ∗ data presentation metrics
  - ***review metrics***
    - ∗ number of reviews in the process
    - ∗ review level metrics
    - ∗ review dependence metrics
    - ∗ review structure metrics
    - ∗ review resources metrics
  - ***productivity metrics***
    - ∗ actual vs. planned metrics
    - ∗ performance metrics
    - ∗ productivity vs. quality metrics
  - ***efficiency metrics***
    - ∗ time behavior metrics
    - ∗ resources behavior metrics
    - ∗ actual vs. planned metrics
  - ***quality assurance metrics***
    - ∗ quality evaluation metrics
    - ∗ error prevention metrics
    - ∗ measurement level
    - ∗ data analysis metrics
- ♦ *Configuration Management Metrics:*
  - ***change control metrics***
    - ∗ size of change
    - ∗ dependencies of changes
    - ∗ change interval metrics
    - ∗ revisions metrics
  - ***version control metrics***
    - ∗ number of versions
    - ∗ number of versions per customer
    - ∗ version differences metrics
    - ∗ releases metrics (version of architecture)
    - ∗ data handling level

How we can measure the OOP system? Here we must choose a special OOP system or an OOP language. The ObjecTool is intended to support C++ or Smalltalk implementations. The evaluation of this phase indicates that a direct re-engineering of the OOD based on experience of the OOP is not supported by the tool. Therefore it is very likely to introduce maintenance problems at this stage. The knowledge of the existing OOP systems or libraries is one of the main obstacles for an efficient OO software engineering. The measures added in this development phase are mainly code measures. For the quality measurement of the process we use the ***development complexity*** (see [DKFW 96]) to assess the used methods and tools and their structure. Other measures (performance etc.) have not been included in this first approach of development complexity evaluation. The measurement tools used in this sample evaluation were implemented in the same method and programming language to reduce development complexity. We have implemented a C++ measurement tool [Kuhrau 94] in C++ and a Smalltalk measurement extension [Heckendorff 95]. The given description of the process measurement is a good example for the method understanding. Some missing tools for the completion of an measurable OOSE method on this basis have been designed and implemented. In general, the following measures help to quantify the maturity of the development process [DFKW 96].



```
countAllSubclasses : aClass
    'count all subclasses and point it into a stream'
    |subs|
    subs := aClass allSubclasses.
    'without the metaclasses'
    subs := subs  reject  [:sub |sub isMeta].
    ^ subs size
```

PROCESS MATURITY METRICS

- ***Organization metrics***
  - personal structure metrics (characteristics of the development teams and hierarchy, CSCW level, staff experience)
  - management metrics (existence or level of the project, quality and configuration management)
- ***Resources, personnel and training metrics***
  - development team metrics (experience, efficiency, flexibility)
  - training's metrics (cycles of courses, necessary enrollments)
  - availability of computer resources
  - brainstorming metrics
- ***Technology management metrics***
  - evaluations of the technology level
  - technology replacing metrics
- ***Documented standards metrics***
  - standards application metrics (IEEE, ANSI, national etc.)
  - number of used standards (for documentation, life cycle, reviews, and maintenance)
- ***Process controlling metrics***
  - management support metrics
  - productivity metrics
  - efficiency metrics
  - process quality metrics
  - actual vs. planned metrics (especially error estimation etc.)
  - traceability measures
- ***Data management and analysis metrics***
  - data management level (metrics data base, evaluation techniques etc.)
  - use of statistical methods metrics
  - visualization level metrics

2.3 The Product Measurement

For product measurement the measure mutations were analyzed, for example the number of notions/names in the problem definition (#notions/names) was related to the number of defined classes in the OOA/OOD model and in the implementation. Other measurements relate adjectives/adverbs to class attributes or variables, verbs to the classes services or methods and dates/constraints to the model documentation and implementation. We can see the essential approach in analyzing the mutations of the μ, m, and M measures. According to [ISO9126 91], the evaluation of the product quality in every development phase is defined as comprehensibility, clarity and usability of the problem statement on the basis of the measures use frequency, availability, size and structure; the completeness, conformity and feasibility for the OOA/OOD phase based on measures consistency, performance, size and structure; and the understandability, stability and effort for the OOP phase on the basis of measures testability, size, structure and reusability. Most of these measures are based on an ordinal scale and can therefore be used to classify the achieved quality. The general metrication of the software product is summarized in the following table[DFKW 96].

PRODUCT METRICS

Size Metrics:

- ***number of elements***
  - lines of code
  - number of documentation pages
  - etc
- ***development metrics***
  - number of test cases
  - consumption of resources metrics
- ***size of components***
  - number of modules/objects
  - average size of components

Architecture Metrics:

- ***components metrics***
  - number of (language) paradigms
  - part of standard software
  - quality level
- ***architecture characteristics***
  - open system level
  - integration level
- ***architecture standard metrics***
  - used standards metrics
  - part of standardization

Structure Metrics:

- ***component characteristics***
  - number of structure elements
  - part of component per structure element
  - average connection level
- ***structure characteristics***
  - composition level
  - decomposition level
  - component coupling metrics
  - tree structure metrics
- ***psychological rules metrics***
  - orientation for structure width
  - orientation for structure depth
  - visualization level

Quality Metrics:

- ***functionality metrics***
  - suitability
  - accuracy
  - interoperability
  - compliance
  - security
- ***reliability metrics***
  - maturity
  - fault tolerance
  - recoverability
- ***usability metrics***
  - understandability
  - learnability
  - operability
- ***efficiency metrics***
  - time behavior
  - resource behavior
- ***maintainability metrics***
  - analyzability
  - changeability
  - stability

  testability
- ***portability metrics***
  - adaptability
  - installability
  - conformance
  - replaceability

Complexity Metrics:

- ***computational complexity metrics***
  - algorithmic complexity
  - informational complexity
  - data complexity
  - combinatorial complexity
  - logical complexity
  - functional complexity
- ***psychological complexity metrics***
  - structural complexity
  - flow complexity
  - entropic complexity
  - cyclomatic complexity
  - essential complexity
  - topologic complexity
  - harmonic complexity
  - syntactic complexity
  - semantic complexity
  - perceptional complexity
  - organizational complexity
  - diagnostic complexity

2.4 The Resource Measurement

One essential aspect in the introduction of OO software engineering are the initial measures of the chosen resources (CASE tools, measurement tools programming environment etc.). In accordance with our validation aspect we can quantitatively evaluate the usefulness of the chosen object-oriented programming system. The evaluation of C++ or Smalltalk/V for Windows for example shows functional characteristics and we can expect a lot of maintenance effort.

The metrication aspects of the software development resources are given in the following [DFKW 96].

RESOURCES METRICS

Personnel Metrics:

- ♦ ***programming experience metrics***
  - programming language experience
  - development methods experience
  - management experience
- ♦ ***communication level metrics***
  - teamwork experience
    - communication hardware/ software level
    - personal availability
- ♦ ***productivity metrics***
  - size productivity
  - productivity statistics
  - quality vs. productivity
- ♦ ***team structure metrics***
  - hierarchy metrics
  - team stability metrics

Software Metrics:

- ♦ ***performance metrics***
  - method productivity
  - programming language productivity
  - development environment level
- ♦ ***paradigm metrics***
  - development method trends
  - programming languages trends
  - paradigm quality
- ♦ ***replacement metrics***
  - level of software portability
  - software development complexity

Hardware Metrics:

- ♦ ***performance metrics***
  - computer performance
  - network performance
  - benchmarks
  - performance profile
- ♦ ***reliability metrics***
  - Mean Time to Failure (MTTF)
  - Mean Time Between Failure (MTBF)
  - Mean Time To Repair (MTTR)
  - Mean Recurrence Time (MRT)
  - Mean Waiting Time in Error States (MWTE)
- ♦ ***availability metrics***
  - time availability
  - security constraints
  - local availability

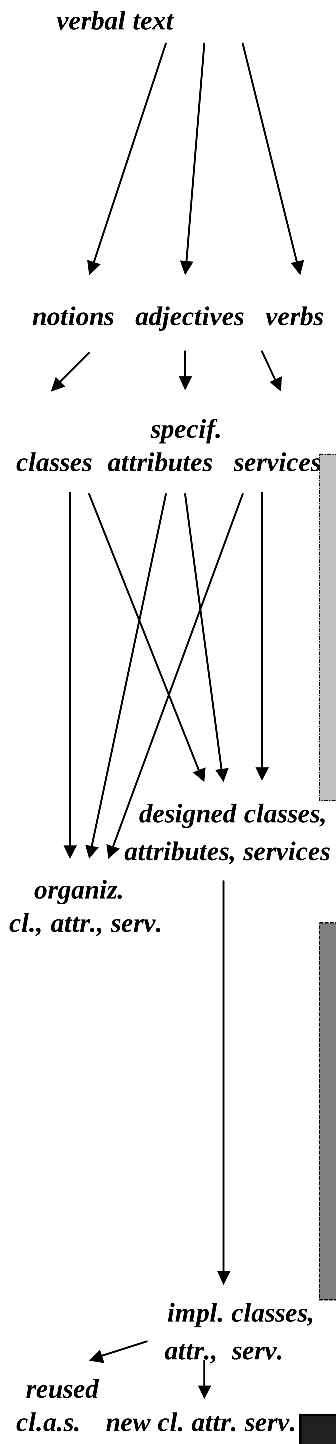
2.5 Conclusions

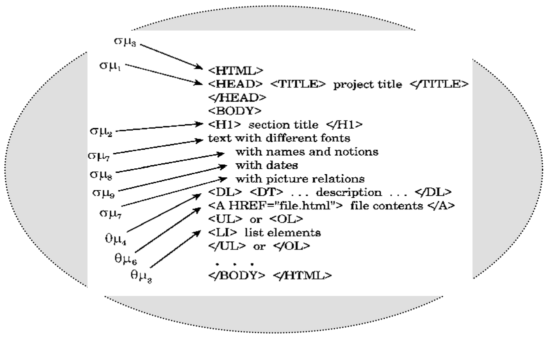
Briefly stated, the metrication of a development method has to include the definition/ application of (object-oriented) software metrics for the elements/components of the method as well as the ***workflow*** of the requirements/elements along the development phases and life cycle activities. A simplified description is given in the following based on the experience from our SMLAB project [DuWi 96].

Note, that the presentation covers ***only the evaluation of the product structure and architecture*** metrication aspects.

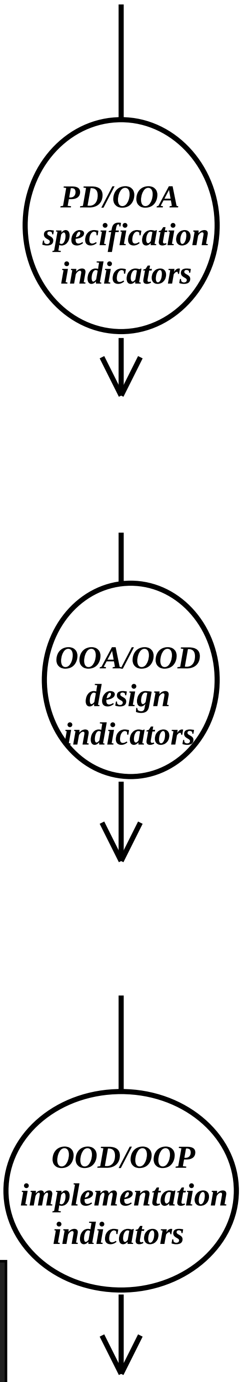
Problem definition (PD)

(as HTML document system):

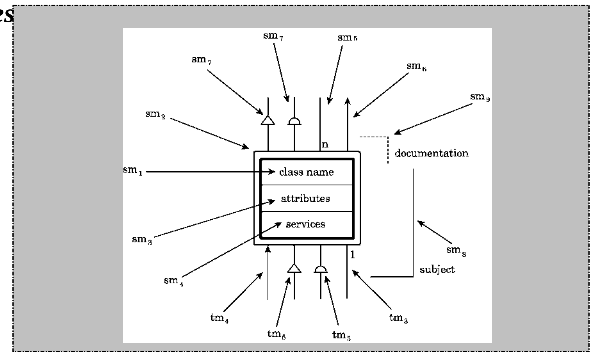




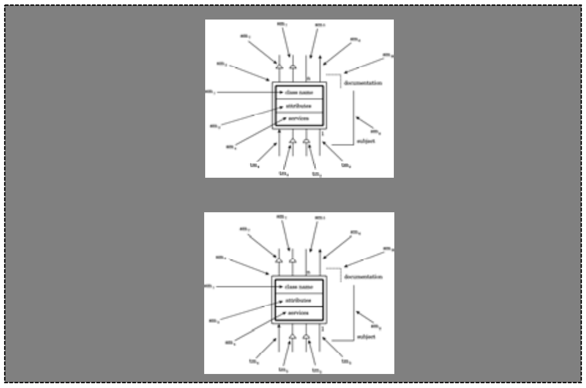
```
<HTML>
<HEAD> <TITLE> project title </TITLE>
</HEAD>
<BODY>
<H1> section title </H1>
text with different fonts
    with names and notions
    with dates
    with picture relations
<DL> <DT> . . . description . . . </DL>
<A HREF="file.html"> file contents </A>
<UL> or <OL>
<LI> list elements
</UL> or </OL>
  . . .
</BODY> </HTML>
```


PD/OOA specification indicators
OOA/OOD design indicators
OOD/OOP implementation indicators

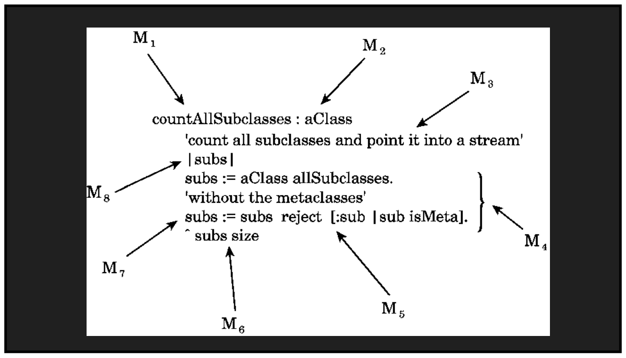
OOA model in the Coad/Yourdon approach (drawing element):



OOD model in the same approach (the same drawing element):



Implementation in Smalltalk (a class method):



```
countAllSubclasses : aClass
    'count all subclasses and point it into a stream'
    |subs|
    subs := aClass allSubclasses.
    'without the metaclasses'
    subs := subs reject [:sub |sub isMeta].
    ^ subs size
```

In a first approximation the following indicators are used to characterize the aspects typical to OO software engineering in the given development method. The ***specification indicators*** as

- class definition indicator **(CDI)** as
  number of defined classes per number of notions,
  (CDI_{SMLAB} = 0.02)

- attribute definition indicator **(ADI)** as
  number of defined attributes per number of adjectives or predicates,
  (ADI_{SMLAB} = 0.03)

- service definition indicator **(SDI)** as
  number of verbs or adverbs per number of defined services,
  (SDI_{SMLAB} = 0.06).

The ***design indicators*** as

- class modification indicator **(CMI)** as
  number of organizational classes per number of all designed classes,
  (CMI_{SMLAB} = 0.33)

- attribute modification indicator **(AMI)** as
  number of organizational attributes per number of all designed attributes,
  (AMI_{SMLAB} = 0.22)

- service modification indicator **(SMI)** as
  number of organizational services per number of all designed services,
  (SMI_{SMLAB} = 0.21).

And the ***implementation indicators*** as

- class implementation indicator **(CII)** as
  number of new implemented classes per number of designed classes,
  (CII_{SMLAB} = 0.31)

- attribute implementation indicator **(AII)** as
  number of new implemented attributes per number of designed attributes,
  (AII_{SMLAB} = 0.51)

- service implementation indicator **(SII)** as
  number of new implemented services per number of designed services,
  (SII_{SMLAB} = 0.22).

We want to stress the point that these indicators are intended to reflect relations over all development phases in a special workflow manner, both for the characterization of the product type (degree of the class reuse, for instance) and of the process efficiency (i. e. degree of the automatization).

