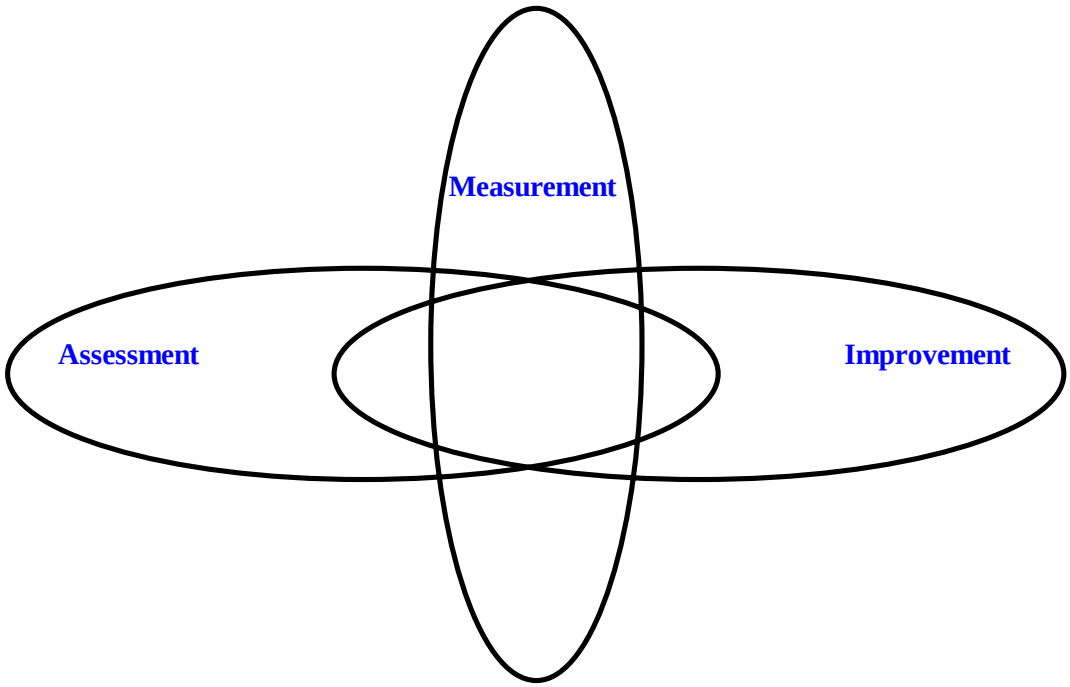
Volume 2, Number 2, December 1997

METRICS NEWS

Journal of the GI-Interest Group on Software Metrics



Editors: ***R. Dumke, C. Ebert, E. Rudolph, H. Zuse***

Otto-von-Guericke-Universität

Magdeburg

The *METRICS NEWS* can be ordered directly from the Editorial Office (for address see below).

Editors:

Reiner Dumke
Professor on Software Engineering,
University of Magdeburg, FIN/IVS,
Postfach 4120, D-39016 Magdeburg, Germany
Tel.: +49-391-67-18664, Fax: +49-67-12810
email: dumke@ivs.cs.uni-magdeburg.de

Christof Ebert
Dr.-Ing. in Computer Science
Alcatel Telecom, Switching Systems Division,
Fr. Wellensplein 1, B-2018 Antwerpen, Belgium
Tel.: +32-3-240-4081, Fax: 32-3-240-9935
email: christof.ebert@alcatel.be

Eberhard Rudolph
Professor on Software Engineering
Hochschule Bremerhaven, FB2 System Analysis,
Mozartstr. 37, D-27570 Bremerhaven, Germany
Tel.: +49-471-26142, Fax: +49-471-207389
email: rudolph@oscar-e.hs-bremerhaven.de

Horst Zuse
Dr.-Ing. in Computer Science
Technical University of Berlin, FR 5-3,
Franklinstr. 28/29, D-10587 Berlin, Germany
Tel.: +49-30-314-73439, Fax: +49-30-314-21103
email: zuse@tubvm.cs.tu-berlin.de

Editorial Office: Otto-von-Guericke-University of Magdeburg, FIN/IVS, Postfach 4120, 39016 Magdeburg, Germany
Technical Editor: DI Erik Foltin
The journal is published in one volume per year consisting of two numbers. All rights reserved (including those of translation into foreign languages). No part of this issues may be reproduced in any form, by photoprint, microfilm or any other means, nor transmitted or translated into a machine language, without written permission from the publisher.

© 1997 by Otto-von-Guericke-Universität Magdeburg. Printed in Germany

EDITORIAL

This is the third issue of a new scientific journal in the field of software metrics and related quantitative aspects, the

METRICS NEWS.

The title was chosen to reflect the Journals attempt to summarize recent software metrics trends as position papers, chosen papers from our metrics workhops, and *news* (as information about the software metrics research area in the world, new books and conferences). The editors are working many years in the software metrics field and are specialized in measurement frameworks, function point analysis, measurement theoretical view, and practical applications.

The background of the METRICS NEWS contributors is the GI-interest group on software metrics founded in 1991. All members from the industry or academia are invited to present their experience or research results in the area of software quality assurance, software metrics, process management, software measurement frameworks etc.

The English language was chosen to reflect the international character of our research contacts and results embedded in European initiatives.

The editors are grateful to the Otto-von-Guericke University of Magdeburg for publishing this journal.

We hope that the new journal will be helpful to increase the awareness of the importance of software metrics issues in the improvement of software development processes and products.

The Editors

The annual Worshops of the German Interest Group on Software Metrics are related to the main topics in the area of software quality assurance, software process and product improvement and software evaluations based on theoretical and practical aspects of software measurement. Some of the topics in the last workshop were

- the practical experiences in the application of metrics programs in an industrial environment,
- the analysis and use of object-oriented software systems,
- the analysis and use of the function point method,
- theoretical research of software metrics and metrics validation,
- application of metrics tools.

The 7th Workshop on Software Metrics was focused on the quality assurance of object-oriented systems, practical experiences in application of software metrics and theoretical aspects of metrics as software measures. The following papers have been presented:

Sneed, H. (SES Munich)[1)]:
Measuring Reusability of Legacy Software Systems,

Zuse, H. (TU Berlin)[2)]:
The Role of Measurement Theory in the area of Software Measurement,

Schwald, A. (IT Consulting, Munich)[3)]:
Metrics, Poeple and Their Roles in a Software Project,

Dumke, R. (University of Magdeburg)[3)]:
Quality Assessment of Objekt-Oriented Software Development Methods,

Schmietendorf, A. (Telekom Berlin)[3), 4)]:
Metrics of Object-Oriented Software Development Technologies,

Ebert, C. (Alcatel Antwerp, Belgium)[1)]:
Quality Management of Software Process Improvement,

Foltin, E. (University of Magdeburg)[1)]:
Concepts of Metrics Data Bases,

Wuest, J. (IESE Kaiserslautern)[1)]:
A Unified Framework of Coupling Measurement in Object-Oriented Systems.

An interesting panel discussion about the benefits, problems and risks of the metrics use was another highlight of this Workshop.

The **8th International Workshop on Software Measurement** will be held at the University of Magdeburg and is organized by the German Interested Group on Software Metrics and the Canadian Software Metrics Interest Group (CIM). The Workshop will be presented in the MBone Video conferencing service and can be observed worldwide. The Call for Paper will be published in the next Journal.

Measurement in Physics and Software Engineering

Part I

Horst Zuse,Technische Universität Berlin

Abstract

In this contribution consisting of three parts we discuss the differences of measurement in physics and software engineering measurement. Measurement in physics has a very long tradition and the concepts of measurement there are clear. It is our impression that a comparison of measurement in physics and software engineering can help to understand the problems in the software measurement area in a better way.

Keywords

Measurement, physics, software engineering.

1 Introduction

Since software engineering measurement is not a well understood science today, we will introduce some concepts of measurement in physics and compare them with measurement in the software engineering area. For this reason we discuss some differences of measurement in physics and software engineering. The general question is: what is problematic in software measurement? Can we learn from measurement in physics and can we transform this to software engineering measurement? In [19] you also can find a more detailed discussion of this subject.

One basic problem of every science ascribing itself to the characteristic *empirical* concerns the meaning of experience. Namely, in the field of empirical science, theories as systems of statements always refer to what can be experienced, in contrast to mathematics and logic, where truth can be established independently of the nature of any reality. The function of experience is therefore considered as a final test of the validity of these statements called science. Most scientists today agree upon the fact that observation always implies certain assumptions, concepts, etc. - in short: that it is conducted by theory.

The question is why is software (engineering) measurement so problematic? One answer may be, following Roche et al. [12], that software engineering is a highly complex process producing highly complex products. Moreover, each project and its products tend to be something of *one off* in nature, a point highlighted by Schneidewind as a difficulty in validating a methodology [13]. Other problems are that people do not like to be controlled by software measures. And, last not least, there is a lack of an intensive education of people in software measurement regarding both: a theoretical framework for software measurement and a soundly planning of experiments.
The major problem of measurement in software engineering, but also in the area of artificial intelligence, is a skepticism of using numerical values because there is no satisfaction in the interpretation the numbers and a semantic of the values is missing. This lack may be true in some cases, but not generally. The assignment of simple numbers to hypotheses without knowing the empirical evidence of these numbers is a major mistake. The empirical evidence of numbers can be characterized, among others, by several empirical conditions and scale types. Numbers are elements of a scale, that means, they are subject of a homomorphic mapping of an empirical to a numerical relational system and vice versa. Mostly, these facts are neglected.

Novertheless, we think, today it is widely accepted that software measurement is a valuable technique for understanding, guiding, controlling and improving software development. It is an interesting phenomenon that the Measure LOC and the Measures of McCabe [11] today still are the most used and discussed software measures. The Measure of McCabe was defined for single module complexity but also for the entire system complexity. The question is still discussed whether the Measure of McCabe is a *good* or a *bad* measure. Another unsolved question is whether the Measure of McCabe can be used as a predictor for software maintenance attributes. We think the reasons for these discussion are the following: firstly, there is a lack of education in the area of software measurement, secondly, many people believe that software measurement is an easy thing, and thirdly, although there exists a proper theory for software measurement - called measurement theory (see for that Zuse [15], [16],

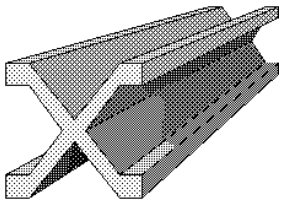
[17], [18], Bollmann-Sdorra and Zuse [4], Baker et al. [3], Fenton et al. [5], [6]) - only a few people consider and apply this theory.

Statistical methods are often used in the software measurement area. This is justified because there are existing many empirical data. It is our view, that a theory of software measurement and the application of statistical methods support each other.

2 Measurement in Physics and Software Engineering

In Part I only consider some general differences between measurement in physics and software engineering. In the Parts II and III we will demonstrate the differences with examples.

2.1 Measurement in Physics



Measurement in physics has a long tradition. In physics quantitative laws are more important than qualitative laws. In physics qualitative laws usually are considered as trivial. Qualitative laws for the measurement of length, in the form of the extensive structure, were developed as measurement has been done successfully some hundred years. The problem of measurement of length was not the qualitative conditions. The problem was to measure length with a high accurateness. In 1824, the English Government via a decree laid down the length of a yard [7], p.262. A basis for that was the length of a pendulum that had a period of oscillation of one second. There were a lot of conferences with contradicting discussions about a normalized length. In 1875 seventeen nations signed a convention about the measurement of length, and one hundred years later, more than fourty-four nations signed the contract. Before this time, it held: *Jedes deutsche Ländchen / hat sein eigenes Quäntchen / eigene Maße hat / fast jeder deutsche Staat.* (Translation by the author: *Every German country / had is own small quantity / own measures has / almost every German state*). The contracting discussion of length measurement were not based on the question: what is length? It was a political problem.

In physics, mostly we have facts, which we want to measure. Humans are not directly involved in this process because the measurement process mostly does not depend on the view of humans. The discussion of empirical conditions plays a more important role in the social sciences. For example, considering a resistor, the length, the height, the weight, etc. can be measured. Empirical or qualitative conditions related to resistors mostly are not considered. In physics we have standards and a well defined system of units.

In physics, very often density measures are used. The natural law

$$d = m / V,$$

where m is the mass, V the volume and d the density, is well known. It has been observed that the relationship of mass to volume for homogeneous substances is equally. It is independent of the size. This law was derived by the measurement of mass and volume. The law d = m / V is a quantitative one, while the measurement of mass and volume are based on non-quantitative

assumptions. From physics we know the Law of Pythagoras and the famous formula: $c^2 = a^2 + b^2$. We have integers, like the power of two. This is also the case with energy: $E = ½ m v^2$. Here we have v^2 and not, for example: $v^{1.5}$. Another example is the formula

$$s = ½ g t^2,$$

which can be seen as a prediction model. From the Time t, the gravity g then Was s is predicted. Do we have similar prediction models in software engineering measurement? In the area of software measurement we do have real numbers in such formulas (not integers as in physics), and density measures have another behavior in physics than in software measurement. The density in software measurement is not independent on size.

On the Conference in *Honor of H.v. Helmholtz and R.D. Luce: Foundations of Measurement: The Theory of Representability and the Nature of Numbers, Kiel, Germany, November 1994*, the role of numbers in physics was a major topic, as already discussed in [8], [1], [2]. Among others, the question was discussed whether the numbers are *in the physical objects*, and the task of scientist is to *find* them or to *get them out* of the objects. This is a very interesting view, but it would be beyond this book to discuss it more deeply.

2.2 Measurement in Software Engineering

We mean, that the situation in software measurement is differently to physics. In the past, software measurement mainly was seen from a quantitative view, too. Very often, the well defined discipline of measurement in physics was stated as a standard or a model for software measurement. An empirical impact of quantitative conditions or results of measurement was not discussed, explicitly. That means, the situation of measurement in software engineering was considered similar to physics. Qualitative conditions and the consideration of scale types were left out. However, implicitly, authors combined quantitative results with empirical statements. In 1974, Wolverton [14] did this with the Measure lines-of-code. He assigned the empirical attribute productivity to the Measure LOC. The requirement of certain conditions for software measures reflects impact of humans in the area of software measurement.

In the area of software engineering, we use so-called latent variables, like in the social sciences. Latent variables are such like intelligence or aggressiveness [10], p.122. In software measurement such latent variables are complexity, maintainability, etc. Maintainability of software is analyzed with dozens of different measures. For example, all these measures are used to quantify the term maintainability, but they are measuring different aspects of maintainability. Length also can be measured with different measures, but these measures can be derived by admissible transformation from the other ones. It is only the problem of uniqueness.

Empirical views and measurement also are connected in the ISO9126 standard. In 1991 the ISO9126 standard [9] has been established by the ISO-Organization. The result is the following:

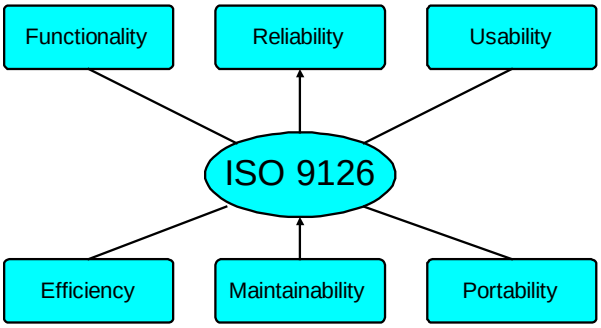


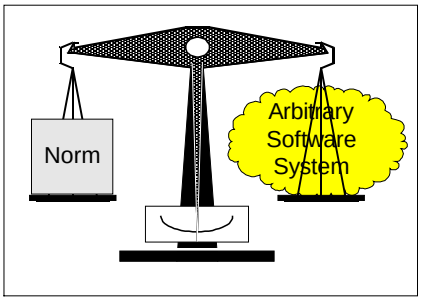
Figure 2.1: The ISO 9126 standard.

ISO9126 was established to characterize the quality of software. It took six years to develop and define the qualitative statements above. Simplified, we can explain the software quality attributes as follows.

1. **Functionality**: Does the software satisfy stated needs.
2. **Reliability:** How often does the software fail?
3. **Usability**: How easy is the software to use?
4. **Efficiency**: How good is the performance of the software?
5. **Maintainability**: How easy is the software to repair?
6. **Portability:** How easy is the software to transport?

These six software quality attribute are attributed with sub-attributes. We illustrate this here.

1. **Functionality:** Suitability, Accurateness, Interoperability, Compliance, Security.
2. **Reliability:** Maturity, Fault Tolerance, Recoverability.
3. **Usability:** Understandability, Learnability, Operability.
4. **Efficiency:** Time behavior, Resource behavior.
5. **Maintainability:** Analyzability, Changeability, Stability, Testability.
6. **Portability:** Adaptability, Installability, Conformance, Replaceability.



The task of software measurement is to characterize the qualitative attributes of the ISO9126 norm with software measures. Since there does not exist a unique view, hundreds of measures were created. Analogous to physics, there is the idea whether we can compare a software quality attribute to a norm. In physics we are doing this all the time. Beam scales are used to compare masses of all kinds. In the area of software measurement, it is more difficult to find the *Ur-meter* in form of a module. In the software engineering area, very often correlation coefficients are used in order to figure out relationships between variables. This is not the case in physics. Correlation coefficients are used if the knowledge is poor.

Another important topic are the units. In physics, a well defined system of units exists. The question is whether such a system of units exists in the software measurement area.

In short: software measurement mostly deals with qualitative conditions, while measurement in physics mostly address the quantitative aspects.

2.3 Measurement in Physics and Software Engineering – Counting

Measurement in physics and in software engineering is based on counting anything. We illustrate this with the next picture.

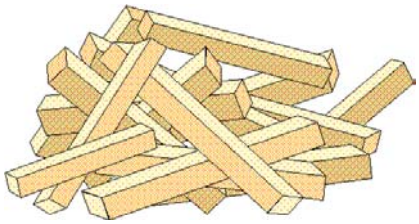


Figure 2.2: Wooden boards.

Wooden boards can be counted. We can say: These are 24 wooden boards. We can assign a unit to them and we also can say: These are two Dozen wooden boards. That means, we can transform the numbers and everybody knows what we mean. This transformation of numbers is well known.

In the software engineering measurement area we also can count objects or entities. The next picture illustrates this.

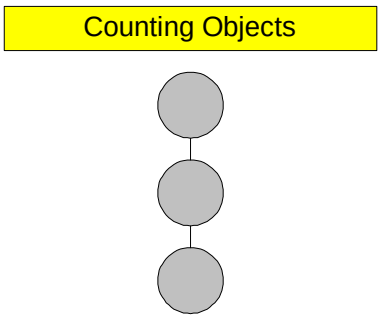


Figure 2.3: Counting of nodes.

In the software engineering area we can count nodes in a flowgraph. The nodes are representing executable statements in a program. We can count this nodes. We can say: These program has 24 nodes. We also can assign a unit, for example LOC. We can transform LOC to KLOC.

However, there are important differences of measurement in physics and in the software engineering area.

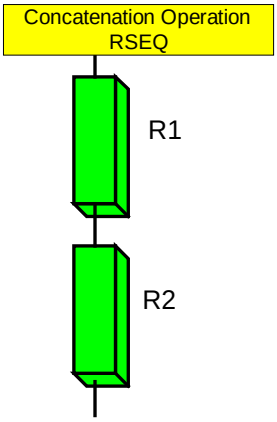


Figure 2.4: Concatenation of two resistors in electrical engineering.

