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

Editors:

ALAIN ABRAN
Professor and Director of the Research Lab. in Software Engineering Management
Quebec-University of Montreal
Departement of Computer Science
C.P. 8888 Succursale Centre-Ville, Montreal, H3C 3P8, Canada
Tel.: *+1-514-987-3000, -89000,* ***Fax:*** *+1-514-987-8477*
abran.alain@uqam.ca

MANFRED BUNDSCHUH
Chair of the DASMA
Sander Höhe 5, 51465 Bergisch Gladbach, Germany
Tel.: *+49-2202-35719*
Bundschuhm@acm.org
http://www.dasma.de

REINER DUMKE
Professor on Software Engineering
University of Magdeburg, FIN/IVS
Postfach 4120, D-39016 Magdeburg, Germany
Tel.: *+49-391-67-18664,* ***Fax:*** *+49-391-67-12810*
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*
christof.ebert@alcatel.de

HORST ZUSE
Dr.-Ing. habil. 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*
zuse@tubvm.cs.tu-berlin.de

Editorial Office: Otto-von-Guericke-University of Magdeburg, FIN/IVS, Postfach 4120, 39016 Magdeburg, Germany
Technical Editor: DI Mathias Lother
The journal is published in one volume per year consisting of two numbers.

C A L L F O R P A P E R S

for the 11th International Workshop on Software Measurement
of the German Interest Group on Software Metrics and the
Canadian Interest Group on Metrics (C.I.M.)
In cooperation with
COSMIC – Common Software Measurement International Consortium

August 28-29, 2001 in Montréal (Québec) CANADA

THEME & SCOPE: SOFTWARE SIZE MEASUREMENT

Software measurement is one of the key technologies to control or to manage the software development process. Measurement is also the foundation of both sciences and engineering, and much more research in software is needed to ensure that software engineering be recognized as a true engineering discipline.

In 2001, a significant number of key institutional documents will be made in the public domain where measurement is considered a fundamental issues (such as the IEEE- Guide to the Software Engineering Body of Knowledge and ISO standards specifics to Measurement).

Therefore, it is necessary to exchange between researchers and practitioners the experiences on the design and uses of measurement methods to simulate further theoretical investigations to improve the engineering foundations through measurement.

The purpose of the workshop is to review the set of issues such as the identification of deficiencies in the design of currently available measurement methods, the identification of design criteria and techniques and measurement frameworks.

We are looking for papers in the area of software measurement, addressing generic research issues, infrastructure issues or specific research and implementation issues on the following issues and topics(but not limited to):

A - Measurements within institutional documents:

- IEEE – Guide to the Software Engineering Body of Knowledge – SWEBOK project `www.swebok.org`
- ISO/IEC JTC1/SC7 new standards and work-in-progress on software measurement
- Measurement program frameworks publicly available

B - Objects and attributes to be measured:

- Types of measurement object targets (functional domains, type of software – layers, specific functional characteristics - algorithms)
- Timely adaptation of the designs of measurement methods to new and emerging technologies (OO, Multi-media, Web-based applications, etc.)
- Size attributes categories (Functional, Technical, Quality, etc.)

C - Measurement methods: design issues

- Design issues of measurement methods: definition of base components to be measured, ISO conformance, weights assignments and theoretical foundations (Basis for consensus, degree of consensus, etc.
- Normalization issues: time dependence, technology dependence, infrastructure changes
- Integration of measurement types: when and how.
- Quality of measurement methods (repeatability accuracy, correctness, traceability, uncertainty, precision, etc).

D - Uses of measurements results in relationships with other measures:

- Productivity Analysis (foundations of productivity models, quality of productivity models, experimental basis and constraints that limit it expandability to contexts outside of the experimental basis).
- Estimation process (uncertainty, identification of inputs, expectations, technical estimates versus business risks estimation, etc.).

PROGRAM COMMITTEE

Alain Abran, University du Québec à Montréal - UQAM, Canada
Fernando Brito e Abreu, INESC Lisboa, Portugal
Luigi Buglione, Invited professor, UQAM
Manfred Bundschuh, DASMA, Germany
François Coallier, Bell Canada, Canada
Jean-Marc Desharnais, CIM Montreal, Canada
Reiner Dumke, University of Magdeburg, Germany
Christof Ebert, Alcatel Antwerp, Belgium
Martin Hitz, University of Klagenfurt, Austria
Franz Lehner, University of Regensburg, Germany
Serge Oligny, UQAM, Canada
Geert Poels, Vlekho Brussel, Belgium
Andreas Schmietendorf, T-Nova Berlin, Germany
Harry Sneed, SES Munich/Budapest, Hungary
Charles Symons, COSMIC - UK
Horst Zuse, TU Berlin, Germany