3 Recent Work in OO Software Metrics

3.1 General Approaches

The recent work in software measurement for object-oriented software development can be subdivided in:

- ***statistical analysis*** of elements of an object-oriented development system (Smalltalk-80) by Rochache [Rocache 89]; of a C++ communication system by Szabo and Khoshgoftaar [Khoshgoftaar 94]; or for different metrics and different C++ libraries and Eiffel programs by Abreu and Melo [Abreu 96],

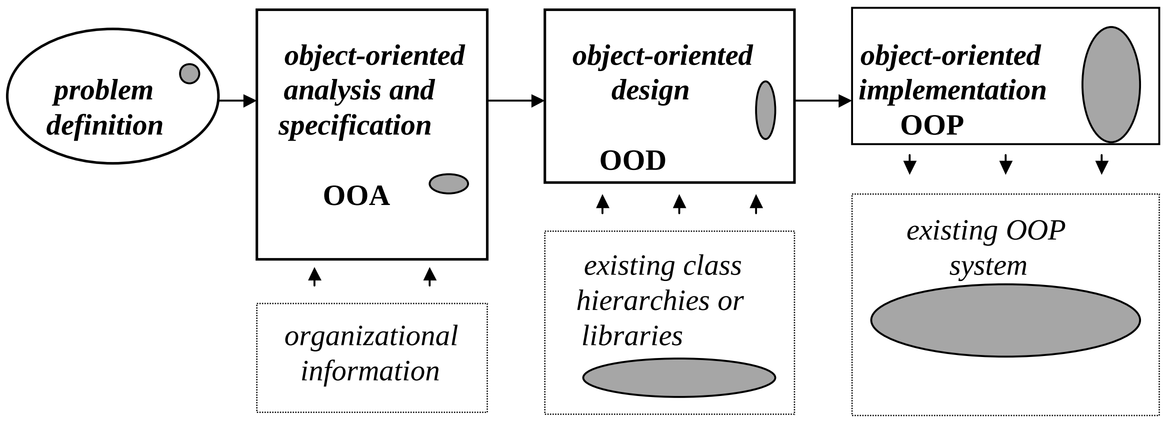
- ***metrics set definitions*** by Abreu and Carapuca in [Abreu 94] for C++ with the two vectors category (design, size, complexity, reuse, productivity, and quality), and granularity (system, class, and method); by Binder in [Binder 94] as a set of C++ metrics to measure encapsulation, inheritance, polymorphism, and complexity; or by Arora et al. in [Arora 95] for real-time software design in C++, by Dumke et al. in [DFKW96] for all phases of the object-oriented development, and by Lorenz and Kidd in [Lorenz 94] as a metrics set that can be used for the C++ language and Smalltalk,

- ***OO aspect measurement*** by Ott et al. in [Bieman 94] or by Lee et al. in [Lee 95] or by Hitz and Montazeri in [Hitz 95] or by Han et al. in [Han 94] of class coupling and cohesion; or by Bieman in [Kurananithi 93], John in [John 95], and Pant et al. in [Pant 96] to measure reusability, or by Chung et al. [Chung 95] to measure the inheritance complexity, or to support object-oriented testing (Chung and Lee in [Chung 94]) and maintenance (Lejter in [Lejter 92]),

- ***information theoretical approaches*** like the measure of conceptual entropy by Dvorak in [Dvorak 94] or the cognitive approach by Henderson-Sellers et al. in [Henderson 96] with the landscape idea along the method routes or the learnability aspects in the use of class libraries in [Lee 94], and

- ***validation of enclosed approaches*** by Chidamber and Kemerer in [Chidamber 94] as an approach of metrics definition based on a measurement theoretical view (with ‘’viewpoints’’ as empirical evaluation), the extension of these measures by Li et al. in [Li 95], the (algebraic) analysis approach of Churcher and Shepperd in [Churcher 95], and the investigations of Zuse in [Zuse 94] and [Zuse 97].

The grey areas in the following simplified object-oriented software development scheme indicate the shared existing metrics approaches.


problem definition
object-oriented analysis and specification
OOA
object-oriented design
OOD
object-oriented implementation
OOP
organizational information
existing class hierarchies or libraries
existing OOP system

3.2 Metrics for OO Systems

For a narrowly-focused presentation of the existing OO metrics we use our general metrics classification [DFKW 96] as

PROCESS METRICS	PRODUCT METRICS	RESOURCES METRICS
Maturity Metrics	***Size Metrics***	***Personnel Metrics***
- organization metrics	- elements counting	- programmer experience metrics
- resources, personnel and training metrics	- development size metrics	- communication level metrics
	- size of components metrics	- productivity metrics
- technology management metrics	***Architecture Metrics***	- team structure metrics
- documented standards metrics	- components metrics	***Software Metrics***
- process controlling metrics	- architecture characteristics	- performance metrics
- data management and analysis	- architecture standards metrics	- paradigm metrics
Management Metrics	***Structure Metrics***	- replacement metrics
- milestone metrics	- component characteristics	***Hardware Metrics***
- risks metrics	- structure characteristics	- performance metrics
- workflow metrics	- psychological rules metrics	- reliability metrics
- controlling metrics	***Quality Metrics***	- availability metrics
- management data base metrics	- functionality metrics	
- quality management metrics	- reliability metrics	
- configuration management m.	- usability metrics	
Life Cycle Metrics	- efficiency metrics	
- problem definition metrics	- maintainability metrics	
- requirement analysis and specification metrics	- portability metrics	
	Complexity Metrics	
- design metrics	- computational complexity metrics	
- implementation metrics	- psychological complexity metrics	
- maintenance metrics		

Based on the recent work on OO metrics, we can establish the following metrics to evaluate the OO products and the processes including some empirical evaluations.

Process maturity metrics: (0)

Process management metrics: (4)

- person-days per class (PDC) *(product class ≤ 40 [Lorenz 94])*
- change dependency between classes (CDBC) *(transparency principle [Hitz 95])*
- cognitive complexity (CCM) *(case study based [Cant 94])*
- time to fix the known errors (TKE) in minutes *(minimizing principle [Harrison 96])*

Process life cycle metrics: (10)

- conceptual specificity (OOCM) *(difference principle [Dvorak 94])*
- conceptual consistency (OOCM) *(difference principle [Dvorak 94])*
- conceptual distancy (OOCM) *(difference principle [Dvorak 94])*
- number of scenario scripts (NSS) *(transparency principle [Lorenz 94])*
- unit repeated inheritance (URI) testing *(test coverage Cn, n>2 [Church 94])*
- number of methods overridden (NMO) *(transparency principle [Lorenz 94])*
- number of methods inherited (NMI) *(transparency principle [Lorenz 94])*
- number of methods added (NMA) *(transparency principle [Lorenz 94])*
- number of modifications requests (MR) *(minimizing principle [Harrison 96])*
- time to implement modifications (TMR) *(minimizing principle [Harrison 96])*

Product size metrics: (17)

- number of abstract classes [Dumke 94]
- number of object/classes [Dumke 94]
- total number of (class/instance) attributes (NIV, NCV [Lorenz 94])
- total number of (class/instance) services/methods (NOM, [Li 95]; NIM,NCM [Lorenz 94]) *(Smalltalk$_{initial}$ =22*#classes [LaLonde 94])*
- number of object connections [Dumke 94]
- number of message connections [Dumke 94]
- number of the subclasses [Dumke 94]
- number of the subject domains [Dumke 94]
- code/text lines of method [Dumke 94]
- length of attribute name [DFKW 96]
- number of ADTs defined in a class (DAC) *(transparency principle [Li 95])*
- number of semicolons in a class (SIZE1) *(case study [Li 95])*
- number of attributes + number of local methods (SIZE2) *(case study [Li 95])*
- number of root classes *(case study = 3 [Lake 92])*
- number of key classes (NCK) *(completeness principle [Lorenz 94])*
- number of support classes (NSC) *(completeness principle [Lorenz 94])*
- number of subsystems (NOS) *(transparency principle [Lorenz 94])*

Product architecture metrics: (2)

- verbatim reuse (VR) *(optimization principle [Bieman 95])*
- generic reuse (GR) *(optimization principle [Kurananithi 93])*

Product structure metrics: (22)

- average number of attributes per class [Dumke 94]
- average number of services per class *(not more than 20 [Lorenz 94])*
- average number of object connections per class [Dumke 94]
- average number of message connections per class [Dumke 94]
- maximal depth of the inheritance (DIF) *(applica-tion$_{initial}$ 3 [Chidamber 94])*
- method hiding factor (MHF) *(initial 19,6 % [Abreu 95])*
- attribute hiding factor (AHF) *(initial 79,7 % [Abreu 95])*
- method inheritance factor (MIF) *(initial 73,5 % [Abreu 95])*
- attribute inheritance factor (AIF) *(initial 56,2 % [Abreu 95])*
- polymorphism factor (POF) *(initial 6,5 % [Abreu 95])*
- coupling factor (COF) *(initial 10,8 % [Abreu 95])*
- number of children (NOC) *(initial 0.9 [Chidamber 97])*
- coupling between object classes (CBO) *(application$_{initial}$ 1.3 [Chidamber 97])*
- response for a class (RFC) *(initial 10 [Chidamber 97])*
- lack of cohesion (LCOM) *(initial 4.1 [Chidamber 97])*
- average code/text lines of methods *(Smalltalk/V$_{initial}$ = 3 [Wilde 92], Smalltalk=8, C++=24 [Lorenz 94])*
- strong functional cohesion (SFC) *(example$_{demo}$ 0.18 [Bieman 94])*
- I-based coupling (ICP) *(example$_{demo}$ [Lee 95])*
- I-based cohesion (ICH) *(example$_{demo}$ [Lee 95])*
- strength of cohesion as part of operations that apply one ADT domain *(case study in C++: 26% [Han 94])*
- method coupling *(non-coupling (nc), concealed coupling (cc) (only directly operation use), partial coupling (pc) (also general operation use), open coupling (oc) (also domain use) case study in C++: nc=20%, cc=10%, pc=45%, oc=25% [Han 94])*
- locality of data (LD) *(transparency principle [Hitz 95])*
- computing cohesion (CH) *(maximum = 1 [Wech 96])*

Product quality metrics: (6)

- understandability (= average number of attributes per class, average LOC per method) *(maximum reducing [Barnes 93])*
- average length of classes/attributes/methods names *(general mnemonic aspects)*

- test order for class firewall (CFW) *(case study: 192 stubs per test order [Kung 95])*
- number of known errors (KE) during testing *(minimizing principle [Harrison 96])*
- percentage of commented methods (PCM) *(transparency principle [Lorenz 94])*
- problem reports per class (PRC) *(empirical criteria [Lorenz 94])*

Product complexity metrics: (8)

- weighted method per class (WMC) *(initial 10 [Chidamber 94])*
- weighted attribute per class (WAC) *(method evaluation case study [Sharble 93])*
- leveraged reuse (LR) *(optimization principle [Bieman 95])*
- subjective assessment of complexity (SC) *(ordinal: 1...5 [Harrison 96])*
- message passing coupling (MPC) *(transparency principle [Li 93])*
- number of tramps (NOT) *(method evaluation case study [Sharble 93])*
- operation complexity (OC) *(case study = 78.5 [Chen 93])*
- attribute complexity (AC) *(case study = 2.2 [Chen 93])*

Resource personnel metrics: (1)

- classes per developer (CPD) *(empirical criteria [Lorenz 94])*

Resource software metrics: (2)

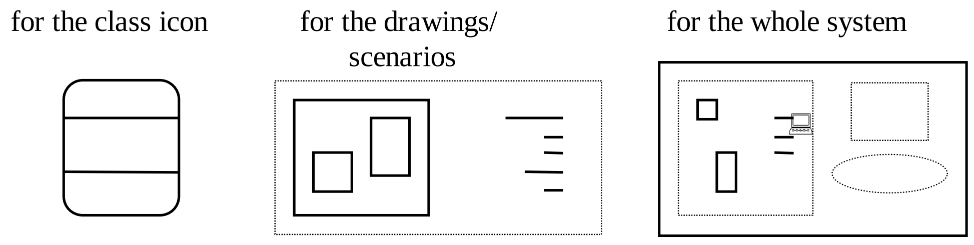
- paradigm related development time *(case study: OO vs. procedural [Lee 94])*
- violations of the law of demeter (VOD) *(method evaluation case study [Sharble 93])*

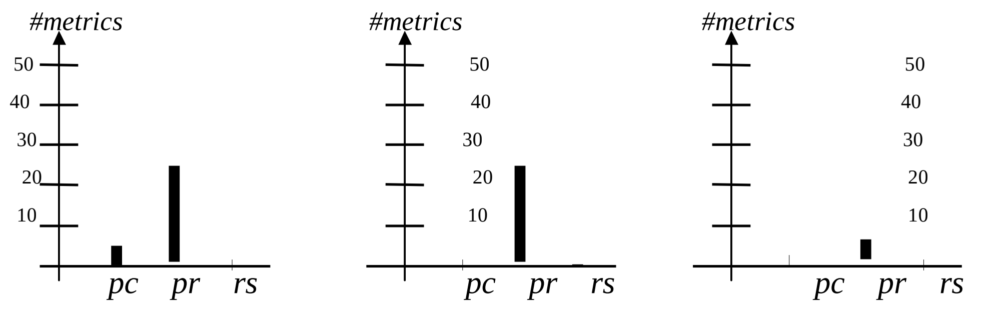
Total number of OO metrics: 72

3.3 Conclusions

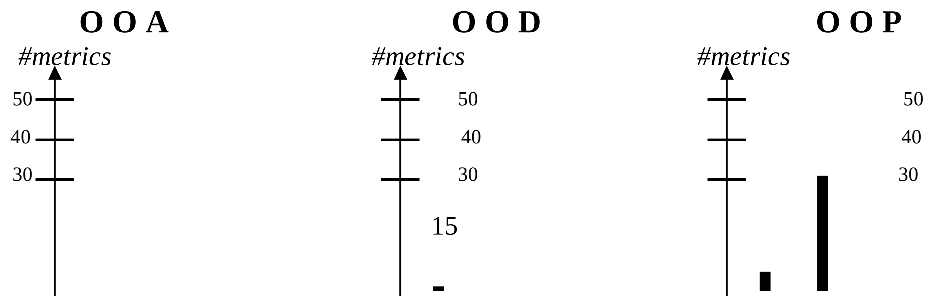
The charts below characterize the facilities and the situation in the OO metrics area. Note, that the charts provide only an approximate overview about the metrics situation. We use **pc** for the process metrics, **pr** for the product metrics, and **rs** for the resources metrics.

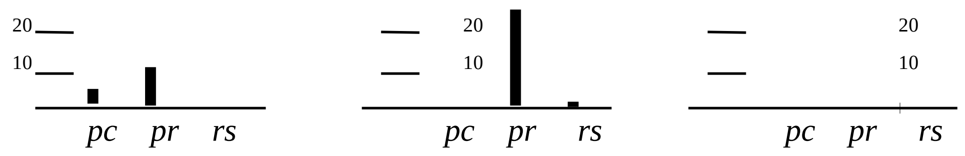
System Model Granularity



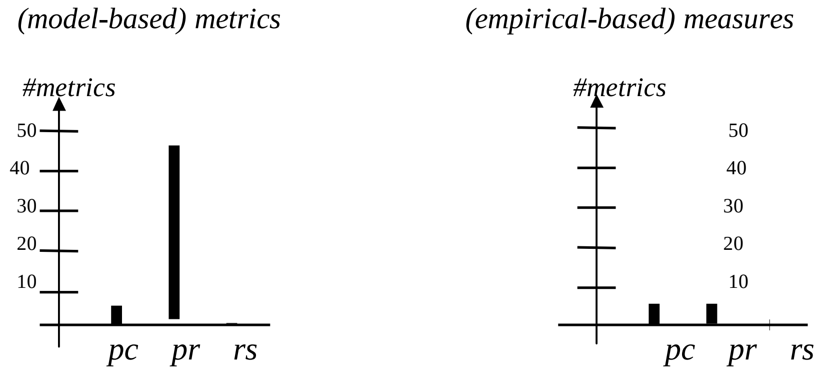


Life Cycle Phase Related





Measurement Area Related



Furthermore, we can establish the following general characteristics of OO software metrics:

- most of the metrics are ***not language independent*** (some of them are especially C++ related),

- most of the OO metrics are metrics and ***not measures*** (they are relations or quotients of OO characteristics),

- the ***empirical evaluations*** are divided into

  * ***not available*** (only feasibility test of the metric for intuitive (quality) aspects),

  * a general ***principle of minimizing*** or maximizing,

  * ***case-study***-based as sample initial values,

  * ***experience-based*** as classification or evaluation values for a quality ‘’area’’,

  * ***unit*** including ratio scaled forms;

- comparing the metrics set with our product metrics classification tree yields a lack of knowledge especially in the following areas

  * very few ***documentation*** metrics,

  * rare ***architecture*** metrics,

  * only a ***few empirical evaluations*** for the quality-oriented metrics are given;

- some metrics are given in ***functional form*** (#methods = 22 $\times$ #classes) or ***tuple form*** (understandability = (average #attributes, average $\text{LOC}_{\text{method}}$)),

- the OO metrics are defined ***for different kinds of development*** components but not for monitoring the development process over time,

- the metrics are mostly used for an assessment but ***not for measurement-based controlling,***

- in general, the given OO metrics are ***not really object-oriented*** themselves.