In physics or in electrical engineering we have resistors. In order to measure the resistance of a resistor we can use an OHM-Meter. If we concatenate two resistors in a sequence, it holds for the whole Resistor R consisting of R1 and R2 in a sequence the law:

$$R = R1 + R2,$$

where R is the resistor consisting of both Resistors R1 and R2. We have here an additive law. The question is whether we have such cases in the software engineering area, too.

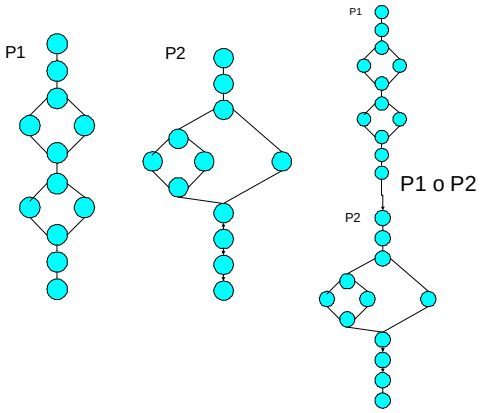


Figure 2.5: Concatenation of two Flowgraphs P1 and P2 to Flowgraph P1 o P2.

The question is whether we can concatenate, for example, flowgraphs in the same kind as resistors. If we can do this then the question is whether it holds:

$$u(P1 \text{ o } P2) = u(P1) + u(P2)?$$

We denote with u a measure, for example a complexity measure. The statement P1 o P2 is the sequential concatenation of two Flowgraphs P1 and P2 to the sequence P1 o P2. u(P1 o P2) means the application of the Measure u to the sequence of the Flowgraphs P1 o P2.

In the next edition of this journal we will show that there are existing similar cases in physics and software engineering measurement, but important differences, too.

References

[1] Adams, E. W.; Fagot, R. F.; Robinson, R. E.: *On the empirical status of axioms in theories. of fundamental measurement.* Journal of Mathematical Psychology, 7, 1970, 379-409.

[2] Adams, E. W.: *On the nature and purpose of measurement.* Synthese, 16, 1966, 125-169. Also in: Lieberman, B. (Ed.): Contemporary problems in statistics. New York: Oxford University Press, 1971, 74-92.

[3] Baker, A.L.; Bieman, J.M.; Fenton, N.; Gustafson, D.A.; Melton, A.; Whitty, R.A.: *NATO Project 0343/88: Formal Foundations of Software Measurement*. Report of 1989 Meetings.

[4] Bollmann-Sdorra, P.; Zuse, H.: *Prediction Models and Software Complexity Measures from a Measurement Theoretic View.* Proceedings of the 3rd International Software Quality Conference, Lake Tahoe, Nevada, October 4-7, 1993.

[5] Fenton, N.: *Software Metrics: A Rigorous Approach*, Chapman & Hall, 1991.

[6] Fenton, N.; Pfleeger, S.: Software *Metrics- A Rigorous Approach*, Thomson Publisher, 1996.

[7] Fischer, Ernst, Peter: *Aristoteles, Einstein & Co. - Eine kleine Geschichte der Wissenschaft in Portraits*. Piper GmbH, München, 1995.

[8] Helmholtz, H. von.: *Zählen und Messen erkenntnisstheoretisch betrachtet.* In: Philosophische Aufsätze. Eduard Zeller zu seinem fünfzigjährigen Doctor-Jubiläum gewidmet. Leipzig: Fues' Verlag, 1887, pp. 15-52.

[9] ISO/IEC Standard: *ISO 9126 Software Product Evaluation - Quality Characteristics and Guidelines for Their Use*, 1991.

[10] Kriz, Jürgen: *Methodenkritik Empirischer Soziaforschung - Eine Problemanalyse sozialwissenschaftlicher Forschungspraxis*. Teubner Studienskripten, 1981.

[11] McCabe, T.: *A Complexity Measure. IEEE Transactions of Software Engineering*, Volume SE-2, No. 4, pp. 308-320, December 1976.

[12] Rochester, John; Jackson, Mike: Software Measurement Methods: *Recipes for Success. Information and Software Technology*, 1994, Volume 36, No. 3, pp. 173-189.

[13] Schneidewind, Norman F.: *Validating Software Metrics: Producing Quality Discriminators*. In: Proceedings of the Conference on Software Maintenance (CSM91), Sorrento, Italy, October 1991, and in: Proceedings of International Symposium on Software Reliability Engineering, 1991.

[14] Wolverton, R.W.: *The Cost of Developing Large-Scale Software*. IEEE Transactions on Computer, Volume C-23, No. 6, pp. 615-636, June 1974. Also in: Tutorial on Programming Productivity: Issues for the Eighties, IEEE Computer Society, Second Edition, 1986.

[15] Zuse, Horst; Bollmann, P.: *Using Measurement Theory to Describe the Properties and Scales of Static Software Complexity Metrics*. SIGPLAN Notices, Volume 24, No. 8, pp.23-33, August 89.

[16] Zuse, Horst: *Software Complexity: Measures and Methods*. DeGruyter Publisher 1991, Berlin, New York, 605 pages, 498 figures.

[17] Zuse, Horst; Bollmann-Sdorra, Peter: *Measurement Theory and Software Measures*. In: Workshops in Computing: T.Denvir, R.Herman and R.Whitty (Eds.): Proceedings of the BCS-FACS Workshop on Formal Aspects of Measurement, South Bank University, London, May 5, 1991. Series Edited by Professor C.J. Rijsbergen. ISBN 3-540-19788-5. Springer Verlag London Ltd, Springer House, 8 Alexandra Road, Wimbledon, London SW19 7JZ, UK, 1992.

[18] Zuse, Horst: *Foundations of Validation, Prediction, and Software Measures*. Proceedings of the AOWSM (Annual Oregon Workshop on Software Metrics), Silver Fall State Park, Oregon, 1994.

[19] Zuse, Horst: *A Framework for Software Measurement*. DeGruyter Publisher, Berlin, Hawthorne, USA, 1997, 755 pages.

Metrics, People and Their Roles in a Software Project

Andreas Schwald, Munich

Abstract

Technical and commerial goals of a project require the synthesis of multiple goals and different views within a project. This is important for features which require evaluation based on personal preferences. The shortcomings of subjective evaluation should be compensated by the application of objective quality criteria which can be evaluated automatically. This position paper emphasizes the necessity of complementary views and their articulation by persons in charge of a definite role within the project. Quality metrics and other measurements are means for rational communication between persons and groups representing different goals and complementary views. This interaction of views is indispensable in the synthesis of a common set of accepted goals and their pursuit in the development and assessment of software.

1 Introduction

Some time after the Olympic Games in Rome (10 gold medals and some 15 others for Italy, 9 gold medals and some 25 others for Germany) two young poeple (german and italian) had a dispute over the virtues of their nations. The german's question „Who made more medals?" was answered by „Gold medals - Italy!".

This episode shows the importance of clear quantitative criteria for the decision of controversial issues, and the implications of criteria selection.

For software metrics, this applies to the selection, goal orientation, and interpretation of criteria, the definition of measurement rules, and their implementation in appropriate tools supporting collection and analysis of metric data. It is easy to find astonishing examples of software projects producing obviously useless results without violating the least of the contractual obligations.

2 Roles, information needs, and measurable properties

„Programers and analysts have a restricted view-point of the software system under consideration" [8]. This holds true also for persons representing other roles in a project.
Example: Assessing the degree of completeness of a program component.

While a programmer is improving the performance and user friendliness of his component, the project manager is not interested in „gold plating" (Boehm), since there is strong pressure for completion in order to fulfil the contractual obligations. A QA person is rightly unwilling to compromize the specified quality criteriy, while a particular user may be quite happy with a rather restricted functionality well suited to his or her application.

The restriction to a narrow view according to a particular role is a fundamental survival strategy for poeple dealing with complex systems. However, this „local" behavior requires compensation. Team building aims at a group comprising competent poeple which are in

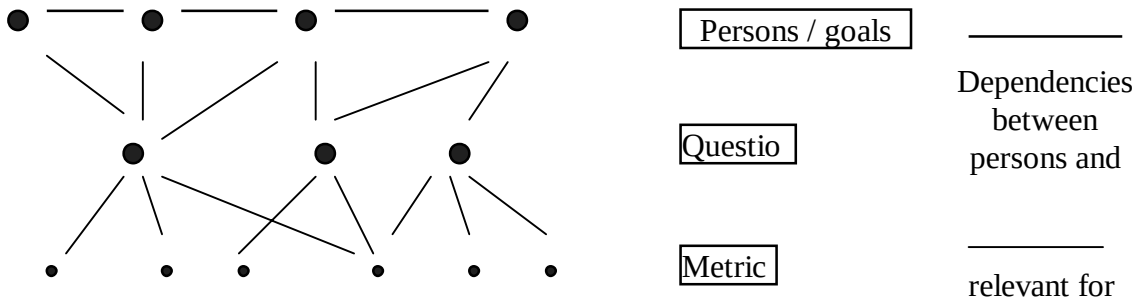
charge of specific tasks (e. g. quality assurance or configuration management) and represent the project goals related to their responsibilities. The qualification these experts and their personal interest to properly fullfil their assignment will ensure appropriate consideration of their views in the balance of multiple project goals.

Project goals are established by general quality requirements, standards, and by the consensus of the interested parties. The project contract documents this agreement, all subsequent decisions are based on this document. This framework protects and restricts particular views and goals of interested parties.

In a more detailed view, the balance of project goals is not static. A contract may be incomplete and subject to changes. Such changes occur due to many different reasons (e. g. changes of environment, of technical or financial circumstances, new insights, new personnel or shifts of personal interests and power, deadline pressure). In this process, team members have different information needs in order to fulfil their tasks and to represent their views. Striving for the general project goals means collection and comprehension of many specific informations, and compromising between different views for every level of abstraction and for every stage of the software process.

This adjustement of goals is vital for the success of a project. It is important to expose problems to an open discussion. There are logical and personal dependencies between the goals of a project and the persons defending them. These subjective influences are the driving forces forces of a project - the may also act as project impediments in a rather destructive way.

This consideration of multiple goals leads to a modification of the well-known GQM paradigm:



The selection of metrics should be goal oriented, i. e. satisfy the information needs of responsible persons. The definition must be objective, i. e. independent from a particular view. ([2]: „An objective, or algorithmic, measure is one that can be computed precisely according to an algorithm. Ist value does not change due to changes in time, place or observer.“) The well-known tendency to supplement information gaps according to specific habits, interests, and knowledge must be compensated by continuous adjustment of views which is based on measurement and driven by the commitment of poeple who are responsible for accepted project goals.

3 Rational Communication

3.1 Principles of rational communication (RC)

[9] discusses principles of rational communication for scientific discourse. „Communication norms are not absolute, they depend on a value (goal). This value is striving for truth - to find true statements and to establish valid norms. ... If the goal of communication is striving for truth then certain conditions will hold for the communication. These principles and conditions are considered to be necessary conditions for *rational communication.* Rational communication is based on interaction among two or more persons using linguistic utterances referencing an object domain.“ This discussion covers several areas

- Poeple: Several persons are involved.
- Language usage: A „text“ is a sequence of sentences which assumes some background knowledge. It is possible to infer common consequences from several sentences of a text (together with the background knowledge and possibly some hypothetical premises).
- Common base for understanding: The applicability of logic and semantics to the sentences, and the application of the same set of rules by RC partners is a base for understanding. Furthermore, the meanings assigned to an expression by the partners of an RC must overlap. It is important to clarify and to discuss implicit assumptions, and to unmask suggested suppositions. Well founded scientific results should be accepted. However, in case of conflicting opinions, it is necessary to restrict the communication on a narrower common base, or to consider such opinions to be hypotheses and to keep in mind their hypothetical nature.

3.2 RC and Software

These RC principles provide guidance for dealing with different views. The clarification of implicit assumptions, and clear recognition of hypothetical statements are vital for RC (and for the success of a project). Enthusiasm without risk assessment may be disastrous. Many statements related to software are hypothetical, even some empirically based assertions due to possible errors and unclear interpretations. Some examples:

- Plans (requirements, estimates, specifications, ...) are hypothetical as far as they predict future events. Within a contract, they are accepted standards.
- Test cases form a sample. Statements on program correctness are hypothetical or restricted (to formally verified properties).
- A system description for a particular role (e. g. user manual) is incomplete, complementary information (e. g. internal documentation, code) may be unavailable.
- The complexity of many software products is a reason for information gaps (e. g. for casual users). Timing restrictions may necessitate decisions based on rather incomplete information (e. g. preselection of software products).
- A new program version is the result of many fixes, changes, and enhancements. Therefore, knowledge based on the experience with older versions becomes hypothetical.
- The relationship between measured attribute values (e. g. complexity) and a property of interest (e. g. effort for and error rates of program changes) is hypothetical, since it depends also on many other factors.

RC may compensate the tendency to narrow judgements, it will explicate the assumptions and risks of hypothetical statements. Often, this will require a more precise formulation of a statement, e. g. for „Program P contains bugs.“ This may be stated more precisely, e. g.: „According to user U's report, dated 20-8-97, he experienced five failures of program P's version 1.8 which was installed on workstation W two weeks ago.“ Even this wording relies on background knowledge, e. g. for the configuration of W and the role of B (normal use, acceptance test, ...). Inherently imprecise statements like „about four weeks“ need consistent interpretation (probabilty and limits of acceptable deviations).

3.3 Approaches to objectivity: quantification and refinement (modeling)

Aiming at objectivity of measurements and assessments (independence from persons, reproducibility; [2]: „The value of an objective, or algorithmic measure does not change due to changes in time, place, or observer“) is important in order to achieve clear decisions based on facts which are accepted also by the proponents of conflicting interests. Quantification of attributes requires precise specification (e. g. „100 km/h“ instead of „enormous speed“, „within two hours“ instead of „as soon as possible“). For complex features, refinements (subgoals, components, checklists, set of criteria) and modeling are required for the definition of measurable attributes. [6] emphasizes the importance of models: „Characterize the environment to the necessary degree to understand the measurement goals, the experimental design, and the data interpretation.“ The specification of such models is a prerequisite for the classification and definition of relevant attributes, for the definition of measures, and for measuring procedures.

Refinement may address different layers and views, e. g. for portability: specification of a range of platforms, design rules, standards for the use of programming languages and system interfaces. Refinement does not necessarily imply quantification or a precise definition (e. g. ISO 9126: „Portability: A set of attributes that bear on the ability of software to be transferred from one environment to the other ... adaptability, installability ...“). Refinement defers the definition of unclear boundaries to a more detailed level, where it may be easier to clarify some of the hazy issues. In this way, refinement may clarify the scope and the content of concepts in a particular context. Ambiguities exist in colloquial speech and technical language (e. g. „specification“). Understanding of diverging interpretations and sufficient commonality are necessary for cooperation within a project.

Global ratings result from the condensation of informations, typically by the computation of weighted means of attribute values for components (e. g. „90% completion of a program“ derived from „70% of modules accepted“ and „30% of modules in test“). Obviously, such predictions based on statistical results are inappropriate for the identification of error prone or difficult items which require special attention. This type of information is appropriate for poeple in charge of other tasks who are unable to go in the details, and for global statements on a project or product - e. g. for an acceptance or a purchase decision. Sometimes, global metrics or quantitative requirements result from bundling quite different attributes or incongruent wishes of individuals.

Specification, modeling, and quantification are means of rational communication. They may show the existence of implied assumptions and requirements. The purposes of metrics include

- Propositions on the subject matter which are accepted by the interested parties (valid standards or facts).
- Indicators for features of interest with respect to agreed or implied project goals
- Measurable goals and requirements
- Measures for project control

Measurement aims at objectivity, not necessarily at precision. Unprecise and hypothetical statements may be necessary and useful information for preparing and supporting decisions.

4 Useful information

4.1 Decision support

The purpose of information is decision support. The level of precision and safety which is required and achievable depends on the useful precision for the purpose in question, on inherent sources of errors, on the precision of mesurements, the effort and time limits for information gathering. View specific selection and weights of criteria should be explicated, based on accepted requirements, and support the goals of a project in the whole. Decisions should be based on true propositions, accepted standards, and well founded hypotheses. Assessment and monitoring of the risks implied in the acceptance of such hypotheses is an obvious requirement. Even precise measures may be error prone and open for different interpretations. Qualification, experience, and goal orientation of experts are indispensable for the interpretation of software and process measurements.

4.2 Collection and interpretation of software metrics

For several basic software measures, there are serious definition and measurement problems. They may depend an subjective views (e. g. self assessment, performance measurement) and the influences of a particular environment (differences of organization, tools and techniques, staff etc.). This applies in particular to the identification of early indicators for quality factors (e. g. reliability, usability). General experience, insights from case studies, and statistical evidence are applied to a new situation, which may be different with respect to important factors. Some important problemareas:

1. Comparability of attributes for measuring similar objects, e. g. size measures (lines of code, specification elements, pages, diagrams etc.) for texts in different specification and programming languages.
2. Completeness and accuracy of raw data, e. g. for defects or program failures (definition, counting of personal errors, flow of defect information) or for the accounting of resources (e. g. allocation of working hours)

3. Kind of the relationships between attribute measures (e. g. flow graph complexity) and quality factors (e. g. maintainability, reliability). A property of interest (e. g. maintainability) depends on many other attributes of a program (e. g. complexity of interfaces) and other influences (e. g. configuration management, documentation quality, staff availability).

4. Measurement of poeple (e. g. performance measurement). Controlled experiments and daily experience show major differences of personal performance indicators. However, task assignment for team members according to individual capabilities may be more helpful than emphasizing individual peformance differences.

Examples:

(1) A comparison of module complexity metrics [10] based on flow graphs reveals substantial differences. Different proposals for measuring „complexity“ are inconsistent even at the ordinal level.

(2) According to (Russel91), the fault detection rate (#faults/h) is independent from the inspection intensity (LOC/h) in a rather wide range (150 to 750 LOC/h). If this experience from a large project is generally valid, then reliability predictions based on the number of faults detected by inspections are rather meaningless.

(3) Portability is defined by the ratio porting_effort / development effort.
[8] defines a portability measure based on program attributes: *portability = (#statements - #data_base_accesses*8 - #TP_operations*8 - #file_accesses*4 - #module_calls*2) / #statements*
This may be a well designed and validated measure. It obviously excludes many influences affecting the effort for porting a program, whereas the definition relies on figures which are estimates rather than measures in the planning stage of a porting project.

Therefore, software metrics should be used as indicators stimulating in-depth consideration of features deviating from plans, requirements, or proved experience. ([4], p. 246) „Perhaps one of the greatest gaps in our knowledge, and a surprising one, concerns the relationship between the nature of the software development process and the characteristics, particularly the operational reliability, of the final product.“

4.3 The benefits of software metrics

In spite of these obvious problems, software metrics - carefully designed, measured and interpreted with respect to clear goals - provide information which is more precise and more reliable than other informations on the state of a project or the quality of a product.