SUBMISSIONS

Authors should send abstract (1-2 pages)
by mail, fax or e-mail by May 1st, 2001 to

Alain Abran
University of Quebec
Dept. of Computer Science
C.P.8888, Succ. Centre-Ville
Montreal (Quebec), Canada H3C 3P8
Tel.: +1-514-987-3000, ext. 8900
Fax: +1-514-987-4501
abran.alain@uqam.ca

or to

Reiner Dumke
Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Postfach 4120
D-39016 Magdeburg, Germany
Tel.: +49-391-67-18664
Fax: +49-391-67-12810
dumke@ivs.cs.uni-magdeburg.de

WORKSHOP TIMETABLE

Submission deadline of abstract: May 1st, 2001

Notification of acceptance:	May 10, 2001
Position paper deadline:	July 15, 2001
Workshop date:	August 28-29, 2001

FEES for authors: none

NEWS

For the latest news about the Workshop see the following Web site:

http://lrgl.uqam.ca/workshop2001

C A L L F O R P A P E R S

Workshop der GI-Fachgruppe 2.1.10 "Software-Messung und -Bewertung"

vom 10.9. - 11.9.2001
an der Universität Kaiserslautern

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

Neben dem einfachen Einsatz von Software-Metriken haben sich im industriellen Bereich immer mehr Software-Messprogramme etabliert, die eine kontinuierliche Software-Messung und Bewertung von ausgewählten Produkt- und Prozessmerkmalen der Software-Entwicklung gewährleisten. Bei der methodischen Vorgehensweise hat sich dabei besonders die Goal-Question-Metrics-Methode bewährt. Andererseits dringen auch immer neue Software-Technologien, wie die komponentenbasierte oder agentenbasierte Entwicklung auf den Markt, deren Auswirkungen auf die Prozess- oder gar Produktqualität noch kaum untersucht wurden. Dazu zählt auch die Erschließung neuer Anwendungsfelder, wie das eCommerce.

Der diesjährige Workshop widmet sich daher vor allem (jedoch nicht ausschließlich) den Themenschwerpunkten

- Erfahrungsberichte zu Metriken-Programmen in der Praxis,
- Anwendungserfahrungen bei der Aufwandsschätzung, insbesondere mit der Function-Point-Methode sowie dem FFP,
- Lösungsformen und Erfahrungen in der Messdatenhaltung,
- theoretische Grundlagen der metriken-basierten Software-Entwicklung und -Anwendung,
- Anwendung neuer Technologien für die Umsetzung und Installation von Metriken-Programmen,
- Erschließung weiterer Bereiche durch quantifizierte Mess- und Bewertungsformen (komponentenbasierte Software-Entwicklung, eCommerce, Web Engineering usw.).

Für die Präsentation sind ca. 20 Minuten vorgesehen, um jeweils ausreichend Zeit für Diskussionen zur Verfügung zu haben. Darüber hinaus sollen wiederum die bewährten Panel-Diskussionen Anwendung finden. Die Beiträge werden im Rahmen der Buchreihe "Information Engineering und IV-Controlling" beim Deutschen Universitätsverlag veröffentlicht.
Für die Zeit des Workshops besteht die Möglichkeit von Tool-Demonstrationen zum Gebiet der Software-Messung und -Bewertung.
Beiträge schicken Sie bitte per Post oder per Email bis zum **15. Juli 2001** an eine der beiden Adressen

Prof. Dr. Dieter Rombach
Fraunhofer Institut für Experimentelles
Software Engineering
Sauerwiesen 6
D-67661 Kaiserslautern
rombach@iese.fhg.de

Prof. Dr. Reiner Dumke
Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Postfach 4120
D-39016 Magdeburg
dumke@ivs.cs.uni-magdeburg.de

Sollte die Zahl der Präsentationen zu groß werden, so treffen die Organisatoren eine Auswahl.

C A L L F O R P A R T I C I P A T I O N

The 4th European Conference on
Software Measurement and ICT Control
(FESMA-DASMA 2001)

Heidelberg, Germany
May 9-11, 2001

http://www.ti.kviv.be/conf/fesma.htm

SCOPE AND THEMES

The scope of the future FESMA conferences will be broadened in the sense that these will also include subjects in the area of Information and Communication Technology (ICT) management control. This will not replace the traditional scope of measurement but will be supplementary. The reason is that the application of measurement is not a target itself but a component of the ICT management control system of an organisation. With this diversification, FESMA makes its conferences more attractive for target groups in the area of ICT and user management and for controllers, financial officers, auditors, a.o. who are involved in support of ICT management.