Last but not least the following quote on the general situation in software measurement also applies to the OO metrics area [Pfleeger 97]: ‘’Researchers, many of whom are in academic environments, are motivated by publication. In many cases, highly theoretical results are never tested empirically, new metrics are defined but never used, and new theories are promulgated but never exercised and modified to fit reality. Practitioners want short-term, useful results. Their projects are in trouble now, and they are not always willing to be a testbed for studies whose results won’t be helpful until the next project.’’

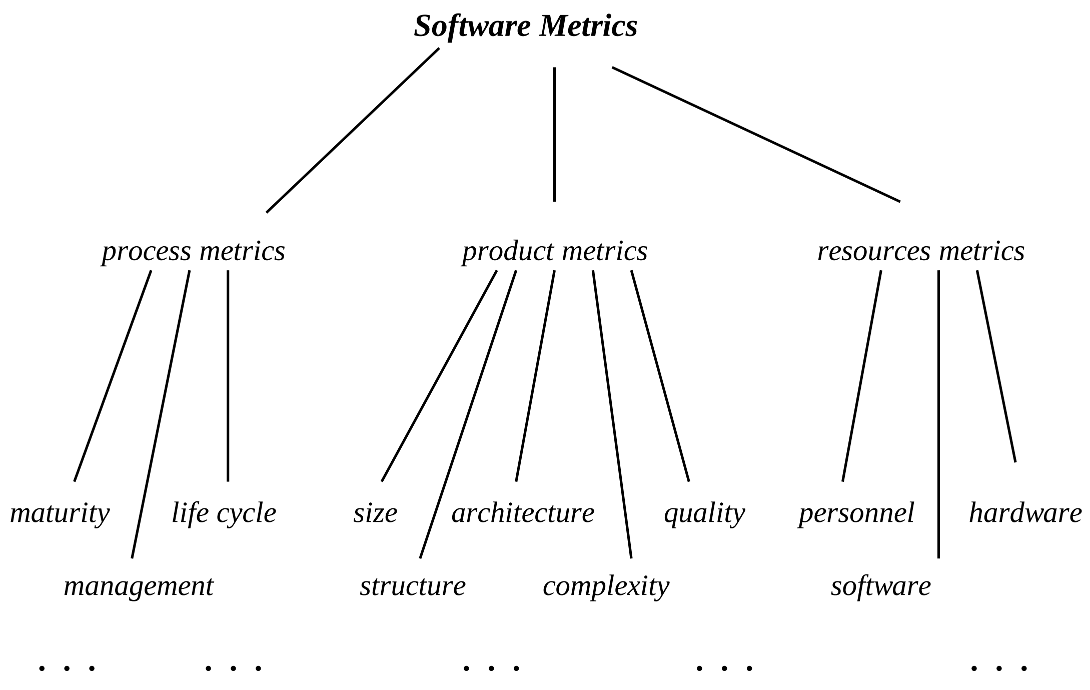
Based on this experience, we defined an object-oriented measurement framework that will be described in a short manner in the next section.

4 A General Object-Oriented Measurement and Evaluation Framework

We define a general software measurement framework with the following components (see also [DFKW 96], [DuWi 96], [DuWi97]):

4.1 Measurement Choice

This step includes the choice of the software metrics and measures from a general ***metrics class hierarchy*** (including the process, product, and resources measurement) with the following contents (derived from an analysis of the SQA literature and standards) (see also 3.2).



(see above for detailed classification)

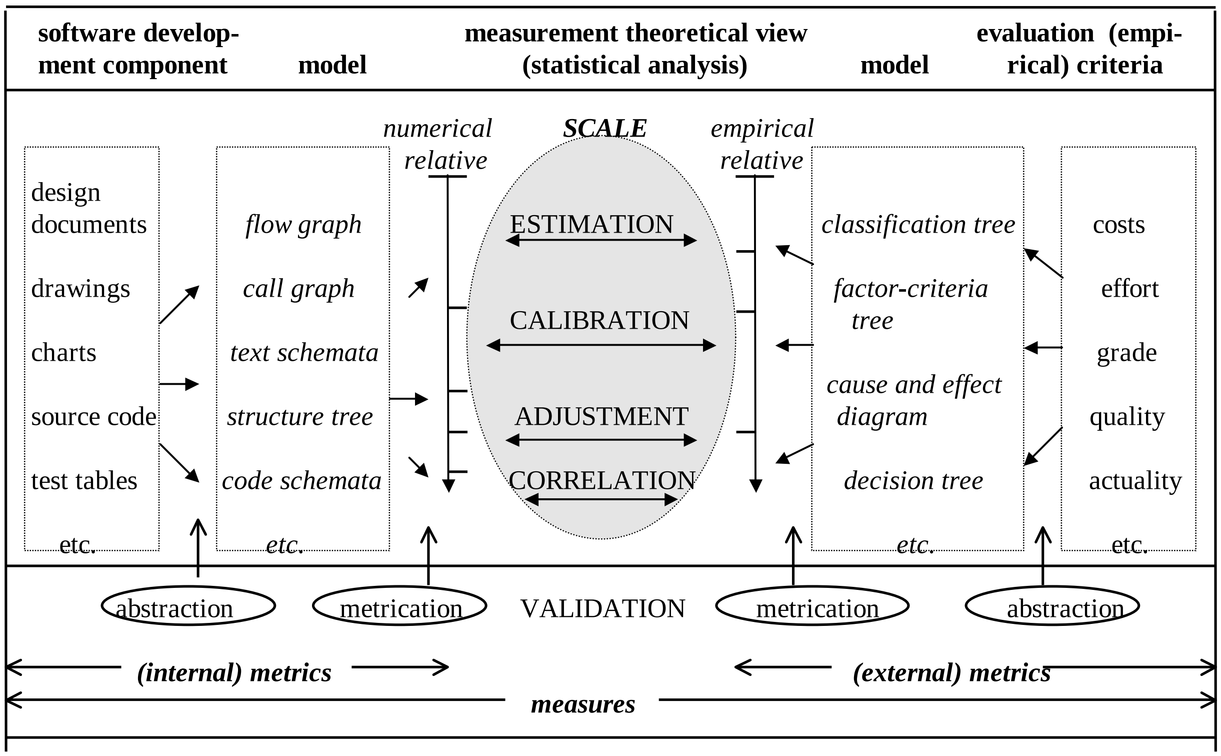
The second part in the measurement choice is the definition of an object-oriented software ***metric as a class/object*** in the Coad/Yourdon approach manner with the default contents as

- attributes: the *metrics value characteristics*, and
- services: *the metrics application algorithms.*

4.2 Measurement Adjustment

The adjustment is related to the experience (expressed in values) of the measured attributes for the evaluation. The adjustment includes the metrics validation and the determination of the metrics algorithm based on the ***measurement strategy.*** The strategy can be *model-based measurement* (e. g. metrics based on the control flow graph; service form: *count*, *execute*), *direct measurement* (such as execution time, storage size; service form: *read the (operating) system dates and/or execute*), *evaluations* (as classification of tools, or process level identification; service form: *evaluate*), and *estimations* (as formula-based execution of software characteristics; service form: *estimate*). In estimation the software measurement results are comprised in the estimation formula.

The following table gives an overview of the validation problem.



The steps of the measurement adjustment are

- the determination of the *scale type* and the *unit*,
- the determination of the *initial values* of the metrics based on prior experience or an *assessment*,

- the use of these values as *favorable values* for the evaluation of the measurement component,

The measurement adjustment in our example is realized by the Prolog metrics tool (PMT) [Kompf 96] and in the Smalltalk measure extension [Heckendorff 96] in the following way. The tool starts with an evaluation of a chosen piece of software (in Smalltalk a part of the system itself). The obtained measures are used as initial empirical evaluation criteria to define ‘acceptable’ quality. Here is a simple example to further explain the idea of measurement adjustment. An application of a Java CAME tool [Patett 97] for JAVA ‘’standard’’ libraries gives the following selected results:

- average number of methods in a JAVA class: 10,
- average lines of code of a JAVA class method: 11.4,
- average number of parameters per method: 1.3.

This values can be used as evaluation criteria (limits) for a ‘good’ Java application. One Java application of our Measurement Laboratory (a measurement data base interface [Fix 96]) can be described in a classical manner with the following values:

- total lines of JAVA code: 1320,
- JAVA classes: 25,
- average number of methods per class: 12,
- average number of parameters per method: 0.88,
- average lines of code per methods: 4.04, etc.

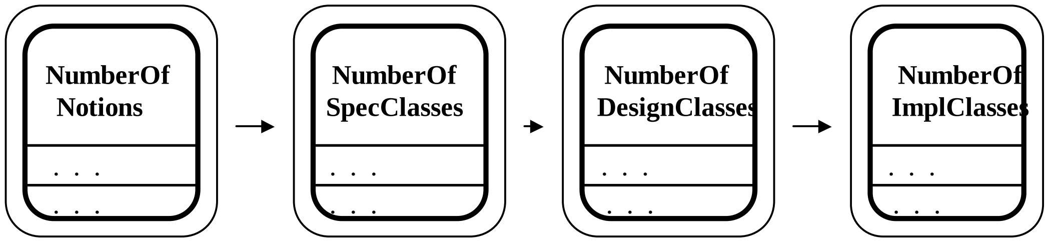
In general we see a conformity of our Java application with the evaluation criteria.

4.3 Measurement Migration

The migration includes ***refinement*** and the ***tracing*** of the metrics ‘mutations’ throughout the development phases for the given development paradigm, e. g. metrics splitting or transforming for different levels of granularity. Thus we define metrics as ‘*quality agents’* in the software development process. The activities of these agents are reasoning on the *software development complexity* [DuWi 96] that is based on the *product* or *project dependency,* the *development methodology dependency,* the *basis software dependency,* the *development team dependency,* the *company area dependency,* and the *time dependency* of the developed software components.

It is necessary to *cover* both directions in the measurement and evaluation paradigm for all components. An example that is described in [Dumke 95] is

phase: *Problem definition* *OO analysis* *OO design* *OO implementation*



It shows an ***adaptive metric class*** NumberOfClasses for the primary phases of an OO development. In the same manner ‘traces’ from adjectives and predicates to the NumberOfAttributes or from verbs and adverbs to the NumberOfServices can be defined.

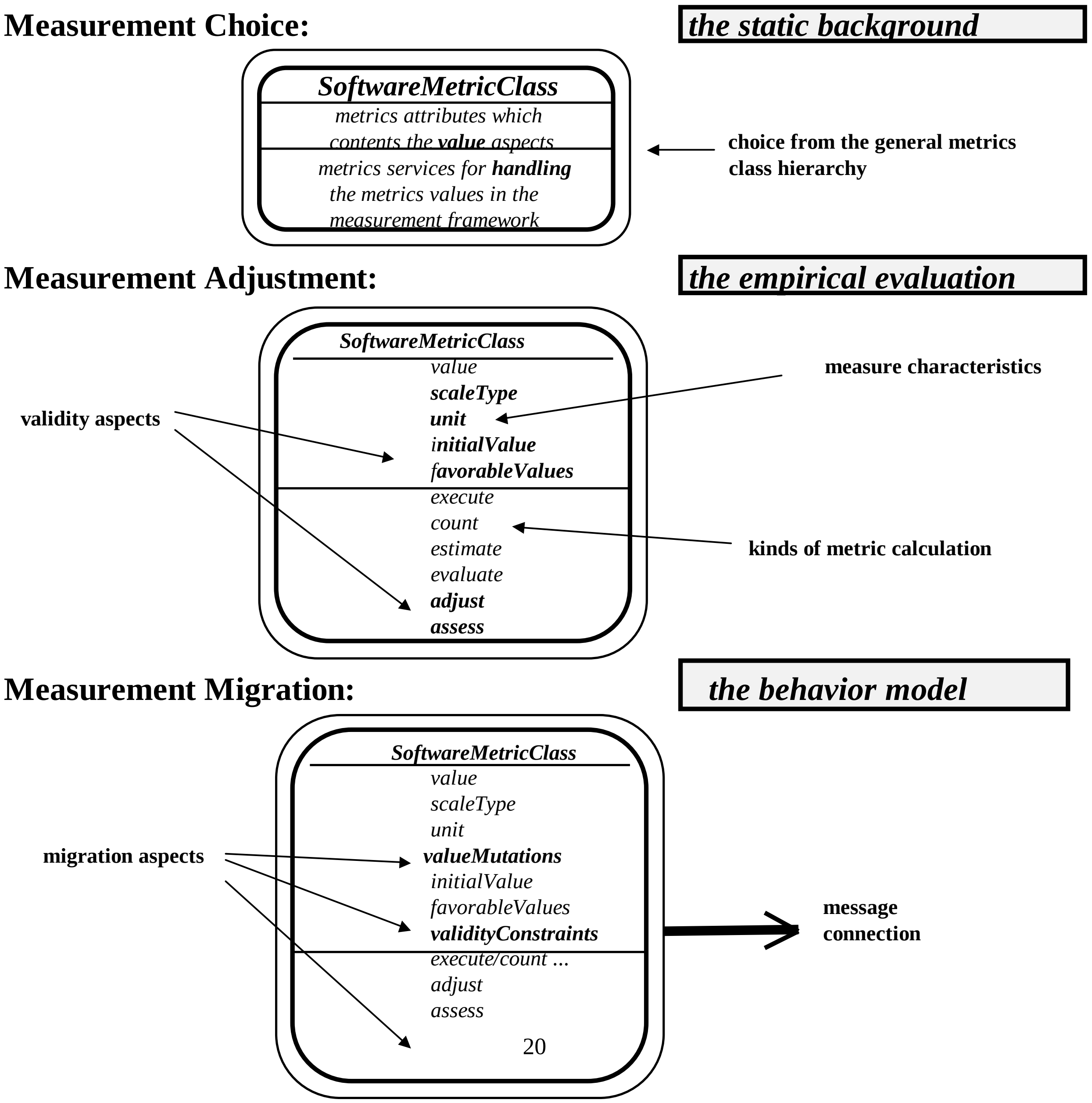
Further, it is necessary to repeat the determination of the ‘environmental’ metric values in time intervals to allow for a *tuning* of the *favorableValues* and their conditional variations as *validityConstraints* to guarantee the achievement of selected quality aspects. Note, that the migration may require a repetition of the adjustment step.

4.4 Measurement Efficiency

This step includes the ***instrumentation*** or the automatisation of the measurement process by tools. It requires to analyze the algorithmic character of the software measurement and the possibility of the integration of tool-based ‘control cycles’ in the software development process.
The acronym of our framework is *measurement choice, adjustment, migration, and efficiency (CAME)*. We use the same acronym (with another meaning) for the tools supporting our framework [Dumke 96].