- Software metrics are approximations to an objective description of software characteristics. For some important attributes (e. g. program size, run time) precise measurement rules are available. For quality characteristics, an approach including refinement, modeling, measurement of criteria, and calculation of index values may lead to an understandable and acceptable quantitative assessment.
- Definition of metrics presupposes careful modeling and definition of qualita criteria.
- Metrics focus attention; this may deviate attention from other unprecisely defined quality characteristics.

- Metrics are indispensable for testing and acceptance of a product. They allow clear statements on the fulfillment of requirements.
- Metrics may clarify quality requirements; this may lead to more realistic discussions on software quality.
- The definition of metrics which may be automatically collected is a prerequisite for the application of measurements to large programs.
- Application of some measurement procedures (e. g. for function points) require in-depth analysis of the underlying documents. This may lead to clarifications of requirments and identification of risks.
- Constructive actions aiming at the fulfillment of quantitative requirements may imply other improvements (e. g. completion of documentation, supplements to the test environment).

The purpose of a measurement program is a set of metrics related to a set of criteria which covers the important requirements and risk areas of a project. These metrics serve as a basis for project planning and control within an organization providing the infrastructure for measurement collection and analysis. Appropriate use of measurements will take into account the different views, interests, and capabilities of poeple and organizations involved.

An important application area of software metrics is the analysis of legacy software. According to [8], the computation of a large number of software measures (of a quality model) resulting in a program quality profile is an important step for reengineering decisions. Such profiles contain indicators of problem areas and information which supports estimates.

[3] gives an impressive example of the size of such reengineering problems. „NSA (National Security Agency, USA) spends many hundred millions of dollars annually on software development and maintenance. ... Pareto's law states that 20 percent of the code will contain 80 percent of the problems. Based on the 25 million lines of code formally analyzed to date, we have found that 10 - 15 percent of the code will have 70 - 80 percent of the problems. For NSA, Pareto's law is closer to some 13 percent of the code accounting for close to 90 percent of the problems, with some 2.5 percent of the total code accounting for close to 90 percent of the most critical showstopper and functional disconnect errors. This pathological code *must* be identified for risk analysis."

References

[1] Collins, W., Miller, K., Spielman,B., Wherry, P.: *How Good is Good Enough*? An Ethical *Analysis of Software Construction and Use*. Communications of the ACM, Vol. 37.1, Jan. 1994, p. 81 - 91

[2] Conte, S., Dunsmore, H., Shen, V.: *Software Engineering Metrics and Models*. Benjamin/Cummings, Menlo Park, 1986, 396 S.

[3] Drake, T.: *Measuring Software Quality: A Case Study*. Computer, Vol. 29.11, Nov. 1996, S. 78 - 87

[4] Fenton, N.: *Software Metrics - A Rigorous Approach*. Chapman & Hall, London, 1991, 337 S.

[5] Heitger, Marian (Ed.): *Verantwortung, Wissenschaft, Forschung. Festgabe zum 20jährigen Bestehen des Internationalen Forschungszentrums in Salzburg*. Herder-Verlag, Freiburg, 1981

[6] Rombach, H. D.: *Design Measurement: Some Lessons Learned*. IEEE Software, Vol. 7.2, March 1990, S. 17- 25

[7] Russell, G.: *Experience with Inspection in Ultralarge-Scale Developments*. IEEE Software, January 1991

[8] Sneed, H., Rothardt, G.: *Softwaremessung*. Wirtschaftsinformatik, Bd. 38.2, April 1996 und SES-Arbeitspapier (Juli 1997)

[9] Weingartner, Paul: *Normative Prinzipien und Grenzen rationaler Kommunikation*. In [5] S. 33 - 58

[10] Zuse, H.: *Software Complexity - Measures and Methods*. DeGruyter Publisher, Berlin, 1991

Metrics of Object-Oriented Software Development Technologies MOSET

Andreas Schmietendorf, Berlin Development Center, Deutsche Telekom AG

1 Initial Situation in the Berlin Development Center

Object-oriented software systems are playing an increasingly important role, particularly in the field of telecommunication applications. The Berlin Development Center also develops application systems for the Deutsche Telekom TMN (Telecommunication Management Network) platform and applications in the field of IN (Intelligent Network). Both types of applications are already being developed using object-oriented technologies such as object-oriented modeling in the analysis and design phase, or implementation with, for example, C++ or Smalltalk. The ISO-9000 accredited Deutsche Telekom development centers recognized early on the importance of using metrics. Since it was set up in 1992, the Berlin Development Center has determined an increasing number of metrics such as function points, quantities such as LOCs, the on-schedule implementation of applications, the percentage of project management tasks, error statistics, user satisfaction, and business management key data within the framework of the development projects, and has maintained associated empirical databases. This is reflected in the current CMM (Capability Maturity Model) level of 3.4 . Above all, these metrics provide Deutsche Telekom management with a practical control instrument. Since metrics have, until now, been used predominantly in the field of structured development, and are to be attributed more to the classical environment, it has been necessary to find suitable metrics for optimizing the software creation process and also for developments that had been carried out on an object-oriented basis.

2 Objectives and Subject Matter of the MOSET Project

The MOSET (Metrics of Object-Oriented Software Development Technologies) research project is intended to provide a starting point for finding suitable measurement variables which are actually capable of providing information for software developments that have been carried out on an object-oriented basis, and which are cost-effective. In order to learn about the use of metrics in object-oriented software technologies, the best option seemed to be a project which used a small prototype to investigate all the software development phases. The various project task groups were to involve project management, quality assurance and product administration, as well as the actual development of the software from analysis through to implementation. In brief, the most important objectives are as follows:

- Understanding a software development process that has been carried out on an object-oriented basis, but not including the introduction and servicing of the software product.
- Application of tools for software development which are also used by the Berlin Development Center for implementing customer projects.
- Use of peripheral tools such as a version administration and a tool for generating documents from the object model.

- Evaluating metrics programs which can then also be transferred to the subsequent software production process at the Berlin Development Center.
- Using the Intranet technology of the Berlin Development Center for providing results to enable a rapid exchange of information and to stimulate discussion about the results.
- Drawing up a tool-based project plan which defines milestones and contains corresponding reviews to safeguard the progress of the project.
- Dividing the project into clear implementation units, with modeling and integration carried out together.

Due to the limited implementation time, a collective brainstorming session to establish the type of application to be created helped to boost the motivation of the project team. From several ideas, a database-supported media administration system (2-level client/server architecture) was chosen for the consultancy department of the Berlin Development Center. This records all media of the field, orders new media, and enables employees of the consultancy department to borrow established media. The framework conditions were defined as: use of an RDBMS based on the SQL server (Windows NT), the executability of the client application under Windows NT/95, and use of the MS help system. Development was carried out using Rational Rose for object modeling (OOA/OOD) and MS Visual C++ for coding (OOP). WinWord was used in conjunction with SoDA to create the documentation for partial generation from the object model.

3 Estimated Expenditure for the MOSET Project

The expenditure should be estimated at the beginning of each project. This is the only way of determining the implementability, the risks, the necessary resources, and the implementation date, and, of course, of making a calculation for a quotation. The Albrechts metric is used to estimate the expenditure at the beginning of the project, whereby values are determined both in accordance with the classical stipulations (IBM 1979) and also based on the new procedure introduced by Deutsche Telekom in accordance with IFPUG 4.0 (International Function Point User Group). This latter procedure takes into account more recent software technology such as, for example, graphical interfaces, or distinguishing between applications to be newly developed and expansion projects, but it does not take into account object-oriented software development. Both procedures yield values which clearly exceed the possibilities of the MOSET project, whereby the latter procedure produced overall values that were slightly lower. Since the timeframe of the project was predefined as approx. 6.5 PM, we were able to carry out a back calculation based on an IBM function point curve which resulted in approx. 100 to 150 function points. There is of course the question as to whether the values determined are applicable under the conditions of an object-oriented software development carried out with the aid of class libraries and code generators. When this report was written, approx. 60 % of the overall application had been implemented, which already casts doubt on the function point values determined.

In accordance with [1], I believe it is necessary to introduce a correction factor which takes into account the current status of the technology. In the case of object-oriented technology, in my opinion, the code to be modified and the use of class libraries, which both feature in a typical object-oriented development, should be taken into consideration. The generated code,

and the code frame generated from the object model, or else the use of the class or application wizard under Visual C++, should also be taken into consideration.

The MOSET project was not intended to question the way in which function points are determined, but merely to observe, from a critical point of view, their applicability in the current form available for object-oriented software developments. Once the project is finished, a follow-up calculation will be made in cooperation with the University of Magdeburg in order to obtain an initial value for our own function point curve in relation to the object-oriented development technology used.

In order to realistically assess the productivity of the project team, and to be able to transfer the project results to other applications, the relevant initial ”know how” was gathered in the form of a questionnaire. This information, gathered on a voluntary basis, was recorded and structured as follows:

- General experience of software development and knowledge of object-oriented modeling, implementation and relational database systems.
- Tool-specific experience such as working with the modeling tool Rational Rose, use of Visual C++, or the administration and configuration of an MS SQL server.
- Project-related experience such as project management, using a configuration management system, or working on projects in a team.

To keep a log of the dynamic course of the project as regards the time required for each problem definition, these values were recorded daily by every project member. This means that in the evaluation, for example, the expenditure ratio in the analysis, design and implementation phases can be established.

4 Configuration Management and Determining Metrics

Nowadays, the commercial development of applications would be inconceivable without a tool for version administration/configuration management (CM tool). This type of system supports the consistent holding of files jointly processed during the course of the software development project, whereby the type of file used for the setting and administration functions in the CM tool is less important. Another aspect of using a configuration management is that it supports the software error handling process.

In the MOSET project, the system ClearCase by Pure Atria was used as the CM tool. The following illustration shows the dialogs for the version tree of an actual file and the text comments for the particular version which can be specified when checking in or out.

ClearCase does not offer direct support for recording metrics, but it does however record, in values, the total number of modifications made to a file, and the number of differences following a comparison of the two versions within the version tree. Additional values for error statistics such as, for example, the distinction drawn between versions produced as part of error processing during acceptance and during normal operation these values must be counted out given the current status of the product.

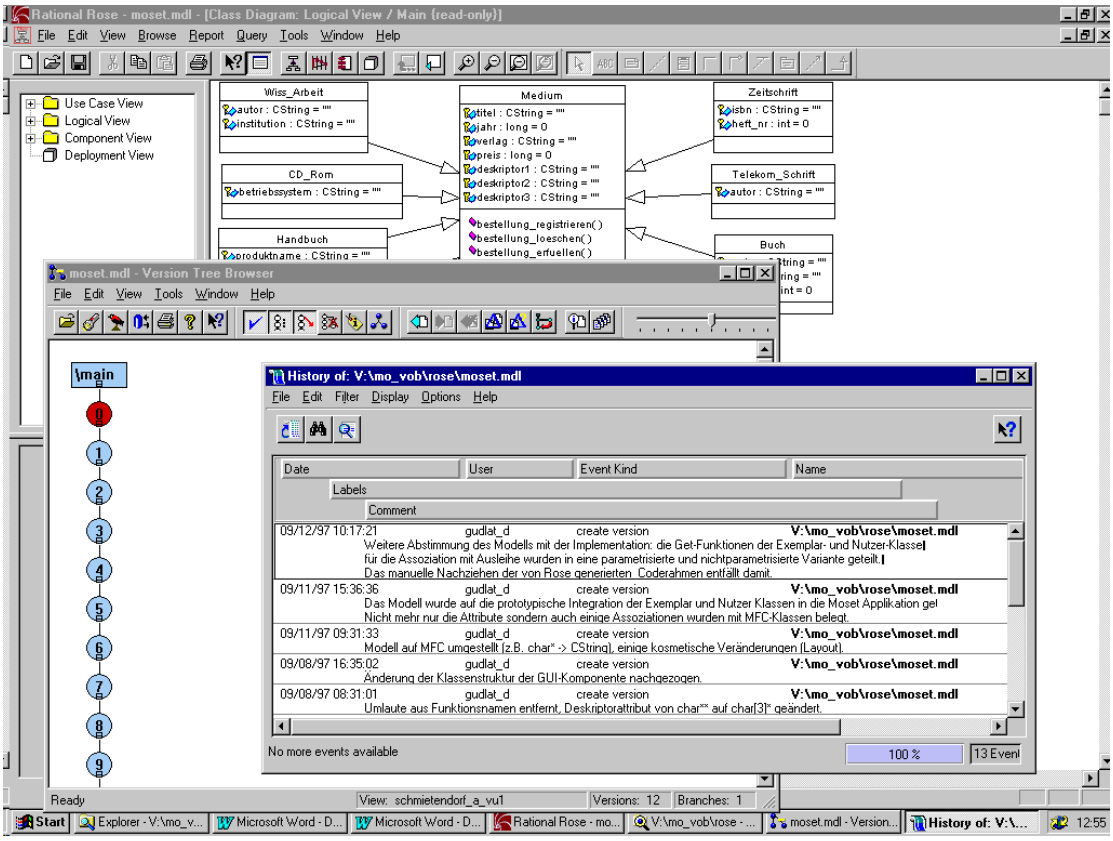


Illustration 1: Version check of a processed file

Booch [2] considers the speed with which classes change to be a good yardstick for measuring the stability of a software project. He evaluates an initial instability as normal, and considers the gradual increase in stability as a good sign for a successful software development. The elimination of a complete class tree in the final stages, however, is seen as a problem. If the CM tool is to support the recording of metrics related to this, a corresponding source file/header file must be created for each class used.

Developers who lack project experience, as is the case with the MOSET project, often do not immediately accept a CM tool system. Its use as regards the file system is transparent, however, an overhead when checking in and out of each individual file, which is necessary when the access changes can not be avoided. It can also make processing tasks more difficult because the same files are required. The less the developers see themselves hindered by using the CM tool, the more advisable it is to segment the problems of software development, since it should be possible to process them independently of each other to a great extent.

In conclusion, I feel that the following metrics may be able to be determined in conjunction with a configuration management system:

- Modification statistics (modifications/time unit) for classes of the object model,
- Recording error statistics separately in acceptance and operation,

- Recording the extent of the modifications between the different version statuses,
- A metric for the sensible segmenting tasks that are largely independent of each other.

5 The Requirements of Metrics Programs

The programs used to record metrics are selected according to the following criteria:

- The measurement tool should be integrated as far as possible into the tool environment already being used.
- Only minimal expenditure should be incurred by using the tool to determine the metrics.
- Notations chosen previously (Booch or UML) for the object-oriented analysis and design phase should be reflected in the metrics extracted.
- Time-controlled automatic recording of metrics by instrumenting the configuration management used from the processed files.
- The selection of a measuring tool should also heed the question of servicing and supporting new versions of the development environment used.
- A measuring tool should also provide empirical values for the metrics used as an initial value in order to offer support for an interpretation of the results right from the very beginning.
- Supporting an accumulation of the recorded metrics in an empirical database to be able to use real experience for the evaluation.
- Considering the improvements achieved in the software product by recording and evaluating metrics.

6 Metrics Programs for the Analysis and Design Phase

Object-oriented analysis and design involved the use of the tool Metrics ONE (Alpha Version 1.1) and a Rational Rose script ”Martin Metrics” to record metrics from a Rational Rose model (Version 4.0). Hereit is appropriate to examine the details of the first tool named, because it currently offers the most comprehensive approach for a Rational Rose object model as regards the metrics determined.

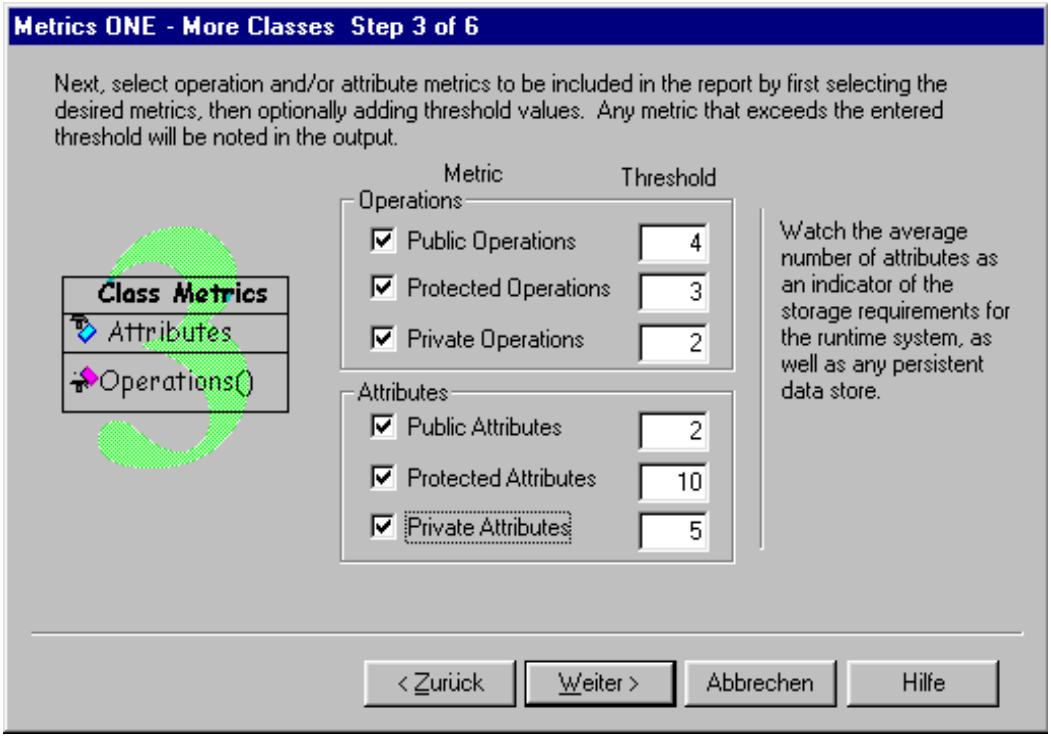


Illustration 2: Dialog for recording metric classes

The inquiry as to which metrics from the OO model are to be generated takes the form of 5 consecutive dialogs. It is also possible to establish threshold values for each metric, and if they are exceeded, a special note of this is made in the Excel tables which are created after the tool has been executed. These threshold values are retained when the tool is used again, but it is not possible to store or load them from an empirical database. For example, a threshold value relating to ”multiple inheritance” can be set to ”1” if this cannot be done in the subsequent implementation, as is the case, for example, under Java. Classes with more than one “super class” are marked as such accordingly in the Excel table.

In the version available for the test, the metrics recorded refer to the following diagrams in accordance with the UML notation:

- **Class diagram** with the metrics for stereotypes, persistent classes, abstract classes, inheritance levels, the parent or child classes of a class, dependencies on and to other classes, public, protected or private operations or attributes,...

- **Use case diagram** with metrics for abstract use cases, the relation to scenario diagrams, class diagrams, superordinate parents, subordinate children, dependencies of the uses cases on the actor,...

- **Component diagram** with metrics for public classes, implementation classes, sub-components, generality, instability, afferent and efferent coupling.

An interpretation of the results is offered for the individual metrics as part of help. The other types of diagram in accordance with the UML notation are not used. In my opinion, at least metrics from the sequence diagram would be desirable here.