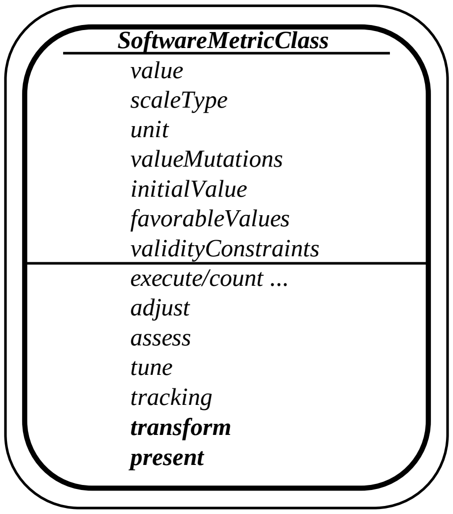
A digest of this framework is given in the next figure. It includes the extension of the metric class to include the facilities necessary to evaluate object-oriented software development

.



tune
tracking

Measurement Efficiency: ***the supporting tools***



services functionality:

- (*execute* / *count* / *estimate* / *evaluate*) the (*value* / *initialValue*)
- *adjust* the *favorableValues*
- *assess* the value relating to the *favorableValues* and the *validityConstraints* in the *scaleType* and the *unit*
- *tune* the *favorableValues* and the *validityConstraints*
- *tracking* the *valueMutations*
- *transform* the *value* (with *unit* and/or *scaleType)*
- *present* the *value* by display or indicate

5 Process Evaluation of Chosen OO Software Development Methodologies

5.1 Evaluation Foundations

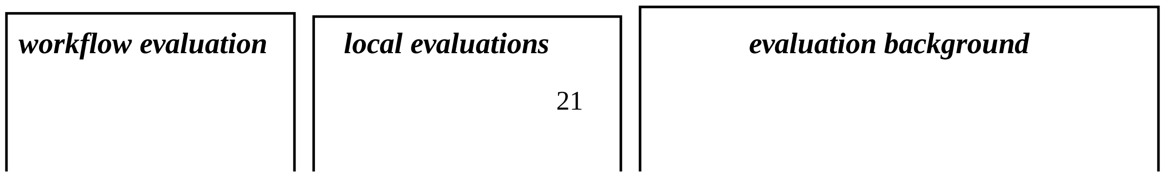
The evaluation includes the general product, process and resources measurement aspects for the OO development methods themselves as

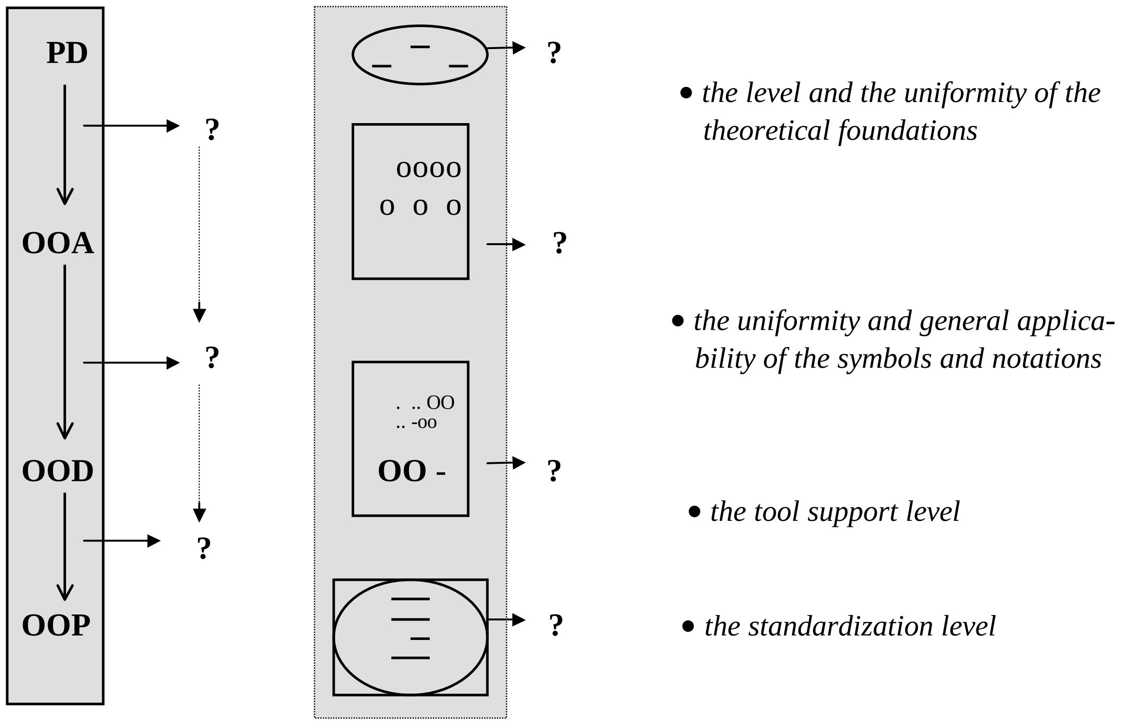
- ***OO method product evaluation:***
  - size,
  - architecture,
  - structure,
  - quality (functionality, reliability, usability, efficiency, maintainability, portability),
  - complexity;
- ***OO method process support evaluation****:*
  - maturity,
  - management (project, quality, configuration),
  - life cycle;
- ***OO method resource evaluation:***
  - personnel (team structure),
  - software (paradigm, replacement).

On the other hand we must consider the general components of an OO development methodology as (see also [Jacobson 95], [Marciniak 94], [Wasserman 88] and [Tepfenhart 97])

- *theoretical foundations,*
- *symbols and techniques,*
- *(CASE) tools,*
- *standards.*

Hence, we must consider the following main areas for a metrication of an object-oriented development methodology:

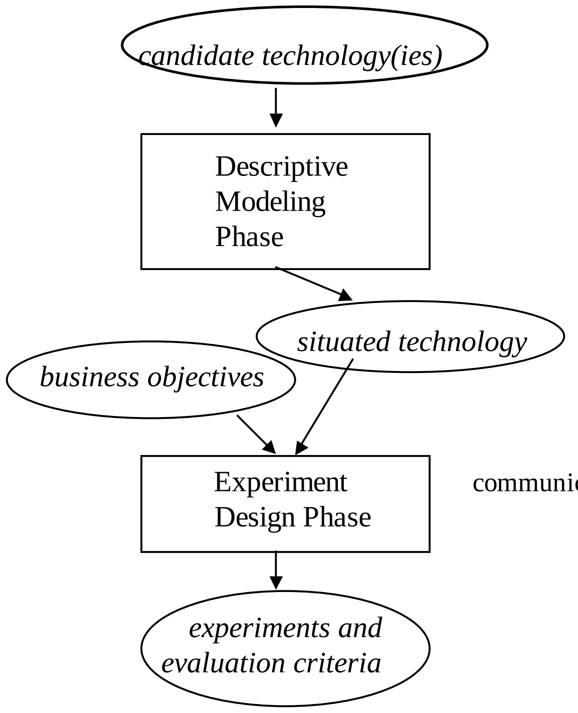



PD
OOA
OOD
OOP
?
oooo
o o o
. .. OO
.. -oo
OO -
• the level and the uniformity of the theoretical foundations
• the uniformity and general applicability of the symbols and notations
• the tool support level
• the standardization level

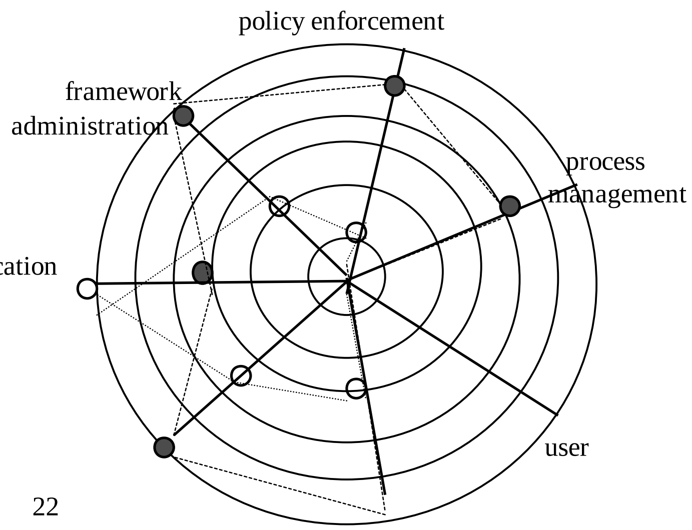
The discussion in [Shet 97] includes that ‘’activity-based methodologies focus on modeling activities instead of modeling the commitments among people’’ and that ‘’advanced workflow management systems allow mobile clients’’. First ***workflow*** measurement ideas can be found in [Ebert 93]. However, they are aimed at only one issue - the complexity.

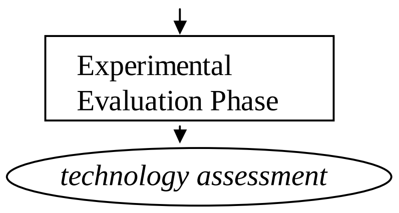
A recent description of ***local evaluations*** is given in section 3 of [Kaschek 96]. Metrics related to the text (size and readability) are also used in the specification and design phases [Kitchenham 89]. Local evaluations may be considered as the ‘’classical’’ measurement approach. A general concept is given in [Brown 96a] and [Brown 96b]. The main idea of this approach is the *technology delta principle.* The framework includes the following phases related to a given (exemplary) result:

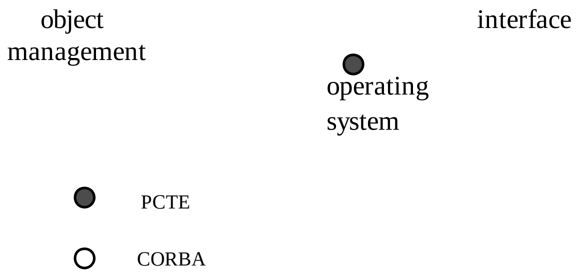
evaluation framework


candidate technology(ies)
Descriptive Modeling Phase
situated technology
business objectives
Experiment Design Phase
experiments and evaluation criteria

evaluation result example







The ***background evaluation*** should be used as indicator for the evaluation of all aspects in the software process.

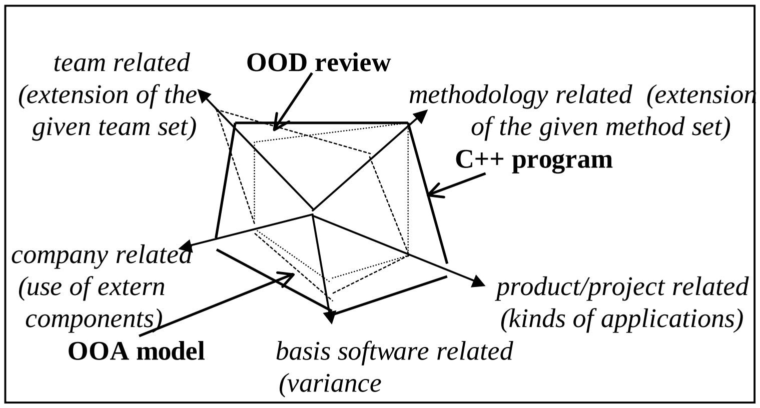
In following we will discuss the workflow evaluation based on so-called ***quality agents*** with the ingredients of the local and background evaluation aspects.

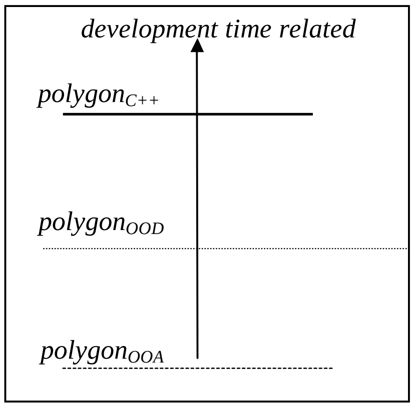
5.2. Software Quality Agents

The ***quality agent*** was based on the idea of the (*mobile) intelligent agent* in the area of distributed systems and networks. Mobile agents are computational processes which are capable of moving from node to node around a network [Appleby 94]. They may be considered as a natural extension of the object-oriented programming philosophy to include features which are tailored to distributed control.
Whereas a mobile agent helps to manage the performance of the network processes, the quality agent controls the software product or process quality in a given software development environment. The idea of the software quality agent is opposite to the total quality management (TQM, see [Marciniak 94]) which want to address the quality assurance in a wholeness manner. The TQM has practice relevance for assessment, whereas software agents are suitable for the process controlling. The quality agent has the following characteristics

- it incorporates ***quality knowledge*** as a set of metrics/measures based on the measurement choice step of our framework,
- ***decision rules*** for the action or reaction of the agent based on the empirical (initial) evaluation values of the chosen metrics (as result of the measurement adjustment step) are defined,
- it is able to ***navigate*** in the software development environment based on the measurement migration step of our framework,
- it provides ***visualization/presentation forms*** based on the measurement efficiency step.

The (product) quality aspects based on ISO 9126 [ISO9126 91] are used as a guide for empirical evaluation. The product functionality and reliability and the process maturity and life cycle aspects are controlled by the ***requirement workflow agents***. These agents include the duality of the functionality as characteristic of the implemented product and the given development method. The product maintainability and portability, the process management and the resource personnel and software aspects should be served by the ***complexity workflow agents.*** Complexity means *software development complexity* as described above. A visualization is given in the following figures which include examples of development components (OOA model, OOD review, and C++ program) with their different polygons related to several complexity aspects.


team related (extension of the given team set)
OOD review
methodology related (extension of the given method set)
C++ program
company related (use of extern components)
product/project related (kinds of applications)
OOA model
basis software related (variance



The product size, structure, architecture, usability, efficiency and complexity, the process management and the resource software performance aspects should be described by the ***component workflow agents.*** These agents observe the specification, design and implementation components defined by the used development method. In the following table we define the concrete agents contents and characteristics for the development paradigm evaluation.

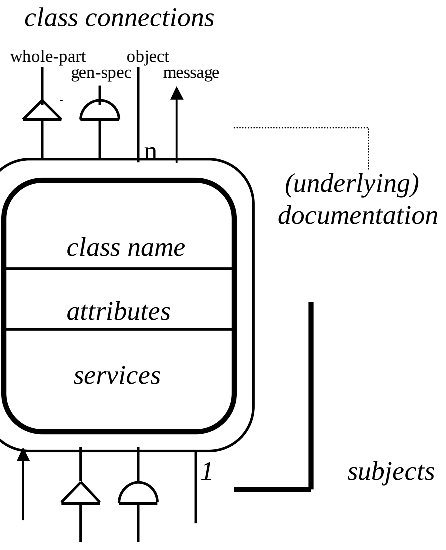
Software Agent	***Choice***	***Adjustment***	***Migration***	***Efficiency***
Requirement Workflow Agent	***kindsOfRequirements*** (Process Life Cycle, Product Functionality Metric) *kinds:*‘functional’, ‘quality’, ‘system’ (platform: hard- and software), ‘control’ (project planning)	*values:* 0, 1, ..., 4 *scaleType:* ordinal *initialValue:* 4 *favorableValues:* <3: no project, =3 (incl. ‘funct.’): incomplete, = 4: complete *service:* count of kinds	*valueMutations:*reduction along the life cycle *validityConstraints:* full functional requirements reduction in the spec. phase, system requirement reduction in the design phase	*evaluation level:* - monolithically, - differently *presentation:* four bars with colored part of the requi. reduction
	tracesOfRequirements (Product Reliability Metric) *traces:* #requirements between two related phases	*values:* [0, 4] *scaleType:* ordinal *initialValue:* 1 *favorableValues:* 4 (ideal) *service:* execute median requ. passing of the 4 types above	*valueMutations:* quotient should remain constant (=1) *validityConstraints:* a missing requirement indicates a singularity; milestones are the measurement points	*evaluation level:* - passing, - interrupting *presentation:* colored indication, of the anomalies
	storageOfRequirements (Process Maturity Metric) *storage:* #requirements in a computational form	*values:* [0, 4] *scaleType:* ordinal *initialValue:* 1 *favorableValues:* 4 (ideal) *service:* execute the median of the storage requirement kinds along the life cycle	*valueMutations:* can be changed along the life cycle *validityConstraints:* the storaged requirements obtain along the life cycle a higher topological binding to the method components	*evaluation level:* - verbal/textual, - formal/analyzable *presentation:* storage attributing of the method components
Complexity Workflow Agent	***similarityOfMethods*** (Product Portability Metric, Resource Software Replacement Metric) *methods:* SA, OO, Petri Nets, ERM, JSD etc.	*values:* ‘continuous’,‘similar’, ‘transferable’, ‘stand alone’ *scaleType:* ordinal *initialValue:* ‘stand alone’ *favorableValues:* ‘similar’ *service:* estimate the change to the new (OO) methodology	*valueMutations:* the similarity can change along the life cycle *validityConstraints:* the estimated values are depended on the given tools and techniques of the new method	*evaluation level:* - approach related, - components related *presentation:* estimation per development phase
	varianceOfPlatforms (Resource Metric) *platforms:* mainframe, PC, WS, distributed etc.	*values:*‘fixed’,‘various’,‘free’ *scaleType:* ordinal *initialValue:* ‘fixed’ *favorableValues:*‘free’ (ideal) *service:* evaluate method dep.	*valueMutations:* can be changed along the life cycle *validityConstraints:* the value ‘fixed’ is also ideal if it is given before	*evaluation level:* -computer related, -architecture related *presentation:* appropriate
	kindsOfApplications (Product Architecture Metric) *application:* IS, Real-time etc.	*values:* ‘defined’, ‘free’ *scaleType:* ordinal *initialValue:*’free’ *favorableValues:* ‘free’ *service:*evaluate method dep.	*valueMutations:* can be changed along the life cycle *validityConstraints:*‘defined’ can also be favorable in the given environment	*evaluation level:* - paradigm related, - resource related *presentation:* appropriate
	changingOfTeams (Resource Personnel Metric) *teams:* spec., test, quality etc.	*values:* ‘splitting’,’indifferently’, ‘reducing’ *scaleType:* ordinal *initialValue:* ‘indifferently’ *favorableValues:* ‘reducing’ *service:* estimate	*valueMutations:* can be changed along the life cycle *validityConstraints:* the final value is the maximum of the estimation during the life cycle	*evaluation level:* - temporary group, - permanent group *presentation:* appropriate