7 Metrics Programs for the Implementation

The Resource Standard Metrics (RSM) tool by M Squared Technologies, available under MS DOS, Windows NT and UNIX, offers extensive possibilities for determining metrics related to C or C++ source code.

```
                Resource Standard Metrics For C & C++
  Version 2.50  (C) 1997 M Squared Technologies   Sat Sep  6 20:32:56
1997
  License Type: Shareware Evaluation License

_____________________________________________________________________
_____________

  File: NutzerBulk.cpp
  Date: Wed Aug 27 12:50:28 1997                File Size: 3867 Bytes
  ---------------------------------------------------------------------

                    ~~ LOC, Keywords and Metrics ~~

  ---------- LoC ----------  case .............:          3
  Lines of Code (LoC)...:       94  break ..............:          3
  Lines of just { or } .:       16  if .................:          9
  Lines of just ( or ) .:        0  else ...............:          7
  Effective LoC (eLoC) .:       78  goto ...............:          0
  ---------- Lines ----------  return .............:          1
  Blank Lines ..........:       20  exit() _exit() abort():    0,  0, 0
  Comment Lines ........:       19  struct, union ........:    0,     0
  Total Logical Lines ..:      133  class, typedef .......:    0,     0
  Total Physical Lines .:      129  template, friend .....:    0,     0
  ---------- Key Words ----------  ---------- Analysis ----------
  Code Statements ; ....:       40  #preproc, Macros .....:    8,     0
  #include .............:        4  Paren Count (,) ......:   56,    56
  #define .............:        1  Brace Count {,} ......:    9,     9
  const ................:        0  Bracket Count [,] ....:   14,    14
  do, while ............:    0,  0  Chars/Line, Notices ..:   30,    20
  for ..................:        1  Code, eCode Lines ....: 70.7%, 58.6%
  switch ...............:        1  Comment, Blank Lines .: 14.3%, 15.0%
  default ..............:        1  Characters, Spaces ...: 96.3%,  3.7%
  ---------------------------------------------------------------------
                    End of File: NutzerBulk.cpp
```

Illustration 3: Metrics of a C++ source file (rsm -v NutzerBulk.cpp > NutzerBulk.txt)

The source files to be investigated are specified once the RSM command and the corresponding options, which identify the types of metrics to be determined, and an ASCII file for recording the metrics on the relevant command line, have been specified.
The following is a short list of what I see as the most interesting options of the tool's variety of possibilities.

- **rsm -v,** recording the most diverse code metrics (LOCs, key words,..)
- **rsm -a,** metrics relating to the allocation/deallocation of memory

- **rsm -b,** benchmark, resource metrics when executing the RSM application
- **rsm -c,** cyclomatic complexity according to the definition of McCabe
- **rsm -i,** recording the C++ class definitions available in the source text

Unfortunately, concrete metrics in relation to object-oriented attributes of the source code are not yet sufficiently supported. The MOSET project therefore had to rely on another tool which was used to gather metrics such as the inheritance structure or methods/attribute statistics.

8 Considering the Performance of the Application

The following should show a metric which can be gathered during the operation of an application, and its use must be considered within the software development. Company-critical client/server applications are often operated for the customers by operating companies. Not least as a result of this fact, a quality agreement is required between the operator and the user of the software application, as is a method of accounting the actual computer power used. One way of ensuring that the quality of the performance is monitored as required is to use the API preprogrammed triggers in the application via ARM - Application Response Measurement.

The objective of application operation should be the preventive monitoring of the performance provided by application systems. If the user notices bad performance and uses it to measure implicitly, it is actually too late. The ARM API heavily advocated by the Computer Measurement Group, and implemented by companies such as HP, Sun, NCR and IBM since the end of 1996, permits instrumentation of the application for the response times to be monitored in relation to defined transactions. To this end, the business transactions to be surveyed (not to be confused with DB transactions) must be defined within the software development. In my opinion, these requirements can be recorded in the OOA/D phase as a time condition in the sequence diagram in accordance with the UML notation.

To identify a transaction which is to be monitored, the functions “arm_getid” and “arm_start”, for identifying the beginning of a transaction, and “arm_stop”, for signaling the end of a transaction, are used. This is equivalent to the transaction brackets used with relational database systems. If these brackets are used in database systems to ensure that a database transaction is executed correctly, the ARM “brackets” are used to measure the response time in connection with a measuring tool such as HP MeasureWare. The measuring tool collects the performance data for the transactions instrumented in this way and can, for example, introduce a warning if response time requirements are not met. These demands must firstly be incorporated in the database of the measuring tool in the form of thresholds to be defined, whereby the data from the UML sequence diagram should be used.

9 Practical Experiences using a Prototype

In accordance with the experiment already performed by Capers Jones [5] to determine “Which tools increase productivity ”, a similar investigation was carried out as part of the MOSET project during a 3-week introductory phase. We evaluated productivity in relation to consistent software development from analysis right through to implementation under the conditions of using Rational Rose (OOA/OOD) and MS Visual C++ (OOP). The problem was

to develop an interface prototype for Windows 95/NT which has only one main window with pull down menus for selecting 2 dialogs. The dialogs were to store some elements such as, for example, pushbuttons, edit boxes and list boxes with very few functionalities such as, for example, a message box (standard Windows output dialog).

For the solution to be successful, it was imperative that the same code frame generated in the design phase from the object model under Rational Rose could be used within the programming. Class definitions and derived objects for all components of the model had to be visible both in the Rose model (firstly), and then in the Visual C++ source text, i.e. meeting the requirement of consistency between object model and source code. These demands permitted neither use of the Visual C++ code generators (application and class wizard), nor use of the MFC class library, because Rational Rose is currently not capable of representing the classes generated in this way in a suitable fashion for further processing. (At the time this study was written, Microsoft had already announced the availability of the tool Visual Modeller in a ß version, which is to offer this support under the application of Rational Rose.)

Summary of some of the more important results and conclusions:

- The implementation time was approx. 0.88 PM, during which period an object model (class model and sequence diagrams), the source code (approx. 5000 LOCs) and program documentation (18 pages) were created. The absolute value of the LOCs, Lines of Code, (all code lines) can not be transferred to other software developments because many software product components were omitted here intentionally (e.g. test documentation, user documentation,...) which would otherwise have caused the LOC value to be considerably lower.

- In comparison to similar problems, the implementation time is very good. This can easily be substantiated by doing away with class libraries and code generators. On the other hand, the availability of a consistent model for the subsequent servicing and maintenance of a software product is a very important factor.

- For consistent software development, attributes of the subsequent implementation tool (e.g. class libraries) must also be reflected in the modeling. Only then can one speak of a constant development environment, otherwise changes in media result in a lack of efficiency and lower quality.

- By using the CAME tool described above, it was also easy for someone not involved in the project to check the consistency between the object model and the implementation quite easily. On several occasions, by comparing, for example, the classes implemented with the model classes, deviations were discovered and corrective action was taken.

10 Conclusion

As regards the metrics to be recorded, the project was oriented towards the [3] classification in relation to process metrics, product metrics and resource metrics. It was clear that in a project of short duration, with a small number of employees, and a relatively unrelated problem the absolute values of the recorded process metrics can not be easily transferred to other projects, or that some measurements, for example, maturity metrics, can not be made effectively at all. Most of the metrics gathered therefore referred to product metrics and resource metrics simply because the corresponding measuring tools were available.

The relatively low incidence of observing the software measurement by the manufacturer of the development tool was surprising. Particularly with regard to our requirements (see point 5), there was no question of tool selection. However, the integrated Visual Basic Script interface of the Rational Rose modeling tool is very positive and supports, amongst other things, the creation of individual tools for gathering metrics from the model.

The use of a configuration management tool, which could be used to take subsequent product measurements on the different version statuses, proved to be highly significant.

The following are some of the metrics that were suggested during the course of the project:

- Due to the difficulty of carrying out consistent software development from analysis through to implementation, I feel it is necessary that deviations are recorded in the form of metrics. For example, comparing the objects, attributes and methods of the object model and implementation could lead to a percentage ratio which reflects the degree of “consistency”.

- As part of the project, the overall application was segmented into small, clear units which were integrated in the subsequent overall application. It would be advisable to have increased support in the form of metrics that could show which segmentation granularity is more expedient or what may lead to additional expenditure.

References

[1] Behrens, C.A.: *Measuring the Productivity of Computer Systems Development Activities with Function Points*. IEEE Transactions on Software Engineering, 1993

[2] Booch, G.: *Qualitäsmaße* - Fachthema der OBJEKTspektrum 4/94 Seite 53. SIGS Conferences GmbH, München, 1994

[3] Dumke R.: *Softwareentwicklung nach Maß*. Friedr. Vieweg & Sohn Verlagsgesellschaft, Braunschweig/Wiesbaden:1992

[4] Dumke R.; Foltin E.; Koeppe R.; Winkler A.: *Softwarequalität durch Meßtools*. Friedr. Vieweg & Sohn Verlagsgesellschaft, Braunschweig /Wiesbaden:1996

[5] Jones, C.: *Assessment and Control of Software Risks*. Yourdan Press, New Jersey, 1994

Quality Assessment of Object-Oriented Software Development Methods

Reiner R. Dumke, Erik Foltin
University of Magdeburg, Faculty of Informatics

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.

1 Introduction

The benefits of the use of the object-oriented software development techniques are widely discussed in many papers ([12], [44], [47], [49], [70] etc.). However, most of these discussions and presentations only enumerate the features of the OO development methods and programming environments, e. g. in [34] as

Feature Name	*OOSA(Embly et al.)*	*OMT (Rum-baugh 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.						

The presentation by Khan et al. [52] gives 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 [26].

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 [24] 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 [45]). 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 [35])

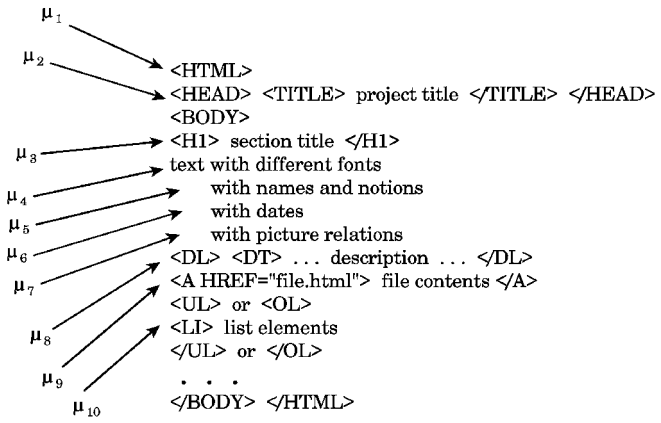
- the **process measurement** for understanding, evaluation and improvement of the development method,
- the **product measurement** for the quantification of the product (quality) characteristics and validation these measures,
- the **resource measurement** for the evaluation of the supports (CASE tools, 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 [21]). 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.

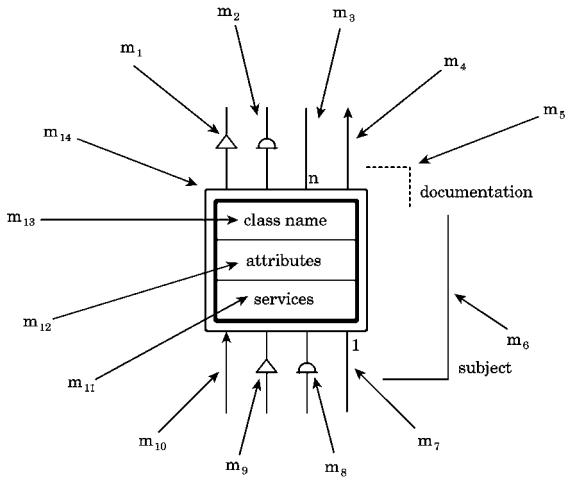


The SMLAB problem definition must be accessible to all members of the software engineering team and the document itself is an essential source for many outputs such as milestones or an overview for some administrational purposes. Therefore, we decided for a html file set of the World-Wide Web Intranet 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 [38]. A more detailed list of life cycle metrics types is given in the following (see also [24]).

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 [73] 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).

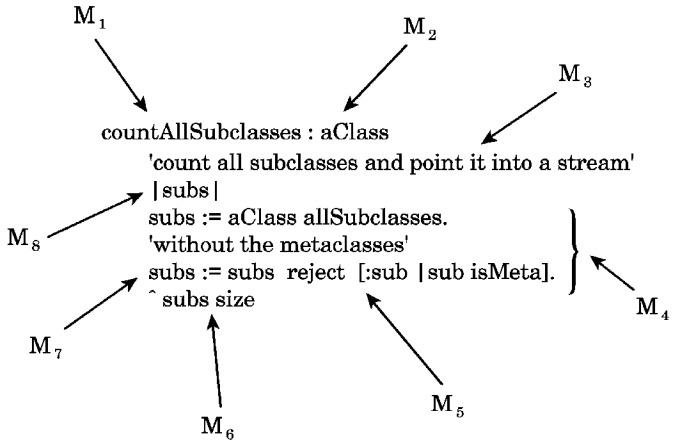


The OOD step ensures a full continuity with the OOA step. It extents (or updates) the OOA model with respect to the chosen implementation environment, i. e. by including libraries for the realization of the user interface or data storage engines. The resulting OOD model is the primary model used later 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 [68]. In general it is necessary to quantify the management activities based on the following metrics [24].

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 availa-bility
    - 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 accep-tance 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 [56] 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 [24].

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 [46], 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[24].

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 [24].

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

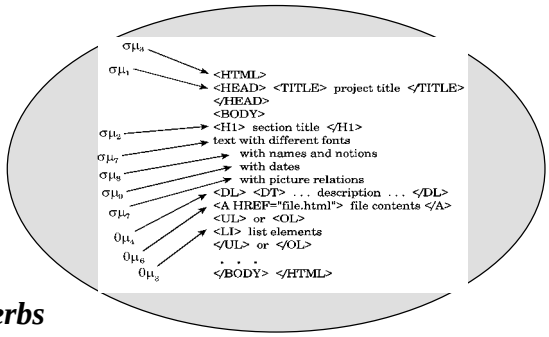
2.5 Conclusions

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 [29].

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

Problem definition (PD)
(as HTML document system):

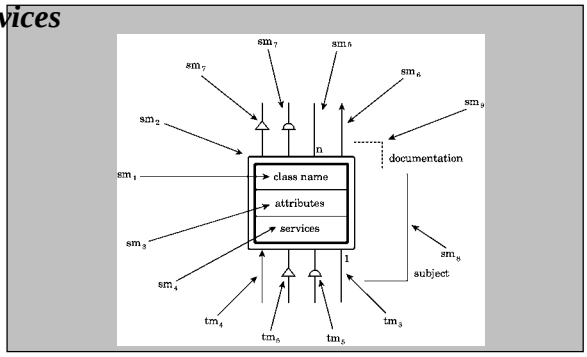
verbal text



notions adjectives verbs

specif. classes attributes services

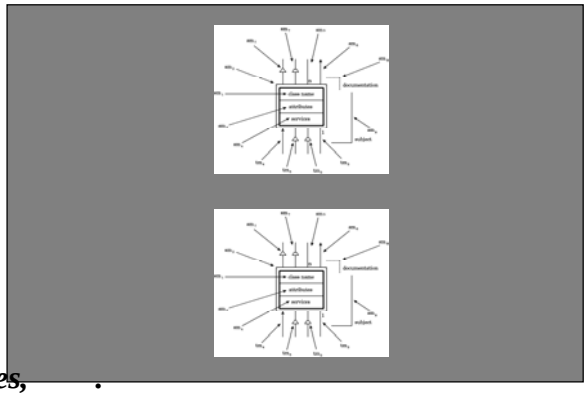
OOA model in the Coad/Yourdon approach
(drawing element):



designed classes, attributes, services

organiz. cl., attr., serv.

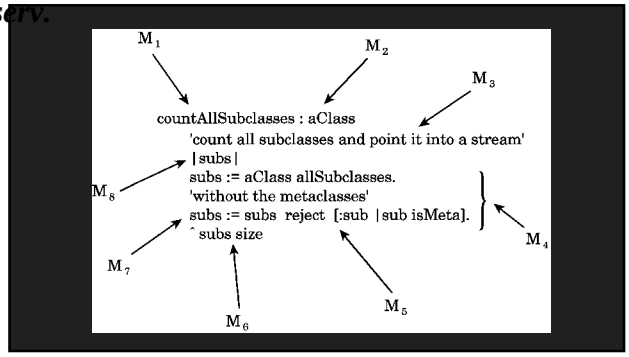
OOD model in the same approach
(the same drawing element):



impl. classes, attr., serv.

reused cl.a.s. new cl. attr. serv.

Implementation in Smalltalk
(a class method):



PD/OOA specification indicators

OOA/OOD design indicators

OOD/OOP implementation indicators

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 [77]; of a C++ communication system by Szabo and Khoshgoftaar [53]; or for different metrics and different C++ libraries and Eiffel programs by Abreu and Melo [3],

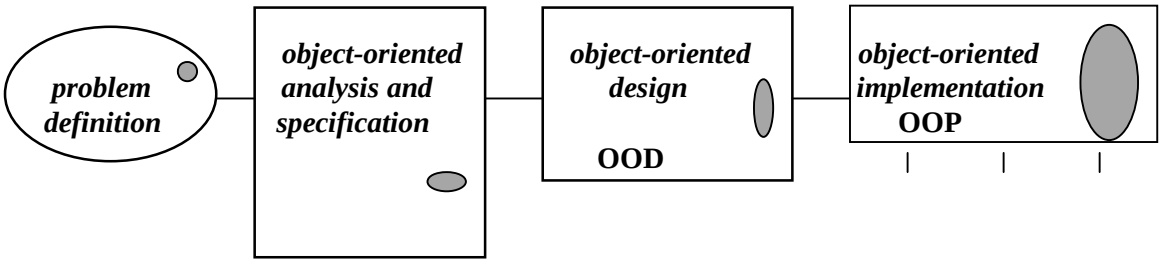
- ***metrics set definitions*** by Abreu and Carapuca in [1] for C++ with the two vectors category (design, size, complexity, reuse, productivity, and quality), and granularity (system, class, and method); by Binder in [9] as a set of C++ metrics to measure encapsulation, inheritance, polymorphism, and complexity; or by Arora et al. in [5] 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 [66] as a metrics set that can be used for the C++ language and Smalltalk,

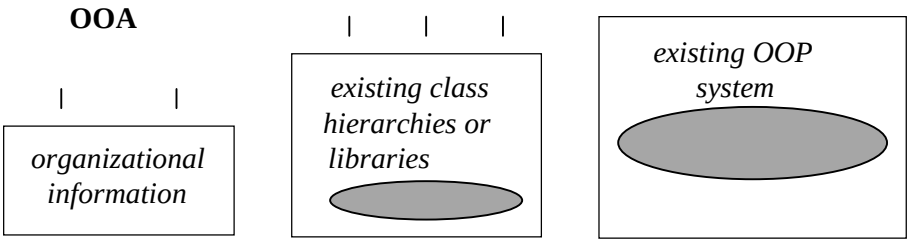
- ***OO aspect measurement*** by Ott et al. in [7] or by Lee et al. in [61] or by Hitz and Montazeri in [44] or by Han et al. in [40] of class coupling and cohesion; or by Bieman in [58], John in [50], and Pant et al. in [72] to measure reusability, or by Chung et al. [18] to measure the inheritance complexity, or to support object-oriented testing (Chung and Lee in [19]) and maintenance (Lejter in [63]),

- ***information theoretical approaches*** like the measure of conceptual entropy by Dvorak in [31] or the cognitive approach by Henderson-Sellers et al. in [43] with the landscape idea along the method routes or the learnability aspects in the use of class libraries in [62], and

- ***validation of enclosed approaches*** by Chidamber and Kemerer in [17] 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 [65], the (algebraic) analysis approach of Churcher and Shepperd in [20], and the investigations of Zuse in [89] and [90].

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





3.2 Metrics for OO Systems

For a narrowly-focused presentation of the existing OO metrics we use our general metrics classification [24] as

PROCESS METRICS	PRODUCT METRICS	RESOURCES METRICS
Maturity Metrics - organization metrics - resources, personnel and training metrics - technology management metrics - documented standards metrics - process controlling metrics - data management and analysis ***Management Metrics*** - milestone metrics - risks metrics - workflow metrics - controlling metrics - management data base metrics - quality management metrics - configuration management m. ***Life Cycle Metrics*** - problem definition metrics - requirement analysis and specification metrics - design metrics - implementation metrics - maintenance metrics	***Size Metrics*** - elements counting - development size metrics - size of components metrics ***Architecture Metrics*** - components metrics - architecture characteristics - architecture standards metrics ***Structure Metrics*** - component characteristics - structure characteristics - psychological rules metrics ***Quality Metrics*** - functionality metrics - reliability metrics - usability metrics - efficiency metrics - maintainability metrics - portability metrics ***Complexity Metrics*** - computational complexity metrics - psychological complexity metrics	***Personnel Metrics*** - programmer experience metrics - communication level metrics - productivity metrics - team structure metrics - ***Software Metrics*** - performance metrics - paradigm metrics - replacement metrics ***Hardware Metrics*** - performance metrics - reliability metrics - availability 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 [66])*
- change dependency between classes (CDBC) *(transparency principle [44])*
- cognitive complexity (CCM) *(case study based [14])*
- time to fix the known errors (TKE) in minutes *(minimizing principle [41])*

Process life cycle metrics: (10)

- conceptual specificity (OOCM) *(difference principle [31])*
- conceptual consistency (OOCM) *(difference principle [31])*
- conceptual distancy (OOCM) *(difference principle [31])*
- number of scenario scripts (NSS) *(transparency principle [66])*
- unit repeated inheritance (URI) testing *(test coverage Cn, n>2 [Church 94])*
- number of methods overridden (NMO) *(transparency principle [66])*
- number of methods inherited (NMI) *(transparency principle [66])*
- number of methods added (NMA) *(transparency principle [66])*
- number of modifications requests (MR) *(minimizing principle [41])*
- time to implement modifications (TMR) *(minimizing principle [41])*

Product size metrics: (17)

- number of abstract classes [27]
- number of object/classes [27]
- total number of (class/instance) attributes (NIV, NCV [66])
- total number of (class/instance) services/ methods (NOM, [65]; NIM,NCM [66]) *(Smalltalk$_{initial}$ =22*#classes [60])*
- number of object connections [27]
- number of message connections [27]
- number of the subclasses [27]
- number of the subject domains [27]
- code/text lines of method [27]
- length of attribute name [24]
- number of ADTs defined in a class (DAC) *(transparency principle [65])*
- number of semicolons in a class (SIZE1) *(case study [65])*
- number of attributes + number of local methods (SIZE2) *(case study [65])*
- number of root classes *(case study = 3 [59])*
- number of key classes (NCK) *(completeness principle [66])*
- number of support classes (NSC) *(completeness principle [66])*
- number of subsystems (NOS) *(transparency principle [66])*

Product architecture metrics: (2)

- verbatim reuse (VR) *(optimization principle [8])*
- generic reuse (GR) *(optimization principle [58])*

Product structure metrics: (22)

- average number of attributes per class [27]
- average number of services per class *(not more than 20 [66])*
- average number of object connections per class [27]
- average number of message connections per class [27]
- maximal depth of the inheritance (DIF) *(applica-tion$_{initial}$ 3 [17])*
- method hiding factor (MHF) *(initial 19,6 % [2])*
- attribute hiding factor (AHF) *(initial 79,7 % [2])*
- method inheritance factor (MIF) *(initial 73,5 % [2])*
- attribute inheritance factor (AIF) *(initial 56,2 % [2])*
- polymorphism factor (POF) *(initial 6,5 % [2])*
- coupling factor (COF) *(initial 10,8 % [2])*
- number of children (NOC) *(initial 0.9 [16])*
- coupling between object classes (CBO) *(application$_{initial}$ 1.3 [16])*
- response for a class (RFC) *(initial 10 [16])*
- lack of cohesion (LCOM) *(initial 4.1 [16])*

- average code/text lines of methods *(Smalltalk/$V_{initial}$ = 3 [87], Smalltalk=8, C++=24 [66])*
- strong functional cohesion (SFC) *($example_{demo}$ 0.18 [7])*
- I-based coupling (ICP) *($example_{demo}$ [61])*
- I-based cohesion (ICH) *($example_{demo}$ [61])*
- strength of cohesion as part of operations that apply one ADT domain *(case study in C++: 26% [40])*
- 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% [40])*
- locality of data (LD) *(transparency principle [44])*
- computing cohesion (CH) *(maximum = 1 [Wech 96])*

Product quality metrics: (6)

- understandability (= average number of attributes per class, average LOC per method) *(maximum reducing [6])*
- average length of classes/attributes/methods names *(general mnemonic aspects)*
  - test order for class firewall (CFW) *(case study: 192 stubs per test order [57])*
  - number of known errors (KE) during testing *(minimizing principle [41])*
  - percentage of commented methods (PCM) *(transparency principle [66])*
  - problem reports per class (PRC) *(empirical criteria [66])*

Product complexity metrics: (8)

- weighted method per class (WMC) *(initial 10 [17])*
- weighted attribute per class (WAC) *(method evaluation case study [79])*
- leveraged reuse (LR) *(optimization principle [8])*
- subjective assessment of complexity (SC) *(ordinal: 1...5 [41])*
- message passing coupling (MPC) *(transpa-rency principle [64])*
- number of tramps (NOT) *(method evaluation case study [79])*
- operation complexity (OC) *(case study = 78.5 [15])*
- attribute complexity (AC) *(case study = 2.2 [15])*

Resource personnel metrics: (1)

- classes per developer (CPD) *(empirical criteria [66])*

Resource software metrics: (2)

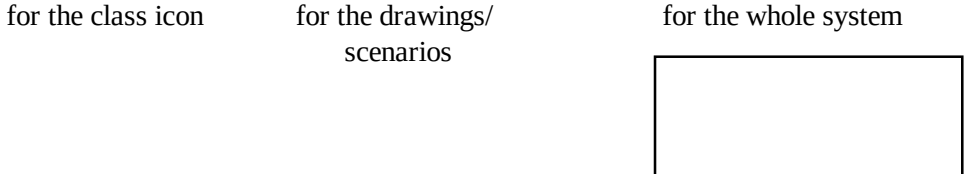
- paradigm related development time *(case study: OO vs. procedural [62])*
- violations of the law of demeter (VOD) *(method evaluation case study [79])*

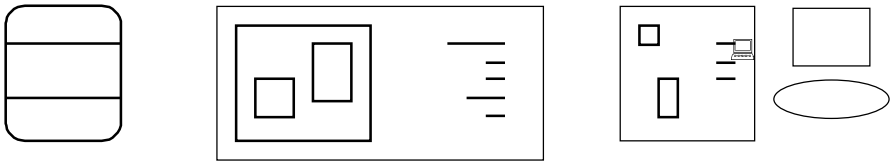
Total number of OO metrics: 72

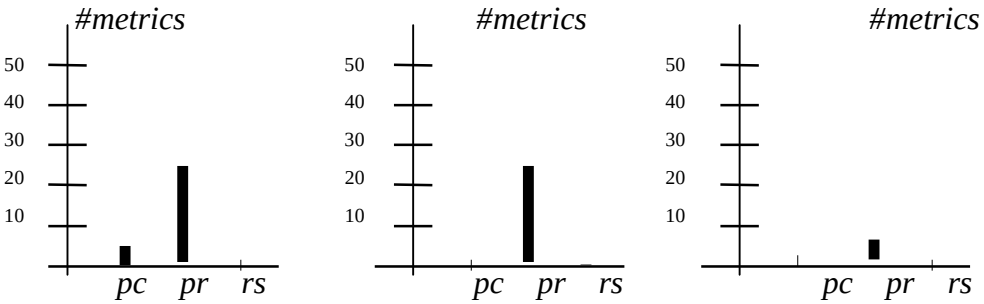
3.3 Conclusions

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.

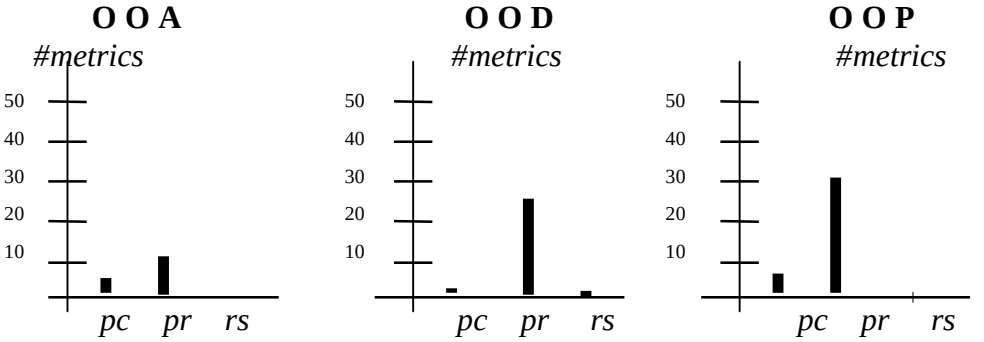
System Model Granularity



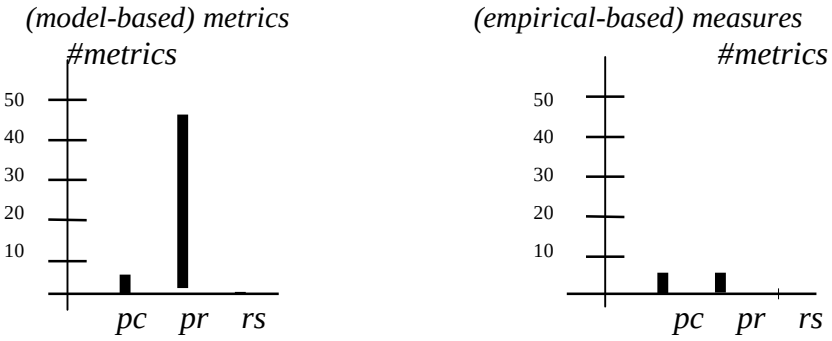




Life Cycle Phase Related



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 × #classes) or ***tuple form*** (understandability = (average #attributes, average LOC_{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 [75]: ''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.

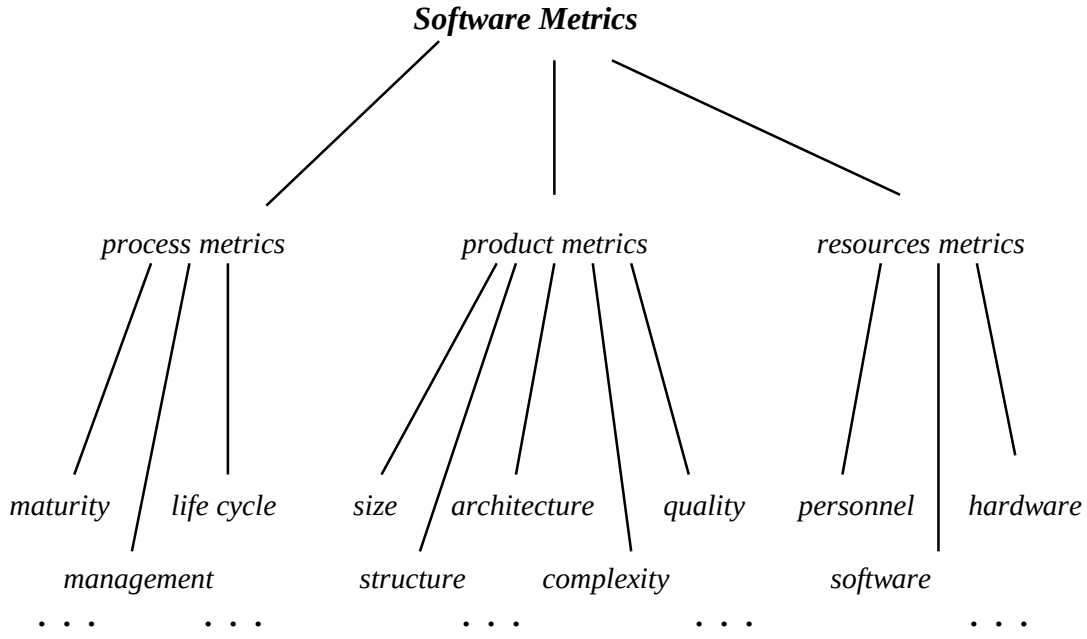
4 A General Object-Oriented Measurement and Evaluation Framework

We define a general software measurement framework with the following components (see also [24], [29], [28]):

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*.

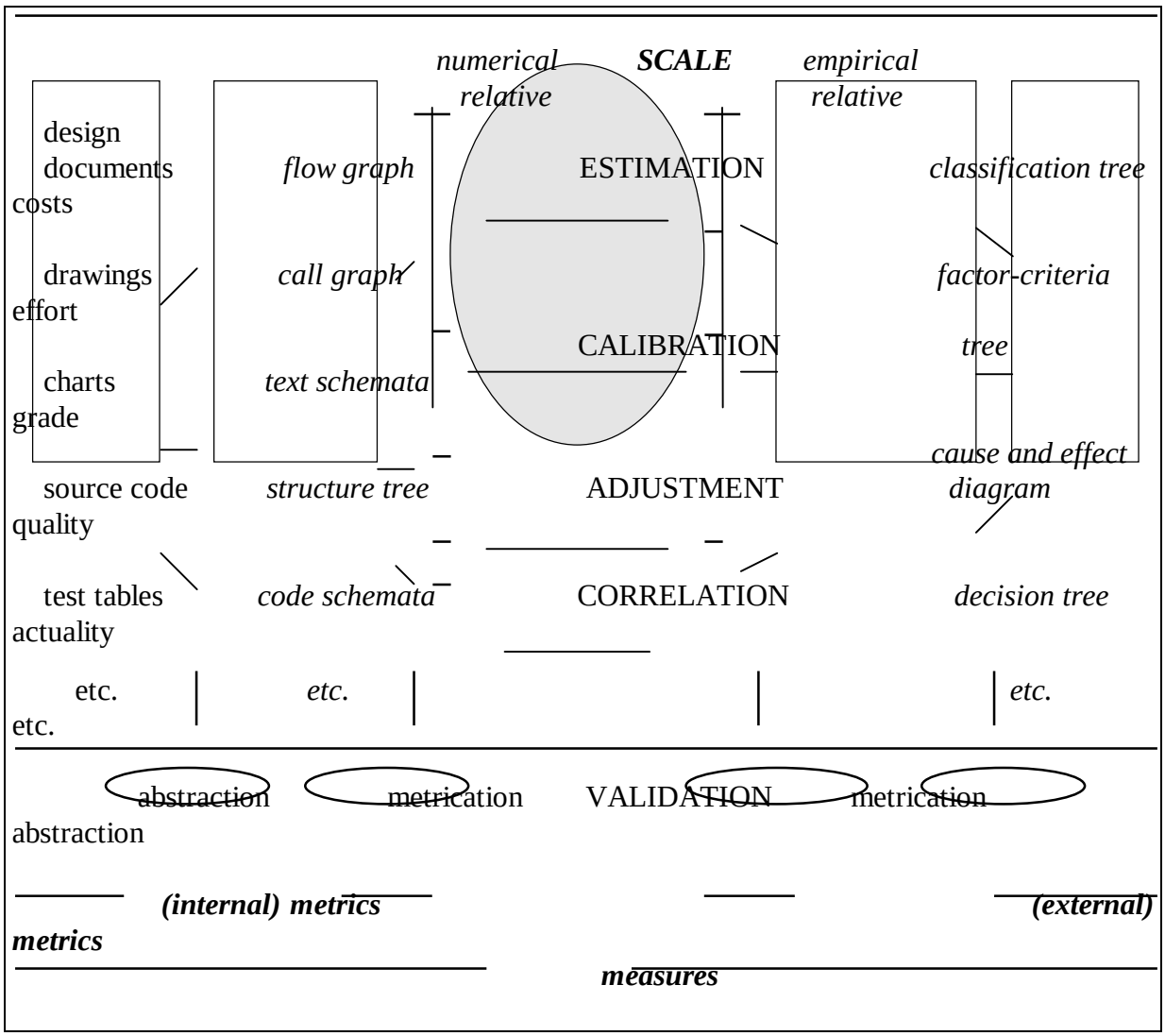
4.2 Measurement Adjustment

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.

software development component	**model**	**measurement (statistical analysis)**	**theoretical view model**	**evaluation (empirical) criteria**



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) [55] and in the Smalltalk measure extension [42] 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 [74] 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 [37]) can be described in a classical manner with the following values:

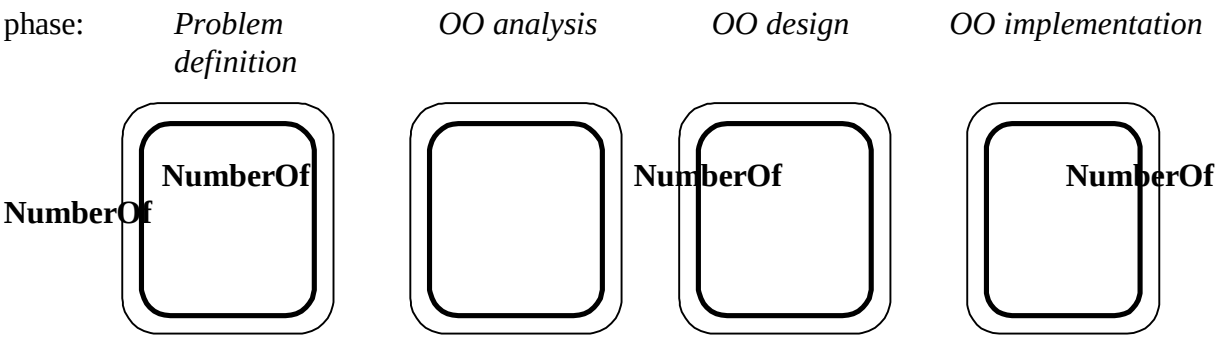
- total lines of JAVA code: 1320,
- JAVA classes: 25,
- average number of methods per class: 12,
- average number of parameters per method: 0.88,
- average lines of code per methods: 4.04, etc.