	differingOfComponents (Process Management Metric) *components:* (trademarked) tools, (involved) standards etc.	*values:* 0,1,2,...,k *scaleType:* ordinal *initialValue:* 0 *favorableValues:* 0 *service:*evaluate method dependent	*valueMutations:* can be changed along the life cycle *validityConstraints:* the final value results from cumulative phases related values	*evaluation level:* - intern implemented or planned, - extern (impl./pl.) *presentation:* appropriate
Component Workflow Agent	***numberOfComponents*** (Product Structure, Usability, Efficiency Metric) *components:* doc's, charts, code, library, repository etc.	*values:* 0,1,2,...,n *scaleType:* ordinal *initialValue:* *m* (from the original method description) *favorableValues: m* *service:* count of components		
	numberOfCharts (Product Architecture, Complexity Metric) *charts:* ERM, Petri Nets, State Trans., DFD etc.	*values:* 0,1,2,...,n *scaleType:* ordinal *initialValue:* *m* (see above) *favorableValues: m* *service:* count of charts	*valueMutations:* may be changed from one development phase to another	*evaluation level:* - opposite components, - similar components
	numberOfSymbols (Resource Software Metric) *symbols:* class/object icons, structural icons etc.	*values:* 0,1,2,...,n *scaleType:* ordinal *initialValue:* *m* (from the original method description) *favorableValues: m* *service:* count of symbols	*validityConstraints:* some of the counting components require a continuity along the development phases	*presentation:* distance presentation depending on the similarity during the life cycle
	numberOfRules (Process Management Metric) *rules:* statements for the definition of the components	*values:* 0,1,2,...,n *scaleType:* ordinal *initialValue:* *m* (see above) *favorableValues: m* *service:* count of rules or development principles		

5.3 Methodology Related Evaluations

As a first application we used these agents to assess OO development methods. We have chosen seven well-known OO development methods. The assessment includes a typical class icon from each method to give a small impression of the features. Then we present the metrics values of the particular method. The first assessed method is the Coad/Yourdon approach **OOA** [Coad 93] with the development steps OOA,OOD, and OOP.

class icon



phases: OOA, OOD, OOP
steps per phase: 5 OOA, 4 (human interface, task, data, problem domain component) OOD, code frame generation
service description: verbal, state transition diagram

quantitative method characteristics

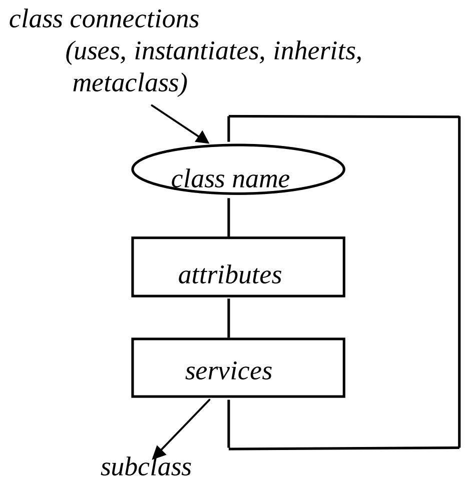
Requirement workflow:
- *kindsOfRequirements:* 2 ('functional', 'system'; monolithically)
- *tracesOfRequirements:* PD→OOA: 0, OOA→ OOD: 2, OOD→OOP: 1; median: 1
- *storageOfRequirements:* median: 1 (textual)

Complexity workflow:
- *similarityOfMethods:* 'stand alone'
- *varianceOfPlatforms:* 'various' (PC, Unix-WS)
- *kindsOfApplications:* 'free'
- *changingOfTeams:* 'indifferently'
- *differingOfComponents:* 2 (OS,OOP language)

Component workflow:
- *numberOfComponents:* 5 (doc, drawing(s), tem-plates, critiques, code frames)
- *numberOfCharts:* 2 (classes, state transition dia- gramm)
- *numberOfSymbols:* 7 (3 boxes, 4 connections)
- *numberOfRules:* 67 (principles)

The next one is the OOD method of Booch [Booch 91] with the following characteristics.

class icon



diagrams: object (symbols for main program, specification, subprogram, package, task and generic forms), state transition, system process, system block, timing and module

quantitative method characteristics

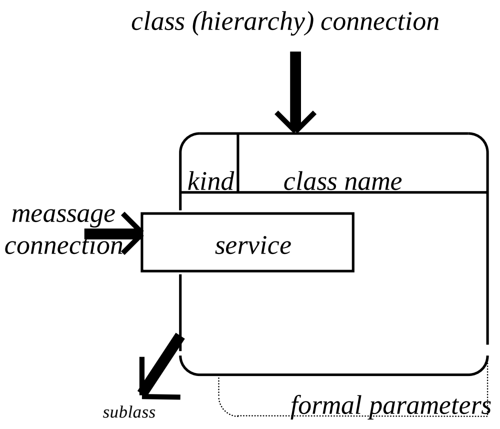
Requirement workflow:
- *kindsOfRequirements:* 2 ('functional', 'system'; monolithically)
- *tracesOfRequirements:* PD→OOA: 0, OOA→OOD: 2, OOD→OOP: 1; median: 1
- *storageOfRequirements:* median: 1 (textual)

Complexity workflow:
- *similarityOfMethods:* 'similar' to modul concept
- *varianceOfPlatforms:* 'various'
- *kindsOfApplications:* 'free'
- *changingOfTeams:* 'indifferently'
- *differingOfComponents:* 2 (OS, OOP language)

Component workflow:
- *numberOfComponents:* 3 (doc.,chart(s), code)
- *numberOfCharts:* 6
- *numberOfSymbols:* 30 (13 boxes, 17 connections)
- *numberOfRules:* 4 (general activity descriptions)

The approach from Robinson et al [Robinson 92] is defined as hierarchical object-oriented design (**HOOD**). An assessment of this method is given in following.

class icon



class diagram as: class hierarchy (HDT), class intern structure and class refinement

kernel: program design language (PDL)

software requirement document (SRD) for functional consistency (relational table: requirement to object)

quantitative method characteristics

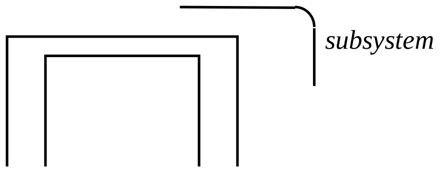
Requirement workflow:
- *kindsOfRequirements:* 2 ('functional', 'system'; monolithically)
- *tracesOfRequirements:* PD→OOA: 0, OOA→OOD: 2, OOD→OOP: 2; median: 1.3
- *storageOfRequirements:* median: 1.3 (SRD, analyzable)

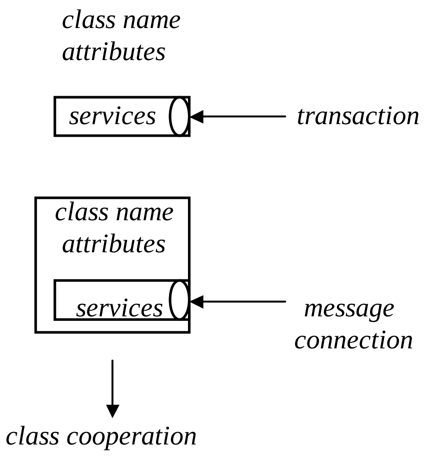
Complexity workflow:
- *similarityOfMethods:* 'stand alone'
- *varianceOfPlatforms:* 'fixed' (Ada related)
- *kindsOfApplications:* 'free'
- *changingOfTeams:* 'indifferently'
- *differingOfComponents:* 2 (OS, Ada)

Component workflow:
- *numberOfComponents:* 6 (SRD, doc., class diagram(s), design tree, PDL codes, Ada code)
- *numberOfCharts:* 2(object diagram, design tree)
- *numberOfSymbols:* 6 (1 structured Box, 5 connections)
- *numberOfRules:* 21 (9 general and 12 special principles) and 54 keywords of a PDL

For the approach of Wirfs-Brock et al [Wirfs-Brock 90] - defined as responsibility-driven design (**RDD**) - we obtain the following assessment.

class icon





diagrams: class hierarchy (with the class relations: is-kind-of, is-analogous-to, is-part-of), class co-operation (with: is-part-of, has-knowledge-of, depends-upon), Venn diagram for the responsibilities

quality rules for the design: suitable number of classes, subsystems and responsibilities

Requirement workflow:

- *kindsOfRequirements:* 3 (‘functional’, ‘system’, ‘quality’; differently)
- *tracesOfRequirements:* PD→OOA: 0, OOA→OOD: 3, OOD→OOP: 0; median: 1
- *storageOfRequirements:* median: 1 (textual)

Complexity workflow:

- *similarityOfMethods:* ‘transferable’
- *varianceOfPlatforms:* ‘free’
- *kindsOfApplications:* ‘free’
- *changingOfTeams:* ‘indifferently’
- *differingOfComponents:* 3 (OS, OOP language, Venn diagram)

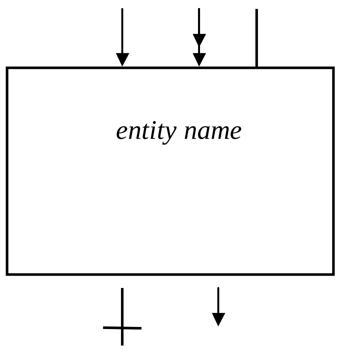
Component workflow:

- *numberOfComponents:* 3 (doc., chart(s), code)
- *numberOfCharts:* 3 (hierarchy, class, Venn)
- *numberOfSymbols:* 11 (6 boxes, 5 connections)
- *numberOfRules:* 26

quantitative method characteristics

The Shlaer/Mellor approach ([Shlaer 96] **OOSA)** is based on the idea of an object as an entity used in the ERM paradigm.

class icon



diagrams: data flow diagram (DFD), entity relation-ship diagram (with the typical types of relations) and an additional class hierarchy diagram

no restrictions for OO

quantitative method characteristics

Requirement workflow:

- *kindsOfRequirements:* 2 (‘functional’,’system’; monolithically)
- *tracesOfRequirements:* PD→OOA: 2, OOA→OOD: 2, OOD→OOP: 0: median: 1.3
- *storageOfRequirements:* median: 1 (textual)

Complexity workflow:

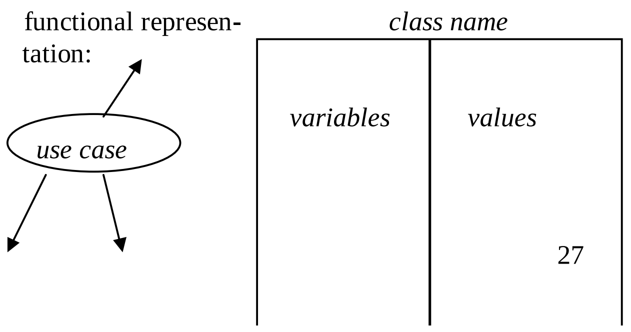
- *similarityOfMethods:* ‘continuous’
- *varianceOfPlatforms:* ‘various’
- *kindsOfApplications:* ‘defined’ (data base)
- *changingOfTeams:* ‘splitting’
- *differingOfComponents:* 3 (OS, programming language, SA technique)

Component workflow:

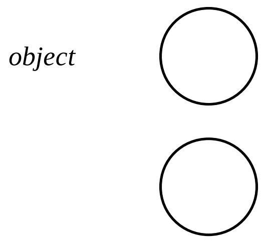
- *numberOfComponents:* 3 (doc.,diagram(s),code)
- *numberOfCharts:* 3 (hierarchy, ER, DFD)
- *numberOfSymbols:* 13 (2 boxes, 11 connections)
- *numberOfRules:* 28

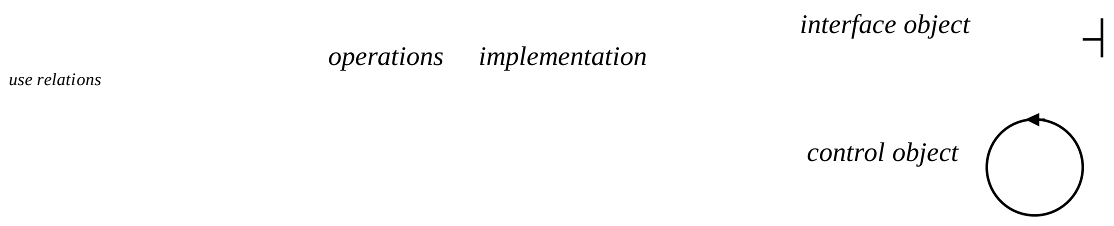
The Jacobson approach **OOSE** [Jacobson 92] defines several types of simple classes. The assessment of this method is given in following.

class icon



symbols for the object diagram:





kinds of models: requirements, analysis, design, implementation, test

diagrams: use cases, object, interaction, design, state transition diagram

quantitative method characteristics

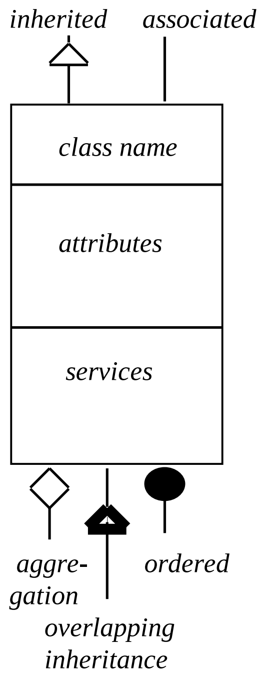
Requirement workflow:
- *kindsOfRequirements:* 3 (as use cases, without ‘control’; differently)
- *tracesOfRequirements:* PD→OOA: 3, OOA→OOD: 3, OOD→OOP: 3; median: 3
- *storageOfRequirements:* median: 3 (textual)

Complexity workflow:
- *similarityOfMethods:* ‘transferable’
- *varianceOfPlatforms:* ‘various’
- *kindsOfApplications:* ‘free’
- *changingOfTeams:* ‘indifferently’
- *differingOfComponents:* 3 (OS, OOP language, state transition diagram (SDL))

Component workflow:
- *numberOfComponents:* 5 (models)
- *numberOfCharts:* 5 (diagrams)
- *numberOfSymbols:*26 (18 boxes, 1 symbol, 7 connections)
- *numberOfRules:* implicite description

Last but not least, the representation used in the **OMT** approach by Rumbaugh et al [Rumbaugh 91] is similar to the representation of the Coad/Yourdon approach. The method assessment is given in following.

class icon



diagrams: class diagram (including the ERM facilities), state transition diagram, data flow diagram

quantitative method characteristics

Requirement workflow:
- *kindsOfRequirements:* 2 (‘functional’,’system’; monolithically)
- *tracesOfRequirements:* PD→OOA: 2, OOA→OOD: 2, OOD→OOP: 2; median: 2
- *storageOfRequirements:* median: 2 (textual)

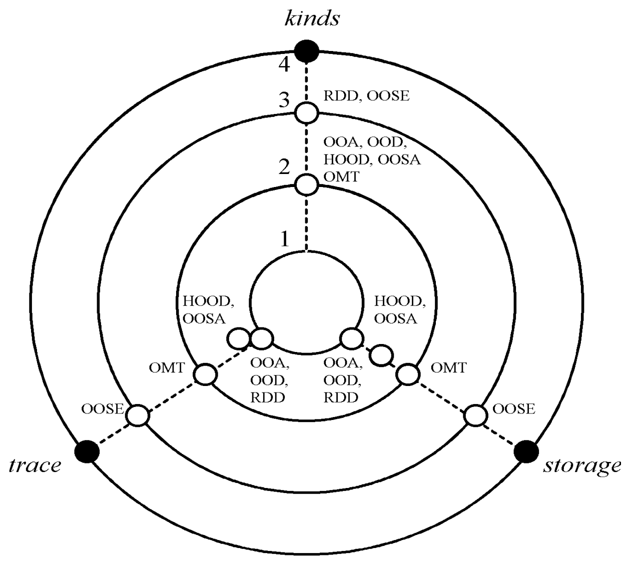
Complexity workflow:
- *similarityOfMethods:* ‘similar’
- *varianceOfPlatforms:* ‘various’
- *kindsOfApplications:* ‘free’
- *changingOfTeams:* ‘indifferently’
- *differingOfComponents:* 3 (OS, OOP language, SA methodology)

Component workflow:
- *numberOfComponents:* 3 (doc, model(s), code)
- *numberOfCharts:* 3 (object, dynamic, functional)
- *numberOfSymbols:* 19 (8 boxes, 11 connections)
- *numberOfRules:* 59

Of course, the evaluation is subject to refinement and therefore open for discussion. The following charts provide a summarization of these evaluations to compare the chosen OO development methods.