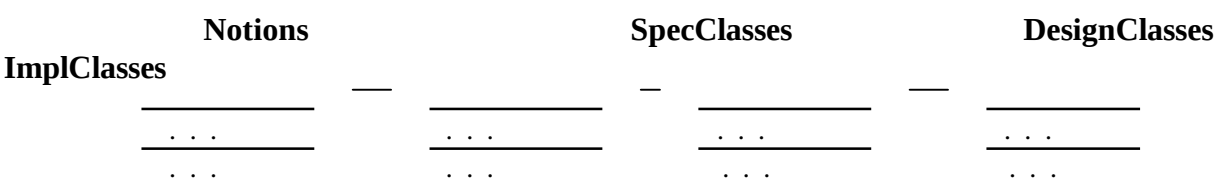
In general we see a conformity of our Java application with the evaluation criteria.

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* [29] 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 [23] is





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.

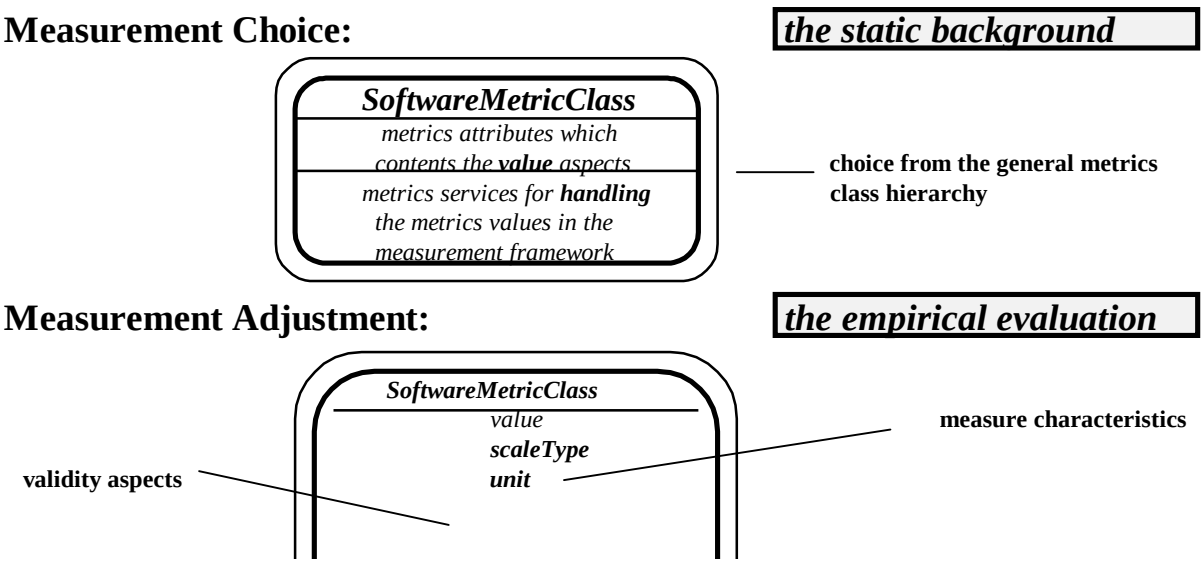
Further, it is necessary to repeat the determination of the 'environmental' metric values in time intervals to allow for a *tuning* of the *favorableValues* and their conditional variations as *validityConstraints* to guarantee the achievement of selected quality aspects. Note, that the migration may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

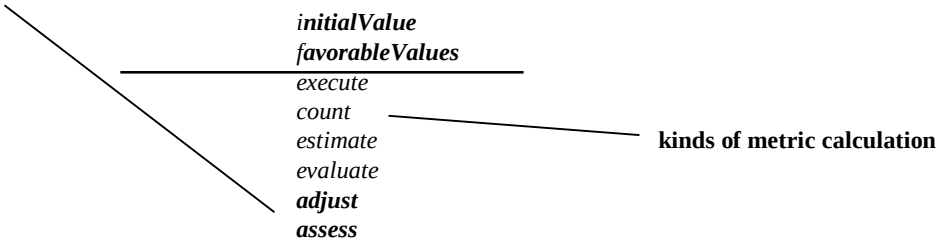
4.4 Measurement Efficiency

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 [22].

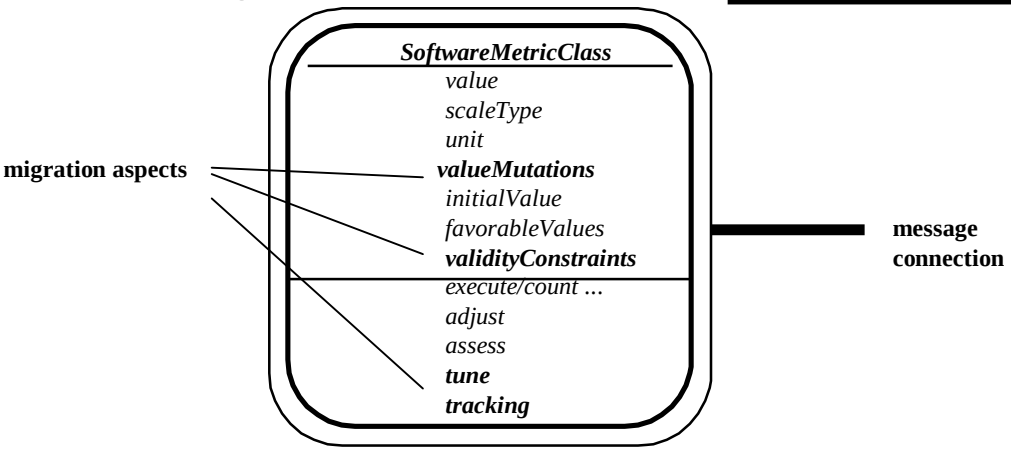
A digest of this framework is given in the next figure. It includes the extension of the metric class to include the facilities necessary to evaluate object-oriented software development.





Measurement Migration:

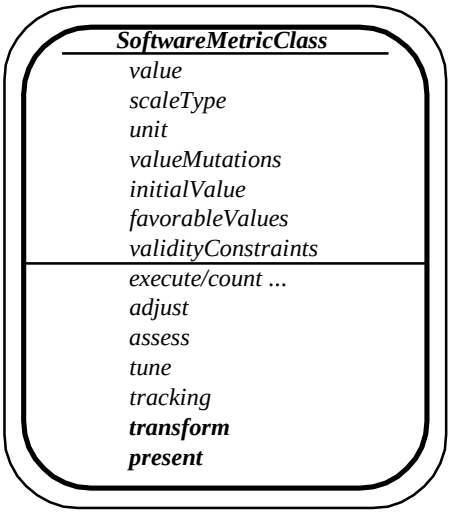
the behavior model



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;

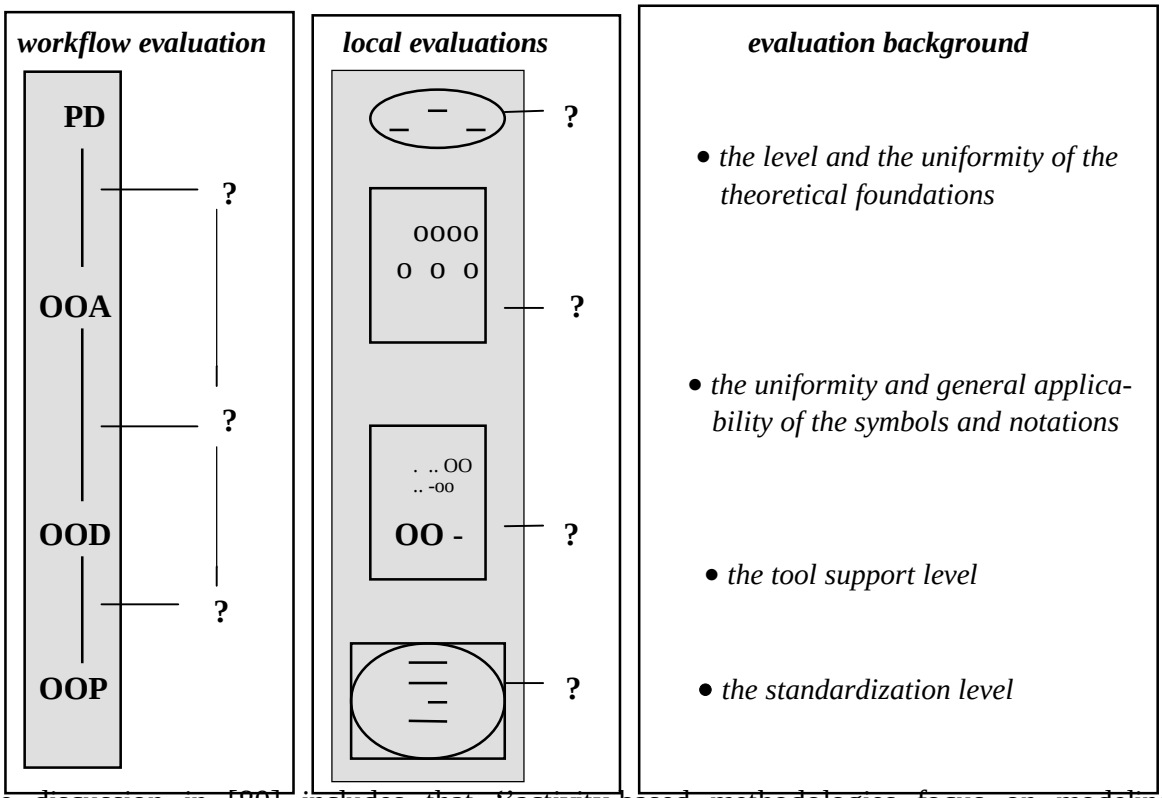
♦ ***OO method process support evaluation****:*
- maturity,
- management (project, quality, configuration),
- life cycle;

♦ ***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 [47], [69], [85] and [82])

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

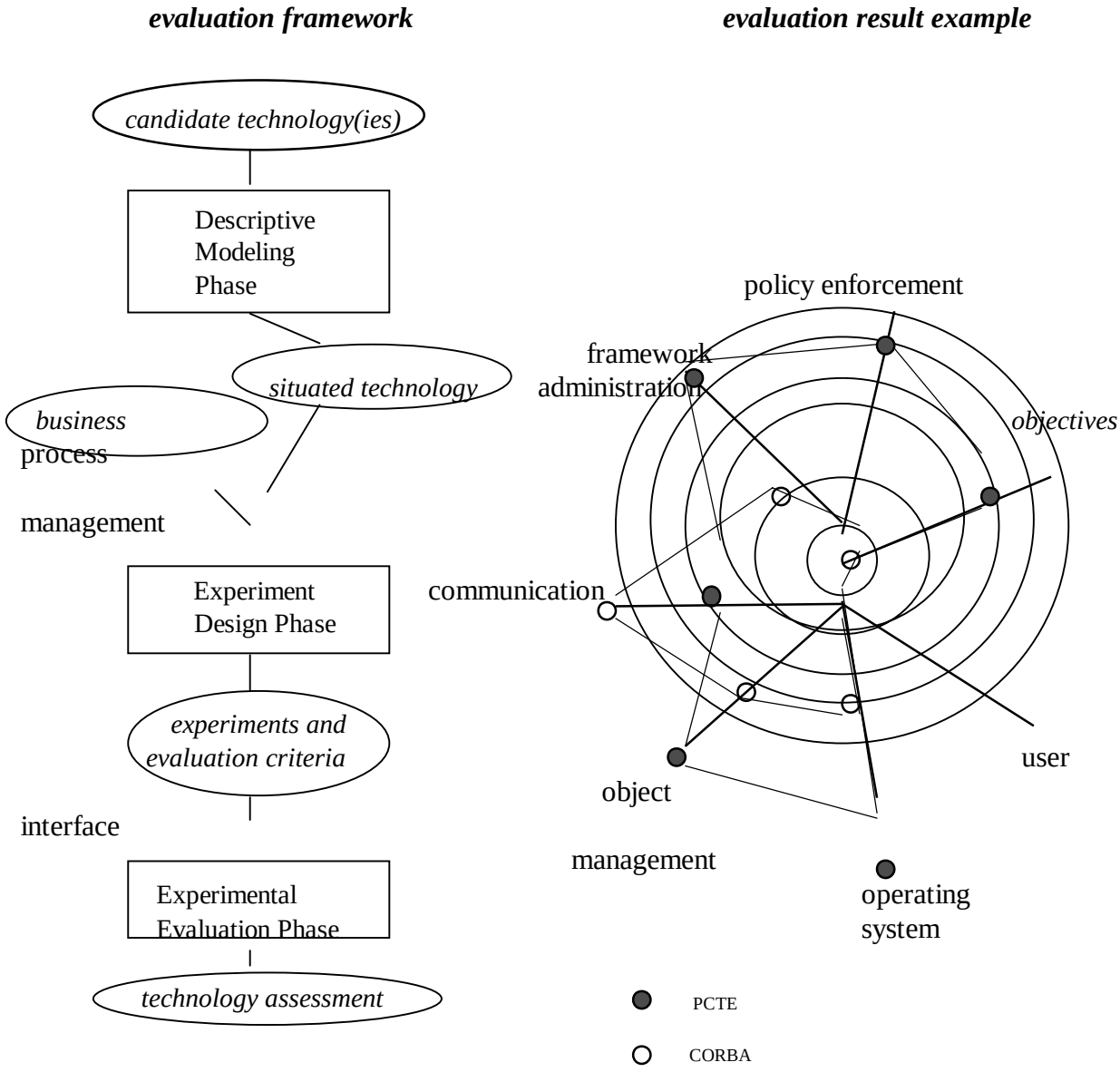
Hence, we must consider the following main areas for a metrication of an object-oriented development methodology:



The discussion in [80] 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 [32]. However, they are aimed at only one issue - the complexity.

A recent description of ***local evaluations*** is given in section 3 of [51]. Metrics related to the text (size and readability) are also used in the specification and design phases [54]. Local evaluations may be considered as the ‘’classical’’ measurement approach. A general concept is

given in [12] and [13]. The main idea of this approach is the *technology delta principle*. The framework includes the following phases related to a given (exemplary) result:



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

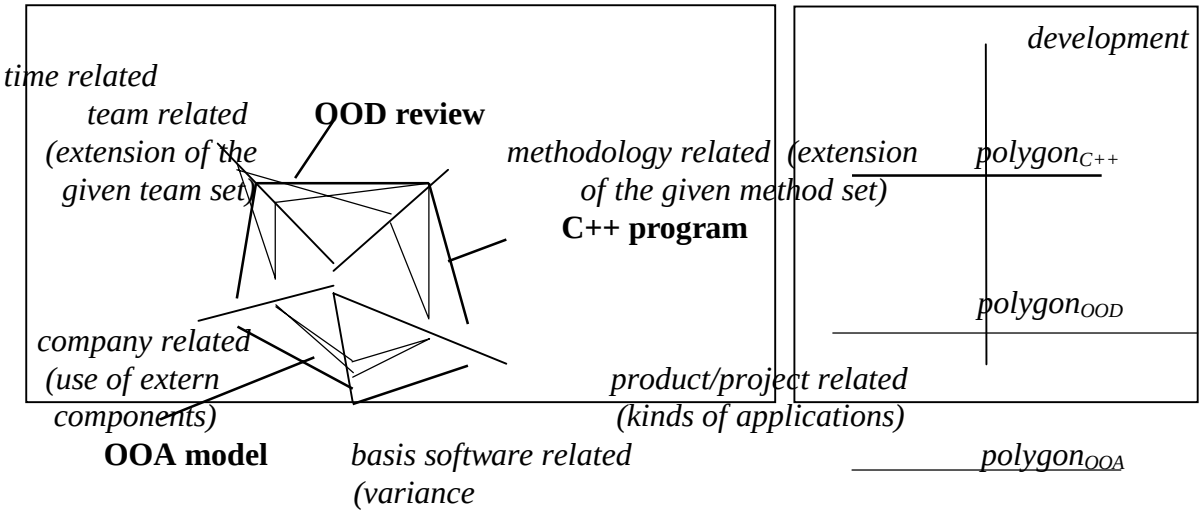
The ***quality agent*** was based on the idea of the (*mobile) intelligent agent* in the area of distributed systems and networks. Mobile agents are computational processes which are

capable of moving from node to node around a network [4]. 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 [69]) 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 [46] 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.



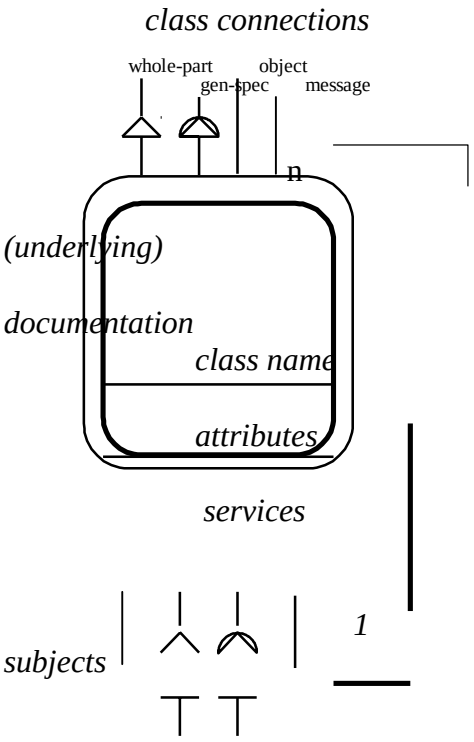
The product size, structure, architecture, usability, efficiency and complexity, the process management and the resource software performance aspects should be described by the ***component workflow agents.*** These agents observe the specification, design and implementation components defined by the 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	*valueMutations:* may be changed from one development phase to another *validityConstraints:* some of the counting components require a continuity along the development phases	*evaluation level:* opposite components, - similar components *presentation:* distance presentation depending on the similarity during the life cycle
	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		
	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		
	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** [21] 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

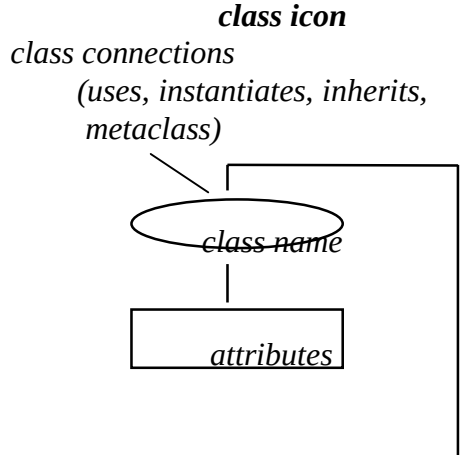
Requirement workflow:
- *kindsOfRequirements:* 2 ('functional', 'system'; monolithically)
- *tracesOfRequirements:* PD→OOA: 0, OOA→ OOD: 2, OOD→OOP: 1; median: 1
- *storageOfRequirements:* median: 1 (textual)

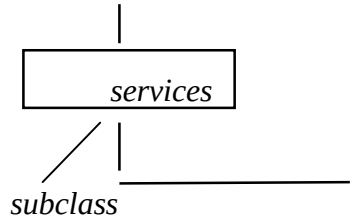
Complexity workflow:
- *similarityOfMethods:* 'stand alone'
- *varianceOfPlatforms:* 'various' (PC, Unix-WS)
- *kindsOfApplications:* 'free'
- *changingOfTeams:* 'indifferently'
- *differingOfComponents:* 2 (OS,OOP language)

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 [10] 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:

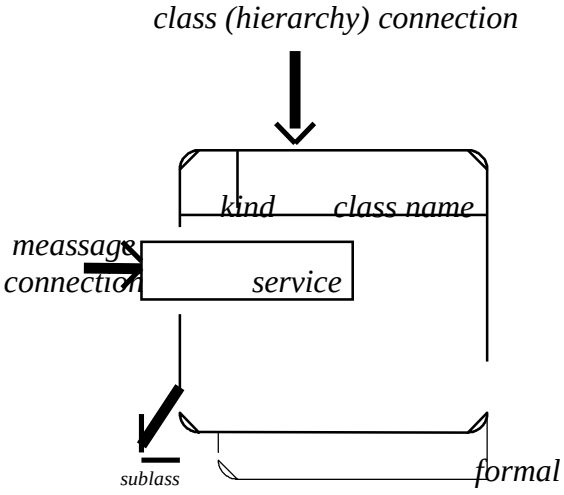
- *similarityOfMethods:* ‘similar’ to modul concept
- *varianceOfPlatforms:* ‘various’
- *kindsOfApplications:* ‘free’
- *changingOfTeams:* ‘indifferently’
- *differingOfComponents:* 2 (OS, OOP language)

Component workflow:

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

The approach from Robinson et al [76] 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 functio-
nal 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)

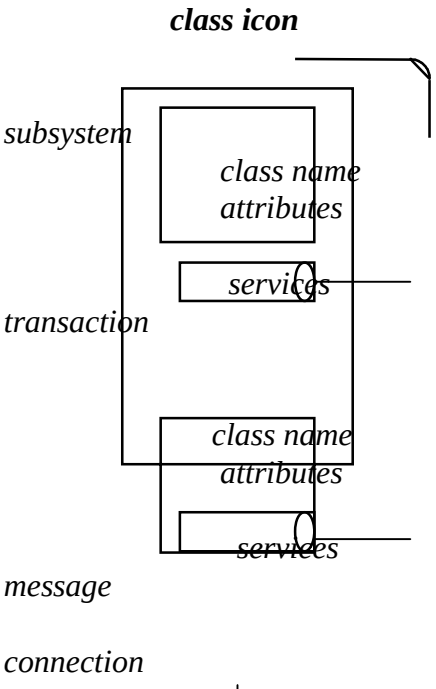
Complexity workflow:

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

Component workflow:

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

For the approach of Wirfs-Brock et al [88] - defined as responsibility-driven design (**RDD**) - we obtain the following assessment.



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, de-pends-upon), Venn diagram for the responsibili-ties

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

quantitative method characteristics

Requirement workflow:

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

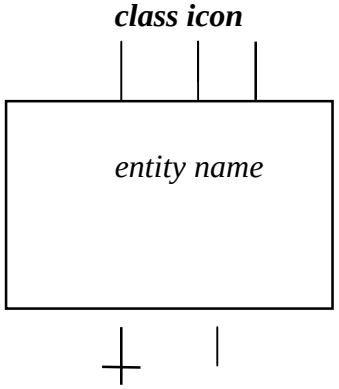
Complexity workflow:

- *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

The Shlaer/Mellor approach ([81] **OOSA)** is based on the idea of an object as an entity used in the ERM paradigm.



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)

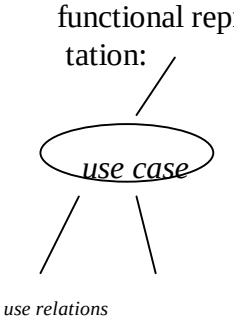
Component workflow:

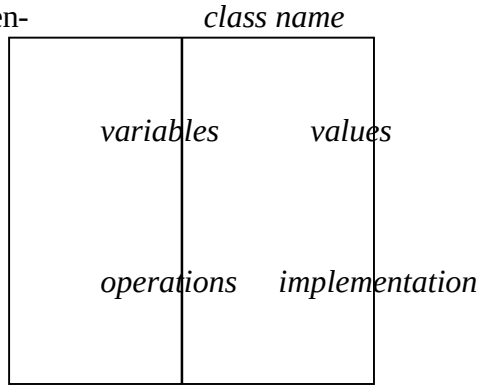
- *numberOfComponents:* 3 (doc.,diagram(s), code)
- *numberOfCharts:* 3 (hierarchy, ER, DFD)
- *numberOfSymbols:* 13 (2 boxes, 11 connec-tions)
- *numberOfRules:* 28

The Jacobson approach **OOSE** [48] defines several types of simple classes. The assessment of this method is given in following.

class icon symbols for the object diagram:

functional represen-tation:





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:

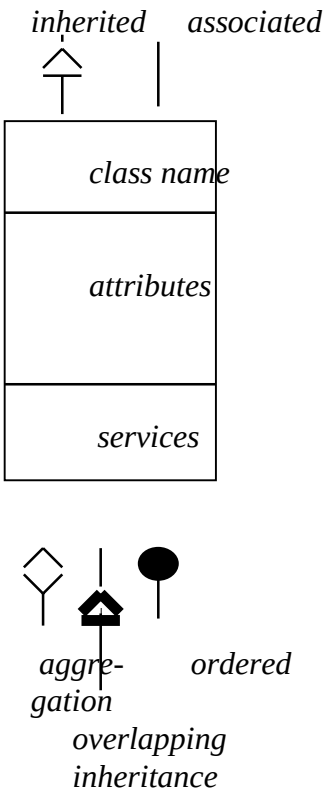
- *similarityOfMethods:* ‘transferable’
- *varianceOfPlatforms:* ‘various’
- *kindsOfApplications:* ‘free’
- *changingOfTeams:* ‘indifferently’
- *differingOfComponents:* 3 (OS, OOP language, state transition diagram (SDL))

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 [78] 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 faci-
lities), 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)

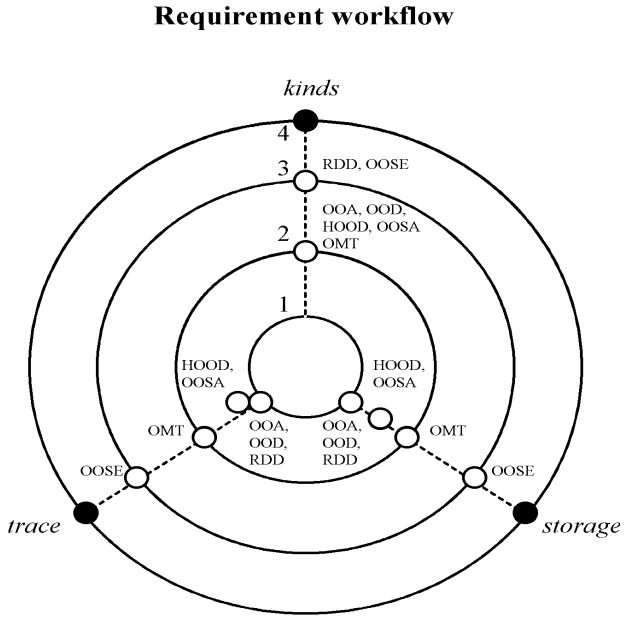
Complexity workflow:

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

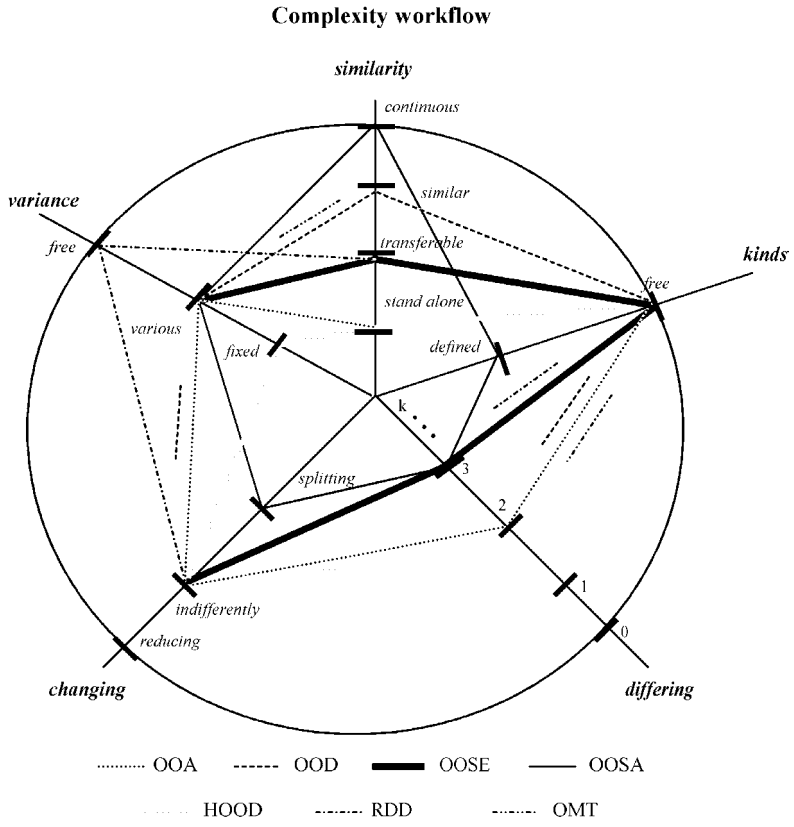
Component workflow:

- *numberOfComponents:* 3 (doc, model(s), code)
- *numberOfCharts:* 3 (object, dynamic, functio-nal)
- *numberOfSymbols:* 19 (8 boxes, 11 connec- tions)
- *numberOfRules:* 59

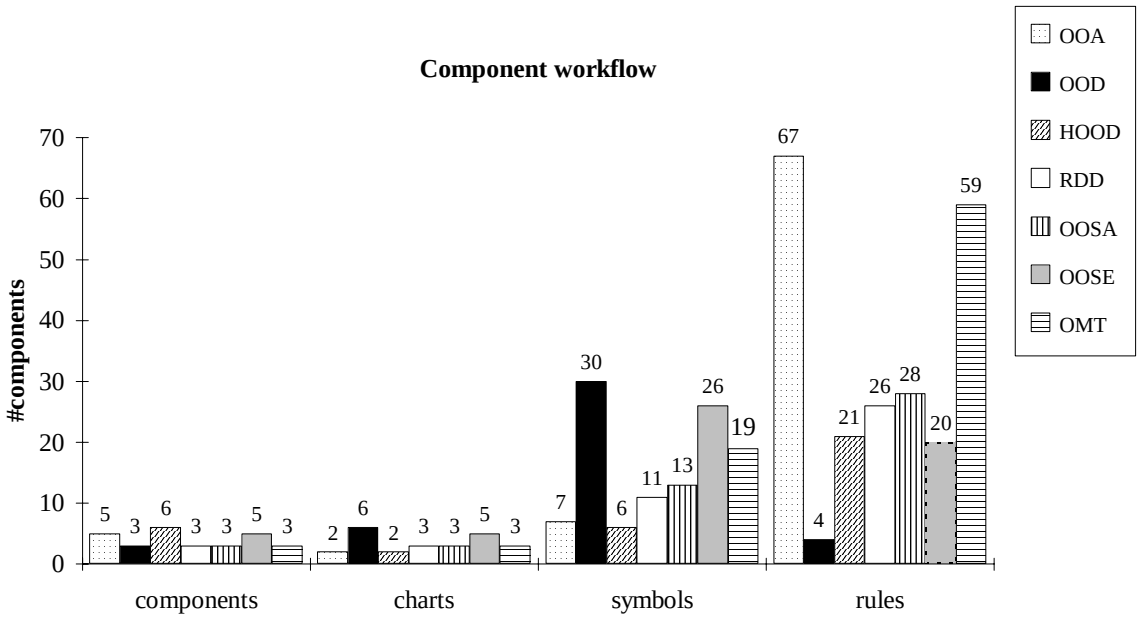
Of course, the evaluation is subject to refinement and therefore open for discussion. The following charts provide a summarization of these evaluations to compare the chosen OO development methods. Note, that this evaluation is only an ***assessment,*** useful as start point of the use of software quality agents. The '•' marked points denote the 'ideal' values of the given aspects.



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



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*** [39]. 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

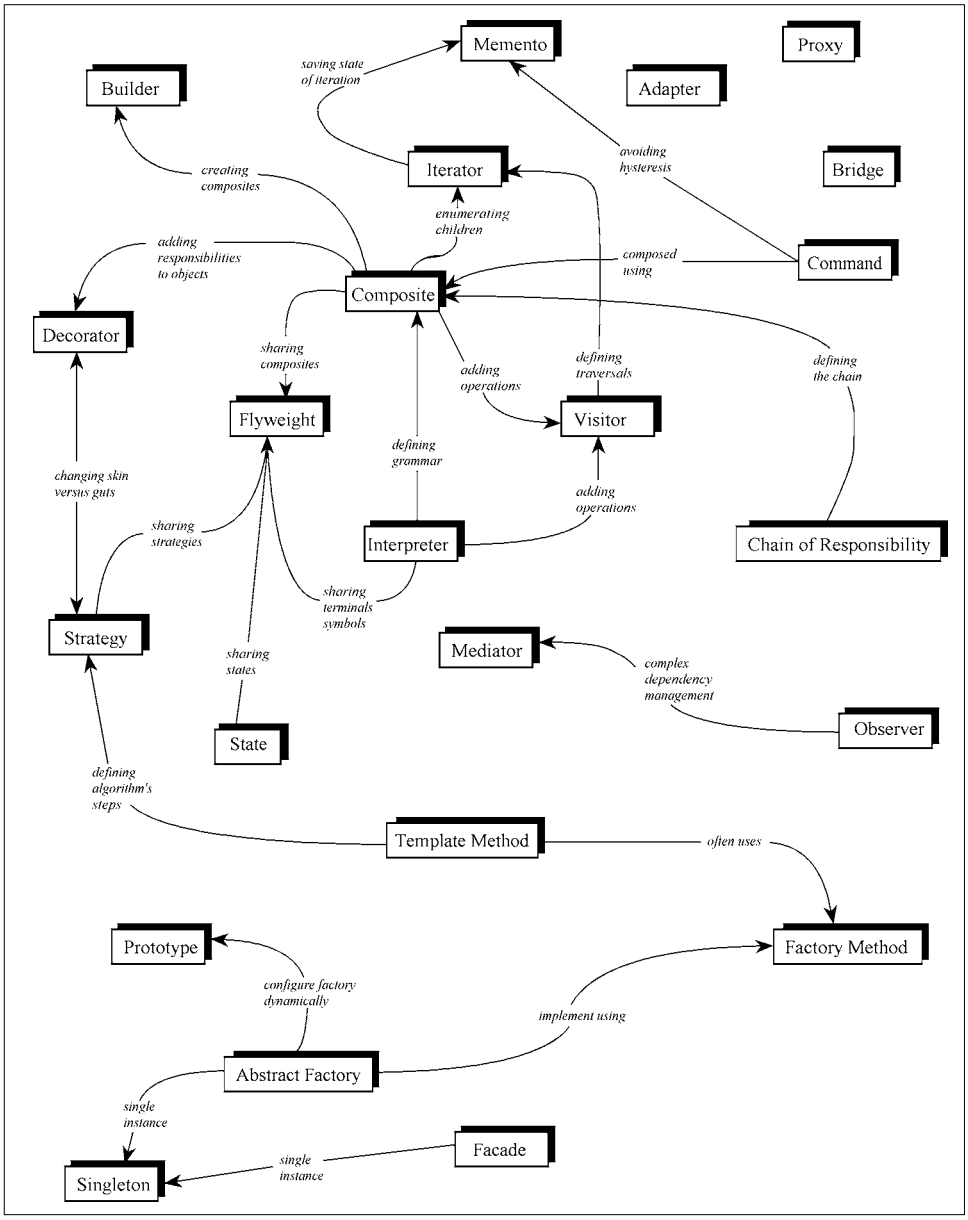
- reducing of product architecture components (by means of standardization),
- increasing the process efficiency in the life cycle,
- using experience for a better process maturity,
- decreasing the structural complexity in the software design,
- increasing of the resource personnel productivity in general.

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 that is instantiated)	**Composite** (structure and composition of an object)	**Iterator** (how an aggregate's 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)*** [11]. 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 [28].

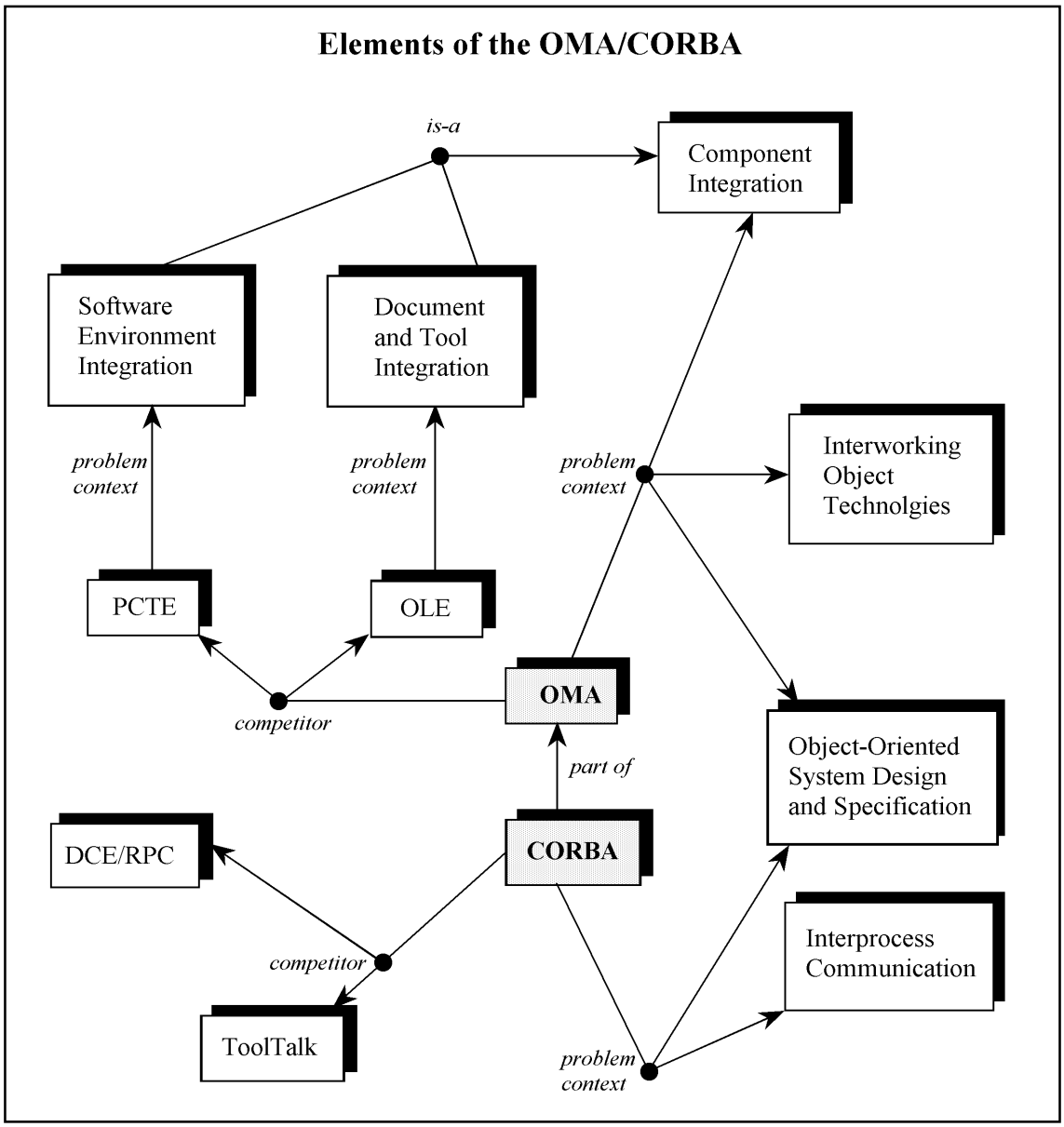
state of components	**characteristics for metrication**
off-the-shelf components (COTS)	unknown/undefined interface; includes the general problem of the estimation of the characteristics of commercial software
qualified components (interface defined)	interface metrics; information hiding aspects
adapted 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; quantification of the general infrastructure (operating system, data base system etc.)
updated components (adaptation to given infrastructure)	metrication of the infrastructure (architecture, 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)*** [71] 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 [12].