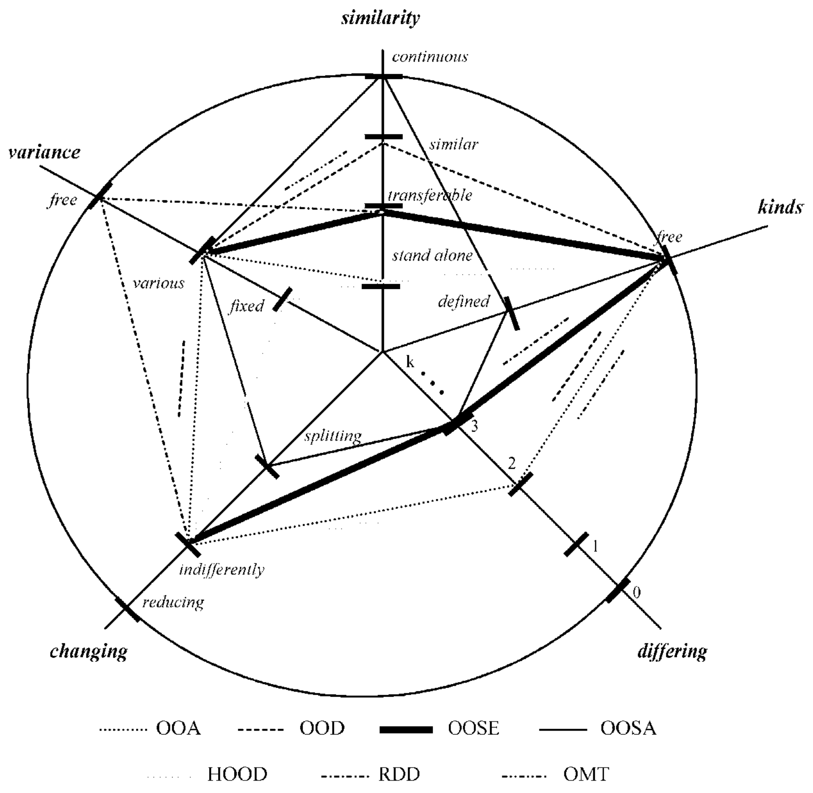
Note, that this evaluation is only an ***assessment,*** useful as start point of the use of software quality agents. The '•' marked points denote the 'ideal' values of the given aspects.

Requirement workflow

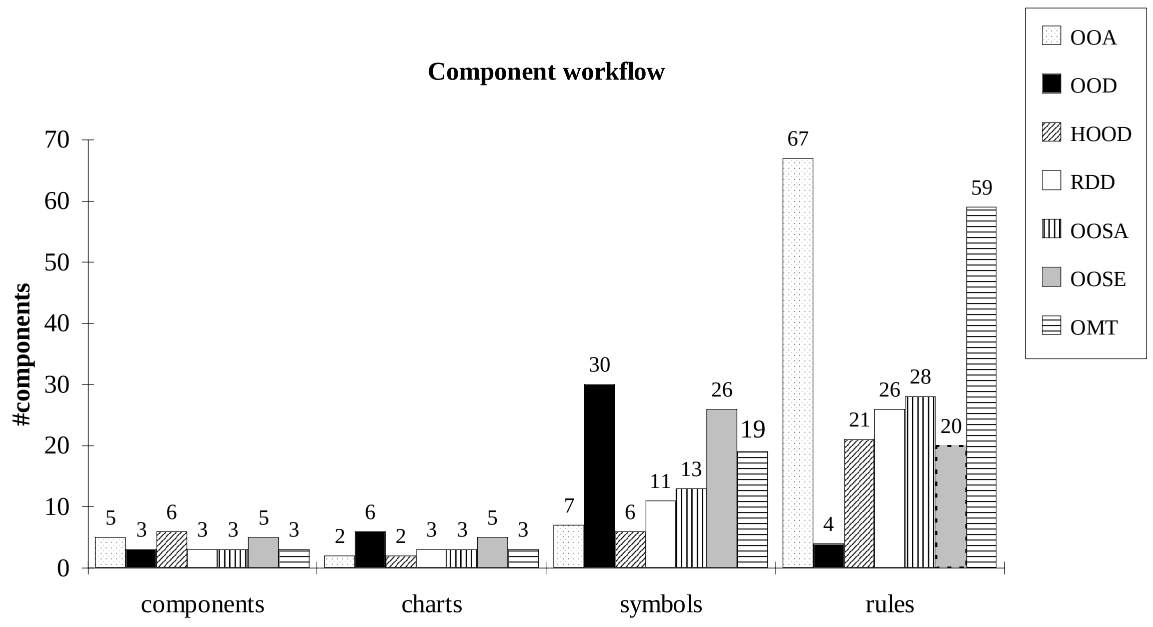


The outer circle in the following chart describes the method related 'ideal' values of the software development complexity aspect.

Complexity workflow



The quantitative evaluations of the method components are put together in the next chart.



The empirical evaluation of the component workflow values depends on the (psychological) experience in the software development in general (usually presented in simple rules like: a maximum number of *three* levels or parts, not more than *seven* elements etc.).

5.4 Evaluation of Further OO Techniques

The first evaluated OO technique are the ***Design Patterns*** [Gamma 95]. The essential objective of this technique is to improve the software design and implementation by formalizing the experience of OO applications in the abstract notion of *patterns*. The improvement aspects are

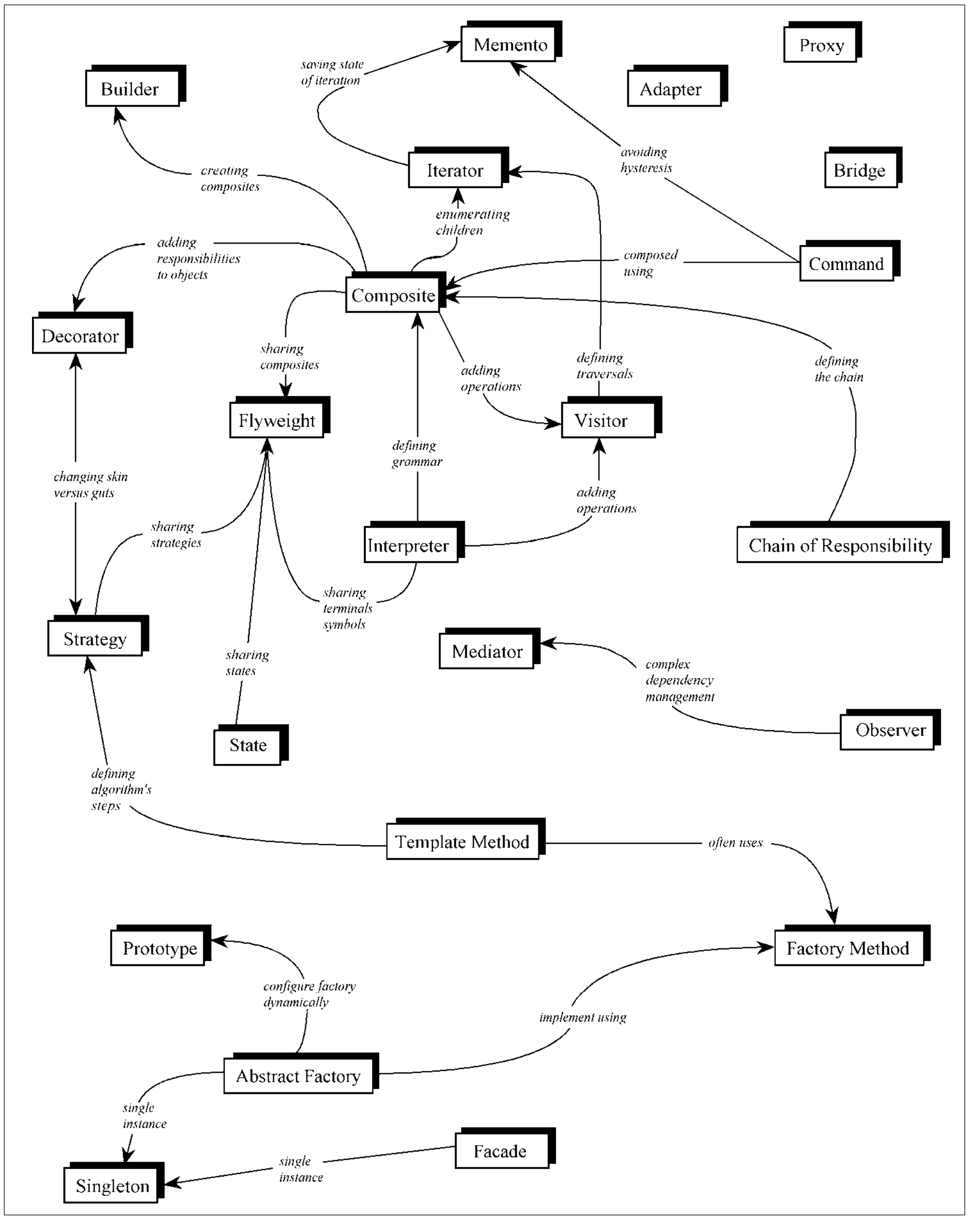
- reducing of product architecture components (by means of standardization),
- increasing the process efficiency in the life cycle,
- using experience for a better process maturity,
- decreasing the structural complexity in the software design,
- increasing of the resource personnel productivity in general.

The following table describes the defined patterns with their design aspects and their characteristics that can vary (in parentheses).

Scope	Creational Purpose	Structural Purpose	Behavioral Purpose
Class	**Factory Method** (subclass of object that is instantiated)	**Adapter (class)** (interface to an object)	**Interpreter** (grammar and interpretation of a language)
			Template Method (steps of an algorithm)
Object	**Abstract Factory** (families of product objects)	**Adapter (object)** (interface to an object)	**Chain of Responsibility** (object that can fulfill a request)
	Builder (how a composite object gets created)	**Bridge** (implementation of an object)	**Command** (when and how a request is fulfilled)
	Prototype (class of object	**Composite** (structure and	**Iterator** (how an aggregate's

	that is instantiated)	composition of an object)	elements are accessed, traversed)
	Singleton (the sole instance of a class)	**Decorator** (responsibilities of an object without sub-classing)	**Mediator** (how and which objects interact with each other)
		Facade (interface to a sub-system)	**Memento** (what private information is stored outside an object, and when)
		Flyweight (storage costs of objects)	**Observer** (number of objects that depend on another object; how the dependent objects stay up to date)
		Proxy (how an object is accessed; its location)	**State** (states of an object)
			Strategy (an algorithm)
			Visitor (operations that can be applied to object(s) with-out changing their class(es))

On the other hand, these patterns are related among themselves in their application in an OO software system. The following chart gives an overview of these relationships.



The application of our method evaluation is described in a short form in the following

- design patterns are a typical approach of *solution by example*,
- the application of design patterns follows the TQM idea in a constructive manner (in order to reduce the analysis/evaluation effort, to keep quality),
- the influence of this approach to our software agents are the followings
  - ∗ the *kindsOfRequirements* are extended by the implicit keeping of special quality aspects,
  - ∗ the design pattern method is similar to the OMT (*similarityOfMethods),*
  - ∗ the *numberOfRules* are reduced by an dominant use of these patterns.

The design patterns are mainly an architecture related approach supporting software development.

The second (not only OO related) approach is the ***Component-Based Software Engineering (CBSE)*** [Brown 96]. The basic idea is the practice of *composing* software by combining self developed parts with so-called *components of-the-shelf (COTS)* with the permanent underlying question 'make or buy' of software components. The CBSE is not really an OO approach, but it involves the general idea of an (instantiated) object. The general characteristics of the CBSE are that [Brown 96, p. 8] the components

- ''are ready 'off-the-shelf', whether from a commercial source (COTS) or re-used from another system;
- have significant aggregate functionality and complexity;
- are self-contained and possible execute independently;
- will be used 'as is' rather than modified;
- must be integrated with other components to achieve required system functionality.''

CBSE defines five types of components (with an increasing level of visibility). The following table explains these types of components together with characteristics of related metrics [DuWi 97].

state of components	**characteristics for metrication**
off-the-shelf components (COTS)	unknown/undefined interface; includes the general problem of the estimation of the characteristics of commercial software
qualified components (interface defined)	interface metrics; information hiding aspects
adapted components (known interface; flexible adaptation (e. g. with mediator, translator etc.))	metrics for standardization of classes; metrics for interoperability; simple kinds of architecture metrics
assembled components (possibility of integration in a given architecture)	'full' use of architecture metrics; quantifi-cation of the general infrastructure (opera-ting system, data base system etc.)
updated components (adaptation to given infrastructure)	metrication of the infrastructure (architec-ture, platforms, methods, enterprise goals, 'peopleware', environments etc.)