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:

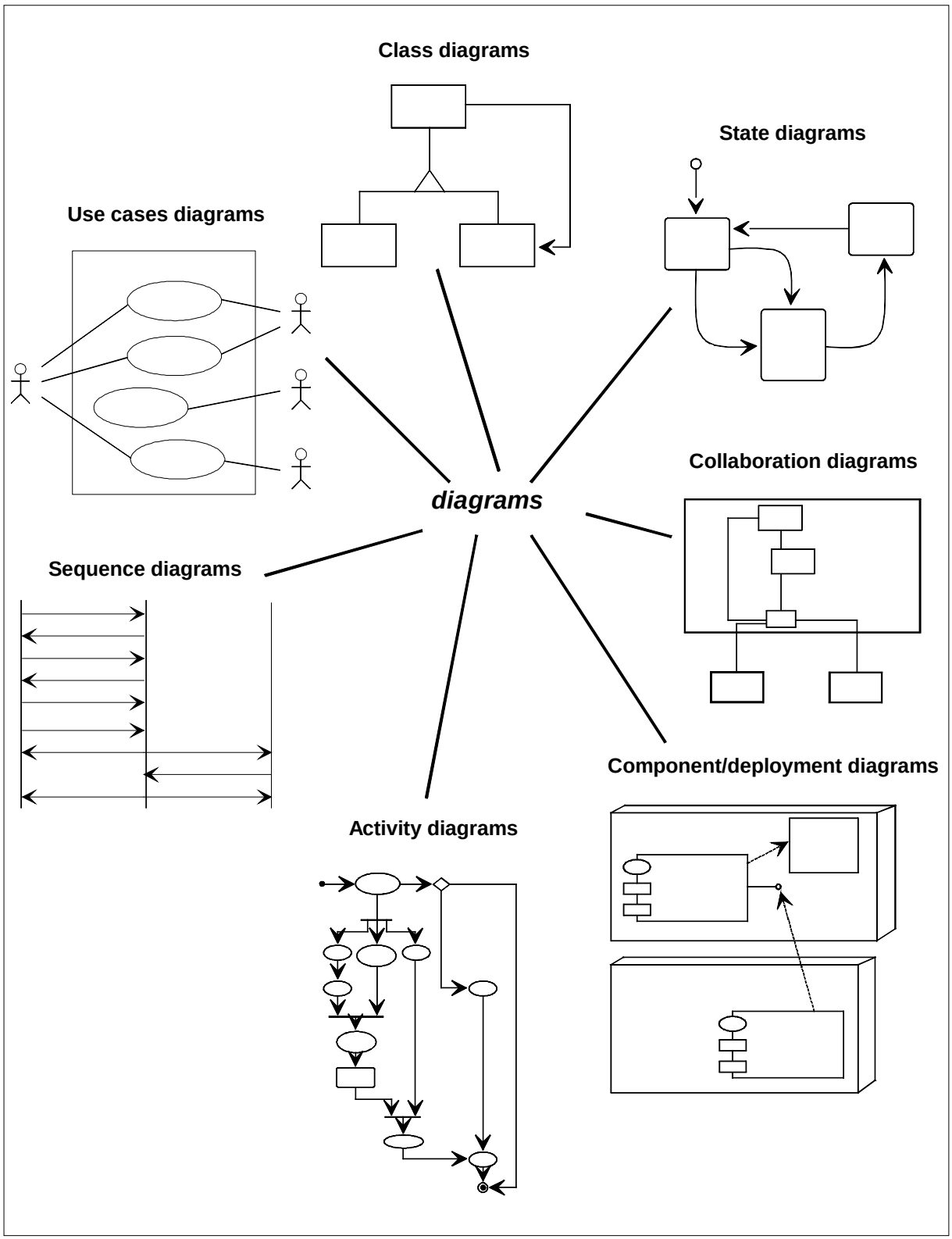
- the general evaluation is similar to the CBSE (see above), because CORBA can be considered as a special kind of component-based development (chosen functionality as *kindsOfRequirements;* some uncertainties in relation to the *tracesOfRequirements* and *storagesOfRequirements;* the *similarityOfMethods* 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)*** [83] [84]. 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)*.



UML is a visual modeling language not a programming language and is based on the diagrams above and a semantic definition [84]. 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 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, Mi-gration 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

References

[1] 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

[2] 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

[3] 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

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

[5] 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

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

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

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

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

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

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

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

[13] 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

[14] 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

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

[16] 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

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

[18] 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

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

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

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

[22] 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

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

[24] 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.)

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

[26] 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

[27] 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

[28] 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

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

[30] 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

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

[32] 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.)

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

[34] 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

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

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

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

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

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

[40] 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

[41] 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

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

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

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

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

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

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

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

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

[50] 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

[51] 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

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

[53] 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

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

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

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

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

[58] 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

[59] 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.

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

[61] 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

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

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

[64] 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

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

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

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

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

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

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

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

[72] 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

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

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

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

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

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

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

[79] 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

[80] 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

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

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

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

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

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

[86] 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

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

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

[89] 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

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

An email information

Fernando Brito e Abreu, INESC - MOOD Project Leader, Lisbon, Portugal

We are actively working on MOODKIT G2 (second generation) which is radically different from previous on (G1). Among the improvement is the ability of metrics capture either by forward (from models in a CASE TOOL) or reverse engineering (from source code in several OO languages). MOODKIT G2 relies on an intermediate OO design language named GOODLY (a Generic Object Oriented Design Language? Yes!).

The GOODLY language is up and running! A GOODLY specifications hypertext browser with high traceability capabilities and several source code examples that were generated with MOODKIT G2 (under construction) are now available at our web site. This bowser will soon show the calculated MOOD metrics values. The MOOD set is being currently reviewed and expanded.

The MOOD Project WWW server is located at the following address:

http://albertina.inesc.pt/ftp/pub/esw/mood

Please use a browser that supports frames (e.g. Netscape 2.0 or later releases).

PRODUCT	STATUS	AVAILABILITY
GOODLY specifications parser and linker	Ready	available on request
GOODLY specifications browser	Ready	use it in the web
GOODLY to Smalltalk converter	2 nd week May	(forecast)
Smalltalk to GOODLY converter	2 nd week May	(forecast)
Eiffel to GOODLY converter	3 rd week May	(forecast)
OMT (ParadigmPlus) to GOODLY converter	3 rd week May	(forecast)
MOOD metrics extraction from GOODLY code	4 th week May	(forecast)
Java to GOODLY converter	4 th week May	(forecast)
C++ to GOODLY parser	2 nd week June	(forecast)
Object Pascal (Delphi) to GOODLY parser	4 th week June	(forecast)

The MOOD team is waiting for your feedback and your cooperation plus!

The MOOD (Metrics for Object Oriented Design) metrics originated from the PhD research work carried out by Fernando Brito e Abreu, enriched by contributions of many others, either originated within the MOOD team or organization where MOOD project team is hosted, see our central web site (http://www.inesc.pt).

The MOOD project is an academic project, not a commercial one! The only thing we ask from you is to share with us the results you got with our tools and your constructive contributions on improving and/or extending the MOOD metrics set. In particular we seek cooperation with reals industrial projects where process data (schedules, effort, defect reports, etc.) are available, in order to construct empirical validation studies, as well as academic theoretical validations ones.

ISBSG - A worldwide Software Measurement Initiative

The ISBSG (International Software Benchmarking Standards Group) had its origins in the work performed by the Australian Software Metrics Association (ASMA) in software benchmarking. In 1990, a Special Interest Group in ASMA met to develop a practical industry standard for quantifying the output from software projects. This led to the establishment of a repository of data on Australian projects in 1992.

The success of this initiative created considerable international interest. In June 1994, the software metrics organisations of New Zealand (SMANZ), the United Kingdom (UFPUG), and the United States (IFPUG), together with ASMA, formed ISBSG. Later other metrics organisations (for instance from Canada, Germany, France) became involved. The ASMA model was used for a de facto international standard. Through ISBSG, the various associations and their members can collect and share data to facilitate international benchmarking. The actual fourth release of the Benchmarking Repository contains data collected from 396 projects from 14 countries.

The ISBSG Repository is based on the following principles:

- **Practitioner Driven** and **Practitioner Accessible**: Each IT-organization, whether they are members of their respective national metrics organisation or not, may contribute to the ISBSG Repository and use the services of ISBSG.
- **Independence** from vested business and research interests whenever they are liable to compromise the objectives of the Repository.
- **Integrity** of the Repository data must be maintained through the application of rigorous procedures.
- **Confidentiality** of the contributors.

The establishment of the ISBSG Repository has made it possible to offer the industry a number of services:

- *The Repository* itself can be used as an alternative to In-house metrics databases
- *A Project Benchmarking Profile Report* is sent back to the contributor. It compares the submitted project with others of the same class within the repository
- *Best Practice Networking* is available for contributors
- *Organisational Benchmarking* is available to organisations to compare themselves against similar organisations
- *ISBSG Releases* (reports on the ISBSG Repository)
- *Customised Analysis and Reports*

ISBSG is working permanently to increase the value of the services offered. At around nine month intervals interested members meet at the ISBSG workshop. At the last workshop, held in conjunction with the IFPUG'97 Spring Conference, two research contracts with the Monash University (Australia) and the Université du Québec à Montréal (Canada) have been initiated.

If you want to learn more about the ISBSG initiative or how to contribute to the ISBSG Repository please see **http://www.bs.monash.edu.au/asmavic/isbsg.htm**.

SMLab's WorldWideWeb Project

The Software Measurement Laboratory of the University of Magdeburg was established to support the Software Metrics efforts of the (local) IT community and to conduct university research and education. As a service for the public, SMLab maintains a Website to inform about new devlopments and to provide a world-wide discussion platform.

In the position paper *Current Situation in Software Measurement Frameworks* beginning on Page 11 of this issue, the author mentions a break between the quality aspects and their quantification with metrics. For the Software Metrics field, a science that is largely dominated by empirical results, conducting experiments and analysing the results is a critical and important step toward the formation of valid models.

In order to provide an overview about experimental results the Software Measurement Laboratory has added a summary of software measurement experiments to its Web-site. The more than fifty eperiment descriptions are grouped in

- *Software Process Experiments* (Process Maturity, Process Management, and Process Life Cycle Experiments)
- *Software Product Experiments* (Size, Architecture, Structure, Quality, and Complexity Experiments)
- *Software Resource Experiments* (Personnel, Software, and Hardware Experiments)

"Classical" Experiments as Halsteads Experiments to the definition of his "Software Science" are included as well as more recent experiments on Object Oriented Programming or World Wide Web design. For every experiment, a reference for further reading is provided. The Software Measurement Laboratory invites you to contribute your experience and experiment to make your results accessible to the software engineering community.

Another point of interest for the practitioner in the software metrics field is the application of Computer Assisted Measurement and Evaluation (CAME) Tools. Based on a general software measurement framework the Web Site contains a short description and evaluation of the better know measurement tools used in the European market.

Some sample on-line applications are available to demonstrate the capabilities offered by hypermedia technologies.

The Web-Site of the Software Measurement Laboratory can be found at:

http://ivs.cs.uni-magdeburg.de/sw-eng/us/

Lehner, F.; Dumke, R.; Abran, A.: Software Metrics - Research and Pracitce in Software Measurement

Gabler-Verlag, Wiesbaden, 1997 (232 p.)

This book contains all presentations of the 1996 workshop of the GI-interest group on software metrics and of the Canadian Group (CIM) in September in Regensburg. It is a collection of theoretical studies in the field of software measurement as well as experience reports on the application of software metrics in Canadian, Austrian, Belgian and German

companies and universities. Some of these papers and reports describe new software measurement applications and paradigms for knowledge-based techniques, maintenance service evaluation, factor analysis discussions and neural-fuzzy applications. Others address the object-oriented paradigm and discuss the application of the Function Point approach to an object-oriented design method, the evaluation of the Java development environment, the analysis of quality and productivity improvements of object-oriented systems, as well as the definition of the metrics of class libraries. Other papers offer a different perspective, presenting a software measurement education system designed to help improve the lack of training in this field, for example, or they include experience reports about the implementation of measurement programs in industrial environment.

ISBN: 3-8244-6518-3

Moore, J.W.: Software Engineering Standards - A User's Rad Map

IEEE Computer Society, 1998 (296 p.)

This book gives a general overview about the software engineering standards - their background and benefits. Therefore, it also includes the software metrics standards such as ISO 9000 et al. and the IEEE-1061-92 (metrics) standard.

Pigoski, T.M.: Practical Software Maintenance - Best Practices for Managing Your Software Investment

John Wiley & Sons, Inc., 1997 (384 p.)

The author discusses the software maintenance from a process view and a process improvement strategy. Therefore, the software maintenance is presented as a part of software process quality supported by a metrics program. Pigoski describes in chapter 14 the software maintenance metrics and in chapter 15 the experiences in this area. The presentations are helpful for software practitioners and include essential examples of metrics applications.

Poulin, J.S.: Measuring Software Reuse

Addison-Wesley, 1997 (195 p.)

With the techniques in this book, you will have the tools you need to design a far more effective reuse program, prove its bottom-line profitability, and promote software reuse within your organization. Measuring Software Reuse brings together all of the latest concepts, tools, and methods for software reuse metrics, presenting concrete quantitative techniques for accurately measuring the level of reuse in a software project and objectively evaluating its financial benefits.

Putnam, L.H.; Myers, W.: Controlling Software Development

IEEE Computer Society, 1996 (79 p.)

This book discusses in a short from the role of process productivity metrics base on size estimation. The authors give an overview about the software process evaluation and its improvement.

Zuse, H.: A Framework of Software Measurement

de Gruyter Publ., Berlin New York, 1997 (755 p.)

This book describes a framework for software measurement from a theoretical, practical and educational view. The main idea is the application of the measurement theory on the area of software measurement.
The book is written in nine chapters and includes exercises for a teaching in software measurement. The chapters describe the software measurement aspect, the history of software measurement, the theoretical foundations from theoretical and practical view, especially the object-oriented software measures, the discussion about the properties and validation, and helpful remarks for a successful application of software measures.
The book includes a CD ROM that include a demo tool for software measurement education based on more than thausend references and metrics.

ISBN 3-11-015587-7

* **2nd Euromicro Working Conference on Software Maintenance and Reengineering (CSMR),**
  March 9-11, 1998, Florence, Italy

* **Empirical Assessment & Evaluation in Software Engineering (EASE),**
  30th March - 1st April 1998, Staffordshire, U.K.

* **Fourth International Conference on Achieving Quality in Software,**
  31 March - 3 April 1998, Venice, Italy

* **Software Quality Management (SQM),**
  6-8 April 1998, Amsterdam,Netherlands

* **Software Measurement (FESMA),**
  6-8 May 1998, Antwerp, Belgium

* **Eleventh International Software Quality Week,**
  26-29 May 1998, San Francisco, USA

* **Evaluation and Evaluation Research in Information Systems,**
  June 5, 1998, Linz, Austria

* **Ninth International Symposium on Software Reliability Engineering (ISSRE),**
  4-7 November 1998, Paderborn, Germany

* *metrics themes are also discussed in the yearly* **OOIS, ECOOP** *and* **ESEC** *conferences*

Other Information Sources and Related Topics

- **http://rbse.jsc.nasa.gov/virt-lib/soft-eng.html**
  Software Engineering Virtual Library in Houston
- **http://www.mccabe.com**
  McCabe & Associates
- **http://www.sei.cmu.edu**
  SEI Pittsburgh
- **http://dxsting.cern.ch/sting/sting.html**
  STING: News Browser, Glossary Search, Projects and Measurement Tools at CERN
- **gopher://gopher.cs.tut.fi/11/pub/src/software-eng/metrics**
  C Metrics Package
- **http://www.spr.com/**
  Software Productivity Research, Capers Jones

- **http://fdd.gsfc.nasa.gov/seltext.html**
  SEL-Homepage
- **http://www.qucis.queensu.ca/Software-Engineering/Cmetrics.html**
  Queens University of Canada
- **http://www.esi.es**
  ESI Spain
- **http://saturne.info.uqam.ca/labo_Recherche/lrgl.html**
  University of Quebec
- **http://www.SoftwareMetrics.com**
  IFPUG Information by David Longstreet
- **http://www.utexas.edu/coe/sqi/**
  Software Quality Institute, University of Texas at Austin
- **http://wwwtrese.cs.utwente.nl/~vdberg/thesis.htm**
  Klaas van den Berg: Software Measurement and Functional Programming
- **http://www.inesc.pt/index-eng.html**
  Metrics for Object Oriented Design (MOOD) Project Team and the
  ftp://albertina.inesc.pt/pub/esw/modd
  MOOD-Server
- **http://divcom.otago.ac.nz:800/com/infosci/smrl/home.htm**
- **http://ivs.cs.uni-magdeburg.de/sw-eng/us/**
  Software Meßlabor der Universität Magdeburg
- **http://www.cs.tu-berlin.de/~zuse**
  Arbeitsgruppe Softwaremetriken
- **http://www.sbu.ac.uk/~csse/publications/OOMetrics.html**
  Object-Oriented Metrics
- **http://www.sbu.ac.uk/~csse/ami.html**
  ami - Application of Metrics in Industry
- **http://www.dfn.de/~atw/bmbf/foerderprogramme/swt/SWT.html**
  Initiative zur Förderung der Software-Technologie in Wirtschaft, Wissenschaft und Technik
- **http://www.iso.ch/9000e/forum.html**
  The ISO 9000 Forum
- **http://ceswww.utexas.edu/sqi**
  Software Quality Institute (SQI)
- **http://www.tiac.net/user/pustaver/**
  The Software Quality Page
- **http://www.theriver.com/qa-inc/**

Quality America, Inc's Home Page

- **http://www.ele.vtt.fi/docs/aslehti/magaz_z.htm**
  A primer for total quality in software development
- **http://www.nist.gov/quality_program/**
  NIST Quality Program
- **http://www.quality.org/qc/**
  Quality Resources Online
- **http://www.almaden.ibm.com/journal/sj33-1.html**
  IBM Systems Journal - Software Quality
- **http://freedom.larc.nasa.gov/spqr/spqr.html**
  Software Productivity, Quality, and Reliability N-Team

News Groups

- **news:comp.software-eng**
- **news:comp.software.testing**
- **news:comp.software.measurement**

METRICS NEWS

VOLUME 2 **1997** **NUMBER 2**

CONTENTS

ISSN 1431-8008