In relation to our software agents we can establish the following influences and evaluation aspects

- the use of components keep the application of all *kindsOfRequirements* for a chosen functionality, but provide no insight into quality and maintenance (as control aspect of the requirements),

- the *tracesOfRequirements* and the *storagesOfRequirement* in the CBSE include uncertain evaluation partitions,

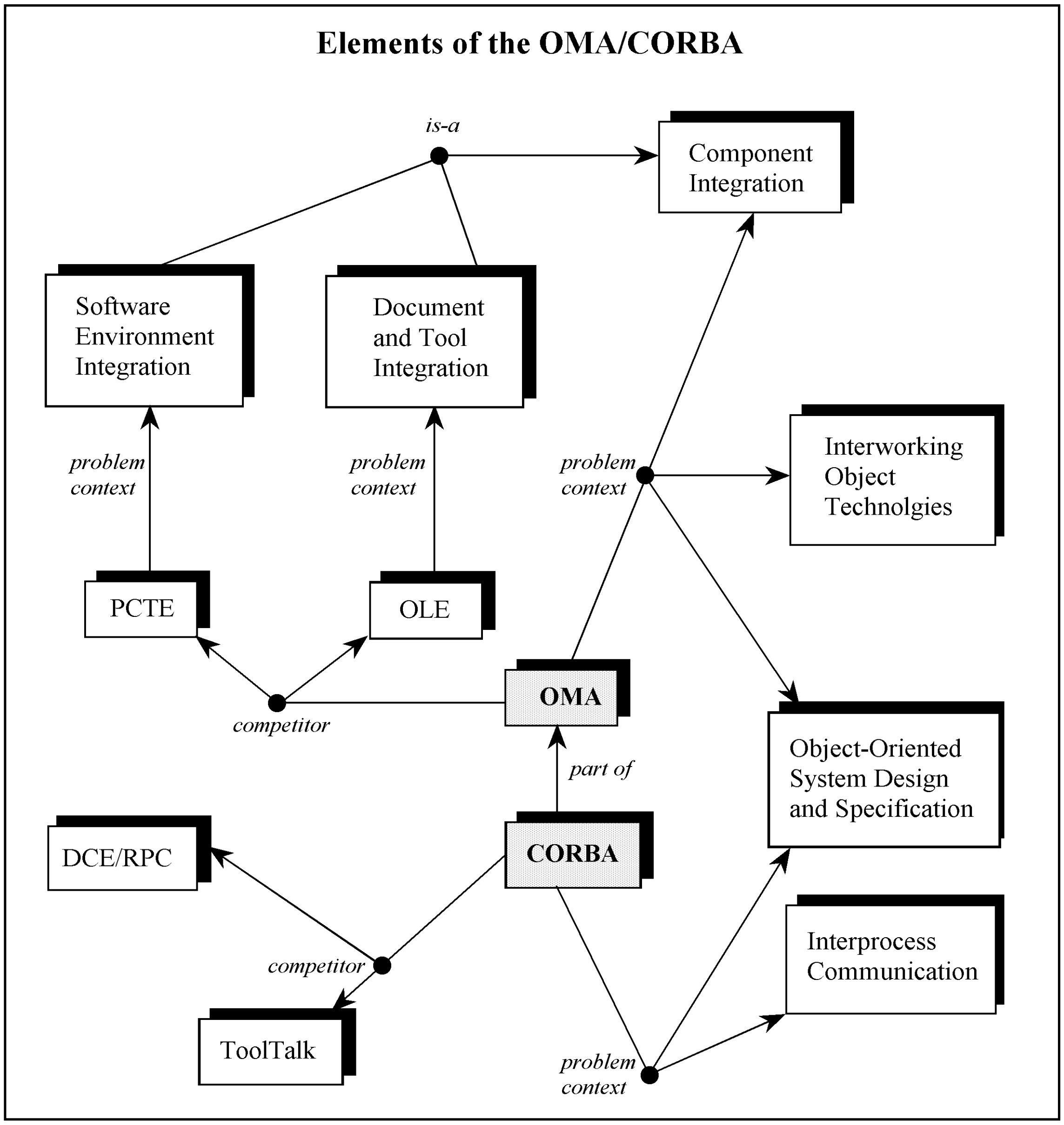
- the *similarityOfMethods* depends on the kind of the component design (see the variants of components in the table above),

- the *differingOfComponents* is the most significant effect in the CBSE and a special form of increasing the software development complexity,

- besides this, the CBSE does not produce a considerably different evaluation.

The CBSE is a typical software architecture related approach. The objective is to clarify the benefits and the risks of the use of existing software products.

The third approach is the ***Common Object Request Broker (CORBA)*** [OMG 95] from the Object Management Group (OMG). This approach supports the implementation of distributed systems and is a kind of so-called *Middleware*. The general overview about the CORBA elements is shown in the following chart of Brown [Brown 96a].


Elements of the OMA/CORBA
is-a
Component Integration
Software Environment Integration
Document and Tool Integration
problem context
problem context
problem context
Interworking Object Technolgies
PCTE
OLE
OMA
competitor
part of
Object-Oriented System Design and Specification
DCE/RPC
CORBA
competitor
Interprocess Communication
ToolTalk
problem context

The acronyms are: PCTE (Portable Common Tool Environment; an object management mechanism), OLE (Microsoft's Object Linking and Embedding), OMA (Object Management Architecture), DCE (Distributed Computing Environment of the Open Systems Foundation Group (OSF)), RPC (Sun's Remote Procedure Call), and ToolTalk (a communication mechanism). The main component OMA includes

- the *Applications Objects*: these object are specific and not subject of standardization by the OMG,
- the *Common Facilities*: these facilities are objects that provide useful but less widely-used functionality, e. g. electronic mail, naming service, copy and delete of objects etc.,

- the *Common Object Services* (COS): these services are widely applicable services, e. g., transactions, event management, general supports, printer service, security and safety service, and persistence and

- the *Object Request Broker (ORB)* for communication between the components above.

The communication between these components is realized with the middleware CORBA among the Object Request Broker that is responsible for all the mechanisms required to find the object implementation for a (client) request. Supports of the ORB are

- the *Interface Definition Language (IDL)* for the definition of the server operations that generate the so-called IDL-stub (including access routines), the interface repository (provides persistent objects in a form available at runtime), the IDL skeleton (including language mapping) and the implementation repository (contains information that allows the ORB to locate and activate implementations of objects),

- the *inter-ORB protocols* for the interoperability (including the Internet and general gateways),

- the *language mapping facilities* (especially for supporting C, C++, and Smalltalk),

- the integration facilities as *Basic Object Adapter (BOA)* for object embedding and the *Object Database Adapter (ODA)* for data base embedding.

According to our methodology evaluation, we can establish the following effects of the CORBA approach:

- the general evaluation is similar to the CBSE (see above), because CORBA can be considered as a special kind of component-based development (chosen functionality as *kindsOfRequirements;* some uncertainties in relation to the *tracesOfRequirements* and *storagesOfRequirements;* the *similarityOfMethods* is given by a language-oriented interface definition form (IDL) to the general PDL paradigms),

- on the other hand, we can establish a similarity to the design patterns as standardization of (here distributed) system functionalities and we can assume a continuity of some implemented qualities,

- the *kindsOfApplications* are reduced, but we can see an increasing of the *differingOfComponents*,

- the *numberOfComponents* are increased, because CORBA is a middleware that requires an additional methodology for software production.

Note, that CORBA is also an architecture related approach to implement distributed and heterogeneous systems.

The fourth considered approach is the ***Unified Modeling Language (UML)*** [UML 97] [UML 97a]. The development of UML began in October 1994 and is an unification of the Booch's OOD, the OMT, and the Jacobson's OOSE method. The method goals are

- to model systems (and not just software) using object-oriented concepts,
- to establish an explicit coupling to conceptual as well as executable artifacts,
- to address the issues of scale inherent in complex, mission-critical systems,
- to create a modeling language usable by both humans and machines.

The UML defines eight types of diagrams: the *use case diagram,* the *class diagram,* the behavior diagrams *(state diagram, activity diagram, sequence diagram,* and *collaboration diagram),* the implementation diagrams *(component diagram* and *deployment diagram).*

separates File!

UML is a visual modeling language not a programming language and is based on the diagrams above and a semantic definition [UML 97a]. For special constraints in UML can be used an *Object Constraint Language (OCL)* specification form.

The UML methodology is a good example of an evaluation process in the three steps as (a) the separate evaluation of the three source methods, (b) a methods evaluation summary, and (c) a (separate) UML evaluation. The evaluation of the UML is given in the following

Requirement workflow:

- *kindsOfRequirements:* 3 ('functional', 'system', 'quality'; differently)
- *tracesOfRequirements:* PD→OOA: 3, OOA→ OOD: 3, OOD→OOP: 3; median: 3
- *storageOfRequirements:* median: 3 (textual)

Complexity workflow:

- *similarityOfMethods:* 'similar'
- *varianceOfPlatforms:* 'various'
- *kindsOfApplications:* 'free'
- *changingOfTeams:* 'indifferently'
- *differingOfComponents:* 4 (OS,OOP language, two other methods)

Component workflow:

- *numberOfComponents:* 4 (models, diagrams, language, code frames)
- *numberOfCharts:* 8
- *numberOfSymbols:* 35 (18 boxes, 17 connections)
- *numberOfRules:* implicit principles

The following table shows a simplified overview of these evaluations.

metric	**OOD**	**OOSE**	**OMT**	**∅ (min)**	**∅ (max)**	**UML**
		Requirement	*workflow*			
kindsOfRequ.	2	3	2	2	3	3
tracesOfRequ.	1	3	2	1	3	3
storagesOfRequ.	1	3	2	1	3	3
		Complexity	*workflow*			
similarityOfMeth.	similar	transferable	similar	transferable	similar	similar
varianceOfPlatf.	various	various	various	various	various	various
kindsOfApplic.	free	free	free	free	free	free
changingOfTeams	indifferently	indifferently	indifferently	indiff.	Indiff.	indiff.
differingOfComp.	2	3	3	3	2	4
		Component	*workflow*	∅ (no min,	no max)	
numberOfComp.	3	5	3	4		4
numberOfCharts	6	5	3	4		8
numberOfSymbols	30	26	19	25		35
numberOfRules	4	ca. 20	59	28		implicit

Note, that the average of 'min' and 'max' is related to the 'weakest' and 'best' in the ordinal manner. On the other hand, there is only few experience with the UML in practice.

6 Conclusions

Every company must perform the decision about the use of new software development methods. However, we can establish the following situation about software development methodologies:

1. the description of a new development method of a ***method/tool distributor*** includes all (possible) benefits of this method and starts in general with a lack of tool supporting, no support for paradigm changing, and with a lot of 'motivation' for a maximal spread in the marketing;

2. the description of a development method in the ***literature*** according to the comparison of different (OO) methods usually includes a comparison of the features and does not address maintenance, porting, and quality issues.

Our paper includes a first analysis of the following software process evaluation aspects and characteristics:

- the aspects and approaches of software measurement in general,

- the short description of the current situation in the object-oriented software metrics research area,

- the definition of a software measurement framework that is opposite to the general TQM approach and is based on the idea of intelligent/mobile agents in computer networks,

- the first application of this framework to evaluate OO software development methods, especially with respect to the requirements, the so-called software development complexity, and the counting of the methods symbols, charts etc.

In this manner we can define in a first approximation the 'ideal' development method with the following characteristics

- a consideration of all requirements (especially the ability to store and trace);

- a low software development complexity with a similarity of the method (e. g. with migration supports from the old method to the new one), with a minimum of platform changing (e. g. with support for the portability), with no restrictions to the application area, with clear statements to the necessary team set and structure, and with a clear description of the external components required;

- a counting of the different components of a method for a characterization of their usability (the empirical evaluations are still necessary).

In our evaluation process, we have also seen one typical effect in the software measurement: *the realization of the measurement starts with the definition of the measured components and leads to a clear understanding of the considered area that should be a necessary premises.*

Further investigations are directed on the implementation of really *workflow* agents in a Java- oriented software development environment.

7 References

[Abreu 94] Abreu, F.B.; Carapuca, R.: *Candidate Metrics for Object-Oriented Software within a Taxonomy Framework*. Journal of Systems and Software, 26(1994), pp. 87-96

[Abreu 95] Abreu, F. B.; Goulao, M; Esteves, R.: *Toward the Design Quality Evaluation of Object-Oriented Software Systems*. Proc. of the Fifth International Conference on Software Quality, Austin, October 23-25, 1995, pp. 44-57

[Abreu 96] Abreu, F. B.; Melo, W.: *Evaluating the Impact of Object-Oriented Design on Software Quality*. Proc. of the Third International Software Metrics Symposium, March 25-26, Berlin, 1996, pp. 90-99

[Appleby 94] Appleby, S.; Steward, S.: *Mobile software agents for control in telecommunications networks.* BT Technl. Journal, 12(1994)2, pp. 25-34

[Arora 95] Arora, V. et al.: *Measuring High-Level Design Complexity of Real-Time Objet-Oriented Systems.* Proc. of the Annual Oregon Workshop on Software Metrics, June 5-7, 1995, pp. 2/2-1 - 2/2-11

[Barnes 93] Barnes, G.M.; Swi, B.R.: *Inheriting software metrics.* JOOP, Nov./Dec. 1993, pp. 27-34

[Bieman 94] Bieman, J.M.; Ott, L.M.: *Measuring Functional Cohesion.* IEEE Transactions on Software Engineering, 20(1994)8, pp. 644-657

[Bieman 95] Bieman, J.M.; Zhao, J.X.: *Reuse Through Inheritance: A Quantitative Study of C++ Software.* Software Engineering Notes, August 1995, pp. 47-52

[Binder 94] Binder, R.V.: *Design for Testability in Object-Oriented Systems*. Comm. of the ACM, 37(1994)9, pp. 87-101

[Booch 91] Booch, G.: *Object Oriented Design.* The Benjamin/Cummings Publ., 1991

[Brown 96] Brown, A.W.: *Component-Based Software Engineering.* IEEE Computer Society, 1996

[Brown 96a] Brown, A.W.; Wallnau, K.C.: *A Framework for Evaluating Software Technology*. IEEE Software,September 1996, pp. 29-49

[Brown 96b] Brown, A.W.; Wallnau, K.C.: *A Framework for Systematic Evaluation of Software Technologies.* in: Brown, A.W.: Component-Based Software Engineering, IEEE Computer Society Press, 1996, pp. 27-40

[Cant 94] Cant, S.N.; Henderson-Sellers, B.; Jeffery, D.R.*: Application of cognitive complexity metrics to object-oriented programs.* Journal of Object-Oriented Programming, July-August 1994, pp. 52-63

[Chen 93] Chen, J.Y.; Lu, J.F.: *A new metric for object-oriented design.* Information and Software Technology, 35(1993)4, pp. 232-240

[Chidamber 97] Chidamber, S.R.; Darcy, D.P.; Kemerer, C.F.: *Managerial Use of Object Oriented Software Metrics.* University of Pittsburgh, Graduate School of Business, Working Paper Series #750

[Chidamber 94] Chidamber, S.R.; Kemerer, C.F.: *A Metrics Suite for Object-Oriented Design*. IEEE Transactions on Software Engineering, 20(1994)6, pp. 476-493

[Chung 95] Chung, C. et al.: *A Metric of Inheritance Hierarchy for Object-Oriented Software Complexity*. Proc. of the Fifth Int. Conf. on Software Quality, October 23-26, Austin, 1995, pp. 255-266

[Chung 94] Chung, C.M.; Lee, M.C.: *Object-Oriented Programming Testing Methodology*. Int. Journal of Mini and Microcomputers, 16(1994)2, pp. 73-81

[Churcher 95] Churcher, N.I.; Shepperd, M.J.: *Towards a Conceptual Framework for Object-Oriented Software Metrics.* Software Engineering Notes, 20(1995)2, pp. 68-75

[Coad 93] Coad, P,; Nicola, J.: *Object-Oriented Programming*. Prentice-Hall Inc., 1993

[Dumke 96] Dumke, R.: *CAME Tools - Lessons Learned.* Proc. of the Fourth International Symposium on Assessment of Software Tools, May 22-24, Toronto, 1996, pp. 113-114

[Dumke 95] Dumke, R.: *Software Quality Measurement in Object-Oriented Software Development.* in: Muellerburg/Abran: Metrics in Software Evolution, Oldenbourg Publ. Germany, 1995, pp. 179-199

[DFKW 96] Dumke, R.; Foltin, E.; Koeppe, R.; Winkler, A.: *Measurement-Based Object-Oriented Software Development of the Software Project ‘’Software Measurement Laboratory’’*. Preprint Nr. 6, 1996, University of Magdeburg (40 p.)

[DFKW 96a] Dumke, R.; Foltin, E.; Koeppe, R.; Winkler, A.: *Softwarequalität durch Meßtools.* Vieweg Publ., 1996

[DuFW 95] Dumke, R.; Foltin, E.; Winkler, A.: *Measurement-Based Quality Assurance in Object-Oriented Software Development.* Proc of the ECOOP’95, Dublin, 1995, pp. 315-319

[Dumke 94] Dumke, R.; Kuhrau, I.*: Tool-Based Quality Management in Object-Oriented Software Development.* Proc. of the Third Symposium on Assessment of Quality Software Development Tools, Washington D.C., June 7-9, 1994, pp. 148-160

[DuWi 97] Dumke, R.; Winkler, A.: *Management of the Component-Based Software Engineering with Metrics.* Fifth Int. Symposium on Assessment of Software Tools, Pittsburgh, June 2-5, 1997, pp. 104-110

[DuWi 96] Dumke, R.; Winkler, A.: *Object-Oriented Software Measurement in an OOSE Paradigm.* Proc. of the Spring IFPUG'96, February 7-9, Rome, Italy, 1996

[DuZu 94] Dumke, R.; Zuse, H.: *Software Metrics in Object-Oriented Software Development. (German)* in: Lehner: Die Wartung von Wissensbasierten Systemen. Haensel Publ., Germany, 1994, pp. 58-96

[Dvorak 94] Dvorak, J.: *Conceptual Entropy and its Effect on Class Hierarchy*. IEEE Computer, June 1994, pp. 59-63

[Ebert 93] Ebert, C.: *Complexity Traces - An Instrument for Software Project Management.* Proc. of the 10th Annual Conf. on Application of Software Metrics and Quality Assurance in Industry, Amsterdam, 1993, Chapter 17 (13 p.)

[EbDu 96] Ebert, C.; Dumke, R.: *Software-Metriken in der Praxis.* Springer Publ., 1996

[Embley 95] Embley, D.W.; Jackson, R.B.; Woodfield, S.N.: *OO Systems Analysis: Is It or Isn't It?* IEEE Software, July 1995, pp. 19-33

[Fenton 97] Fenton, N.; Plfeeger, S.: *Software Metrics - A rigorous & practice approach*. Chapman & Hall Publ., 1997

Fetcke, T.: *Software Metrics in Object-Oriented Programming.* (German) Diploma Thesis, GMD Bonn/TU Berlin, 1995

[Fix 96] Fix, A.: *Conception and Implementation of a Measurement Data Base for Distributed Use*. Diploma Thesis, University of Magdeburg, July 1996

[Foltin 95] Foltin, E.: *Implementation of a problem definition measurement tool PDM*. Technical Report, University Magdeburg, 1995

[Gamma 95] Gamma, E. et al.: *Design Patterns.* Addison-Wesley Publ., 1995

[Han 94] Han, K.J.; Yoon, J.M.; Kim, J.A.; Lee, K.W.: *Quality Assessment Criteria in C++ Classes.* Microelectron. Reliability, 34(1994)2, pp. 361-368

[Harrison 96] Harrison, R.; Samaraweera, M.R.; Lewis, P.M.: *Comparing programming paradigms: an evaluation of functional and object-oriented programs.* Software Engineering Journal, 11(1996)4, pp. 247-254

[Heckendorff 96] Heckendorff, R.: *Design and Implementation of a Smalltalk Measurement Extension*. Diploma Thesis, University of Magdeburg, 1996

[Henderson 96] Henderson-Sellers, B.: *Object-Oriented Metrics - Measures of Complexity*. Prentice Hall Inc., 1996

[Hitz 95] Hitz, M.; Montazeri, B.: *Measuring Product Attributes of Object-Oriented Systems.* Proc. of the ESEC'95, Sitges, Spain, 1995, pp. 124-136

[EEE 93] IEEE Standard for a *Software Quality Metrics Methodology*. IEEE Publisher, March 1993

[ISO9126 91] ISO/IEC 9126 Standard for Information Technology, *Software Product Evaluation - Quality Characteristics and Guidelines for their Use.* Geneve 1991

[Jacobson 95] Jacobson, I.: *A confused world of OOA and OOD.* JOOP, September 1995, pp. 15-20

[Jacobson 92] Jacobson, I.: *Object-Oriented Software Engineering.* Addison-Wesley Publ., 1992

[Jones 94] Jones, C.: *Gaps in the object-oriented paradigm.* IEEE Computer, June 1994, pp. 90-91

[John 95] John, R.; Chen, Z.; Oman, P.: *Static Techniques for Measuring Code Reusability*. Proc. of the Annual Oregon Workshop on Software Metrics, June 5-7, 1995, pp. 3/2-1 - 3/2-26

[Kaschek 96] Kaschek, R.; Mayr, H.C.: *A Characterization of OOA Tools.* Proc. of the Fourth International Symposium on Assessment of Software Tools, May 22-24, Toronto, 1996, pp. 59-67

[Khan 95] Khan, E.H.; Al-Aali, M.; Girgis, M.R.: *Object-Oriented Programming for Structured Procedure Programmers.* IEEE Computer, October 1995, pp. 48-57

[Khoshgoftaar 94] Khoshgoftaar, T.M.; Szabo, R.M.: *ARIMA models of software system quality*. Proc. of the Annual Oregon Workshop on Software Metrics, April 10-12, 1994, Oregon

[Kitchenham 89] Kitchenham, B. A.; Walker, J.G.: *A quantitative approach to monitoring software development.* Software Engineering Journal, January 1989, pp. 2-13

[Kompf 96] Kompf, G.: *Conception and Implementation of a Prolog Measurement and Evaluation Tool.(German)* Diploma Thesis, University of Magdeburg, July 1996

[Kuhrau 94] Kuhrau, I.: *Design and Implementation of a C++ Measurement Tool*. Diploma Thesis, University of Magdeburg, March 1994

[Kung 95] Kung, D.C. et al: *Class firewall, test order, and regression testing of object-oriented programs.* JOOP, May 1995, pp. 65

[Kurananithi 93] Kurananithi, S.; Bieman, J.M.: *Candidate Reuse Metrics for Object-Oriented and Ada Software.* Proc. of the First Int. Metrics Symposium, May 21-22, Baltimore, 1993, pp. 120-128

[Lake 92] Lake, A.; Cook, C.: *A Software Complexity Metric for C++*. Proc. of the Fourth Annual Workshop on Software Metrics. Oregon, March 22-24 1992, 15 p.

[LaLonde 94] LaLonde, W.; Pugh, J.: *Gathering metric information using metalevel facilities.* JOOP, March/ April, 1994, pp. 33-37

[Lee 95] Lee, Y.; Liang, B.; Wu, S.; Wang, F.: *Measuring the Coupling and Cohesion of an Object-Oriented Program Based on Information Flow.* Proc. of the ICSQ'95, Slovenia, pp. 81-90

[Lee 94] Lee, A.; Pennington, N.: *The effects of paradigm on cognitive activities in design.* Int. Journal of Human-Computer Studies, (1994)40, pp. 577-601

[Lejter 92] Lejter, M.; Meyers, S.; Reiss, S.P.: *Support for Maintaining Object-Oriented Programs.* IEEE Transactions on Software Engineering, 18(1992), pp. 1045-1052

[Li 93] Li, W.; Henry, S.: *Maintenance Metrics for the Object-Oriented Paradigm.* Proc. of the First Int. Software Metrics Symposium, May 21-22, Baltimore 1993, pp. 52-60

[Li 95] Li, W.; Henry, S.; Kafura, D.; Schulman, R.: *Measuring object-oriented design.* JOOP, July-August 1995, pp. 48-55

[Lorenz 94] Lorenz, M.; Kidd, J.: *Object-Oriented Software Metrics.* Prentice Hall Inc., 1994

[Lubahn 96] Lubahn, D.: *The Conception and Implementation of an C++ Measurement Tool.(German)* Diploma Thesis, University of Magdeburg, March 1996

[Lubahn 94] Lubahn, D.: *The OOC tool description*. Technical Report, University of Magdeburg, 1994

[Marciniak 94] Marciniak, J.J.: *Encyclopedia of Software Engineering.* Vol. I and II, John Wiley & Sons, 1994

[Moser 96] Moser, S.; Nierstrasz, O.: *The Effect of Object-Oriented Frameworks on Developer Productivity.* IEEE Computer, September 1996, pp. 45-51

[OMG 95] *The Common Object Request Broker: Architecture and Specification.* Revision 2.0, Mass., July 1995

[Pant 96] Pant, Y.; Henderson-Sellers, B.; Verner, J.M.: *Generalization of Object-Oriented Components for Reuse: Measurement of Effort and Size Change.* JOOP, May 1996, pp. 19-31

[Papritz 93] Papritz, T.: *Implementation of an OOM tool for the OOA model measurement*. (German) Technical Report, TU Magdeburg, July 1993

[Patett 97] Patett, I.: *Implementation of a JAVA metrics tool. (German)* Diploma Thesis, University of Magdeburg, 1997

[Pfleeger 97] Pfleeger, S.L.; Jeffery, R.; Curtis, B.; Kitchenham, B.: *Status Report on Software Measurement.* IEEE Software, March/April 1997, pp. 33-43

[Robinson 92] Robinson, P.J.: *Hierarchical Object-Oriented Design.* Prentice Hall Inc., 1992

[Rocache 89] Rocache, D.: *Smalltalk Measure Analysis Manual*. ESPRIT Project 1257, CRIL, Rennes, France, 1989

[Rumbaugh 91] Rumbaugh, J. et al.: *Object-Oriented Modeling and Design.* Prentice Hall Publ., 1991

[Sharble 93] Sharble, R.C.; Cohen, S.S.: *The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods.*Software Engineering Notes, 18(1993)2, pp. 60-73

[Shet 97] Shet, A. et al.: *Report from the NSF Workshop on Workflow and Process Automation in Information Systems.* Software Engineering Notes, 22(1997)1, pp. 28-38

[Shlaer 96] Shlaer, S.; Mellor, S.J.: *Objektorientierte Systemanalyse.* Hanser Publ., 1996 (Original: 1988)

[Tepfenhart 97] Tepfenhart, W.M.; Cusick, J.J.: *A Unified Object Topology.* IEEE Software, January 1997, pp. 31-35

[UML 97] *Unified Modeling Language - Summary.* version 1.0.1, Santa Clara, USA, March 1997

[UML 97a] *Unified Modeling Language - Glossary & Notation Guide.* version 1.0, Santa Clara, January 1997

[Wasserman 88] Wasserman, A.I.: *Tool Integration in Software Engineering.* Lecture Notes in Computer Science, Volume 467, 1988, pp. 137-149

[Welch 96] Welch, L.R.; Lankala, M.; Farr, W; Hammer, D.K.: *Metrics for quality and concurrency in object-based systems.* Annals on Software Engineering, 2(1996), pp. 93-119

[Wilde 92] Wilde, N.; Huitt, R.: *Maintenance Support for Object-Oriented Programs.* IEEE Transactions on Software Engineering, 18(1992), pp. 1038-1044

[Wirfs-Brock 90] Wirfs-Brock, R.; Wilkerson, B.; Wiener, L.: *Object-Oriented Design.* Englewood Cliffs Publ. 1990

[Zuse 94] Zuse, H.: *Foundations of the Validation of Object-Oriented Software Measures*. in: Dumke/Zuse: Theory and Practice of Software Measurement (German). DU-Publ., 1994, pp. 136-214

[Zuse 97] Zuse, H.: *The Software Measurement Framework.* to be published

8 Glossary

AC	Attribute Complexity: sum of the attribute values of a class; based on the evaluation: Boolean or integer (0), char (1), real (2), array (3-4), pointer (5), record, struct (6-9), file (10)

ADI	Attribute Definition Indicator
AHF	Attribute Hiding Factor: sum of all visible/usable attributes of all classes divided by all attributes of all classes
AIF	Attribute Inheritance Factor: sum of all inherited attributes in all classes
AII	Attribute Implementation Indicator
AMI	Attribute Modification Indicator
BOA	Basic Object Adapter
CAME	Measurement Choice, Adjustment, Migra-tion and Efficiency
CAME Tool	Computer Assisted Software Measurement and Evaluation Tool
CASE	Computer Aided Software Engineering
CBO	Coupling Between Object classes: the number of other classes to which it is coupled
CBSE	Component-Based Software Engineering
CCM	Cognitive Complexity Model: sum of chunk understanding, complexity and difficulty of tracing
CDBC	Change Dependency Between Classes: the potential amount of follow-up work to be done when a server class is being modified
CDI	Class Definition Indicator
CFW	Class FireWall: the set of classes that could be affected bay changes to a special class; the test order is the topological sorting of the CFW graph including the dependence relation
CH	Computing Cohesion
CII	Class Implementation Indicator
CLOS	Common LISP Object System
CMI	Class Modification Indicator
COF	Coupling Factor: maximum possible number of couplings in all classes
CORBA	Common Object Request Broker Archi-tecture
COS	Comon Object Services
COTS	Components Off-The-Shelf
CPD	Classes Per Developer
DAC	number of ADTs defined in a class
DCE	Distributed Computing Environment
DIT	Depth of Inheritance Tree: the maximum length from the node to the root of the tree
GR	Generic Reuse: reuse by generic functions/ macros
HOOD	Hierarchical Object-Oriented Design
HTML	Hypertext Markup Language
ICH	I-based cohesion: information flow-based, message argument related, internal count
ICP	I-based coupling: information flow-based, message function related, external count
IDL	Interface Definition Language
KE	number of Known Errors
LCOM	Lack of Cohesion in Methods: the set of instance variables used by the method
LD	Locality of Data: the sum of the non-public and inherited protected instance variables divided by the sum all variables of a class
LR	Leveraged Reuse: reuse by method inheri-tance
MHF	Method Hiding Factor: sum of all visible/callable methods of all methods divided by the number of all methods of all classes
MIF	Method Inheritance Factor: sum of all inherited methods in all classes
MPC	Message Passing Coupling: number of send-statements defined in a class
MR	number of modifications requested
NCM	Number of Class Methods
NCV	Number of Class Variables
NIM	Number of Instance Methods
NIV	Number of Instance Variables
NKC	Number of Key Classes
NMA	Number of Methods Added
NMI	Number of Methods Inherited
NMO	Number of Methods Overridden
NOC	Number Of Children: the number of immediate subclasses
NOM	Number Of Methods
NOS	Number Of Subsystems
NOT	Number of Tramps: number of extraneous (not referred to by the method body) parameters
NSC	Number of Support Classes
NSS	Number of Scenario Scripts
OC	Operation Complexity: sum of the method values for a class based on the empirical evaluation as null (0), very low (1-10), low (11-20), nominal (21-40), high (41-60), very high (61-80), extra high (81-100)
OCL	Object Constraint Language
ODA	Object Database Adapter
OLE	Object Linking and Embedding
OMA	Object Management Architecture
OMG	Object Management Group
OMT	Object Modeling Technique
OO	object-oriented
OOA	Object-Oriented Analysis
OOC	Object-Oriented classes Comparison
OOCM	Object-Oriented Conceptual Modeling is based on entropy measures for the OOA relating to class hierarchy as specificity (class refinement), as (semantically) consistency and (semantically) distance
OOD	Object-Oriented Design

OOP	Object-Oriented Programming
OORA	Object-Oriented Requirements Analysis
OOSA	Object-Oriented Systems Analysis
OOSD	Objet-Oriented Software Design
OOSE	Object-Oriented Software Engineering
ORB	Object Request Broker
OS	Operating System
OSF	Open Systems Foundation
PCM	Percentage of Commented Methods
PCTE	Portable Common Tool Environment
PD	Problem Definition
PDC	Person-Days per Class
PDL	Program Design Language
PDM	Problem Definition Metrics Tool
PMT	Prolog Metrics Tool
POF	Polymorphism Factor: actual number of possible different poly-morphic situations
PRC	Problem Reports per Class
RDD	Responsibility-Driven Design
RFC	Response For a Class: the response set for a class
RPC	Remote Procedure Call
SC	Subjective assessment of Complexity provided by the system developer in ordinal integer scale
SDI	Service Definition Indicator
SFC	Strong Functional Cohesion: the token of the data slices divided by all data tokens in a program
SII	Service Implementation Indicator
SIZE1	number of semicolons in a class
SIZE2	number of attributes + number of local methods in a class
SMI	Service Modification Indicator
SMLAB	Software Measurement Laboratory of the University of Magdeburg
SQA	Software Quality Assurance
SRD	Software Requirement Document
TKE	Time to fix Known Errors in minutes
TMR	Time to implement Modifications
UML	Unified Modeling Language
URI	Unit Repeated Inheritance: a set of class hierarchy regions with the Euler's region number 2 for reducing the OO test cases
VOD	Violations of the Law of Demeter: coupling between classes in both directions (as minimizing)
VR	Verbatim Reuse: reuse of library compo-nents
WAC	Weighted Attributes per Class: number of attributes weighted by their size
WMC	Weighted Methods per Class: sum of the (McCabe) complexities