The conference theme is

"Managing ICT in internetworked enterprises"

The subtheme for ICT professionals is

"Measures for quality control and cost estimation in the e-world"

The subtheme for management and support functions is

"The balanced scorecard as a support tool for ICT management and ICT involved user management"

TOPICS

Quality and cost of E-business applications
Quality and cost of internet site development
ICT balanced scorecard : new developments and experiences
Evaluating new ICT-related methods, techniques and tools
Evaluating and controlling software process improvement programs
System and software cost and quality benchmarking studies
System and software size measurement and estimation
System and software quality measurement and prediction
Empirical investigations of system and software quality and costs
Machine learning and other AI techniques for
advanced analysis of system and software quality and costs
Assessing conceptual schema and database schema quality
Measurement support for UML-compliant system and software development
Measurement support for component-based development
Measurement support for system and software reengineering
Other topics related to the application of ICT balanced scorecard, software measurement and software process improvement are also welcome.

FESMA

This non-profit making organisation was founded in Amsterdam in 1996 to co-ordinate and support the activities of the various Software Metrics Associations in Europe. The main objective of FESMA is to promote the use of software metrics, in the broadest sense, to enable best practice in the development and delivery of software products. At the moment software metrics associations from 10 European countries: *Belgium*, *Denmark*, *Finland*, *France*, *Germany*, *Great Britain*, *Italy*, *the Netherlands*, *Spain* and *Sweden* are participating in FESMA. *Canada* and *Japan* are associated members.

IMPORTANT DATES

Conference	May 9-11, 2001
Workshops/Tutorials	May 7-8, 2001

PRE-CONFERENCE TUTORIALS AND WORKSHOPS

A number of tutorials and/or workshops will be run on Monday and Tuesday, May 7 and 8, 2001.

VENDORS

A tools fair will be held during the conference to give attendees the opportunity to see the newest tools available. If you would like to display at the tools fair please contact the conference director.

PROGRAM CHAIRS

Geert Poels, Katholieke Universiteit Leuven/Vlekho-Brussel, Belgium
gpoels@vlekho.wenk.be)

Manfred Bundschuh, DASMA, Germany
bundschuh@acm.org

PROGRAM ADVISORY BOARD

Alain Abran, Université de Québec à Montréal, Canada
Guido Dedene, Katholieke Universiteit Leuven, Belgium
Reiner Dumke, University of Magdeburg, Germany
Bruno Peeters, DEXIA Bank, Belgium
Dieter Rombach, University of Kaiserslautern, Germany
Eberhard Rudolph, Hochschule Bremerhaven, Germany
Rini van Solingen, Fraunhofer IESE, Germany

CONFERENCE DIRECTOR

Martin Hooft van Huysduynen, FESMA,
Oosterzijweg 43, 1851 PC Heiloo, The Netherlands
Tel.: +31 654264386
mjhooftvh@cs.com

CONFERENCE ADMINISTRATION OFFICE - CONTACT DETAILS

Rita Peys,
FESMA Conference Manager,
Technologisch Instituut vzw,

Desguinlei 214,
B-2018 Antwerpen, Belgium
Tel.: +32 3 216 09 96
Fax: +32 3 216 06 89
fesma@conferences.ti.kviv.be

URL of conference web site:

http://www.ti.kviv.be/conf/fesma.htm

VENUE

The conference will be held at the Marriott Hotel in Heidelberg. The hotel is located on the banks of the river Neckar with its own landing jetty, only 500 meters from the motorway exit as well as from Heidelberg's main train station. There is an underground parking.

HEIDELBERG

Heidelberg : *the fairy tale setting has captivated imaginations and inspired creative hearts for centuries. In works preserved for all time. From writers such as Goethe, Eichendorff, Hölderlin, Jean Paul, Victor Hugo, and Mark Twain to name just a few, to painters including Turner, Rottmann, Issel, and Trübner, who created rich paintings, the town on the Neckar River. Composers such as Schumann, C.M. von Weber, Brahms also captured their impressions of Heidelberg's unique blend of river landscape, historic town, and hillside castle in their music.*

C A L L F O R P A R T I C I P A T I O N - PE2001

2th GI-Workshop Performance Engineering within the Software Development

University of the German Federal Armed Forces Munich
19. April 2001

SCOPE

One of the most critical non-functional quality factors of a software system is the performance characteristic. The main idea of performance engineering is to consider the performance as a design target throughout the whole software development process and especially in its early phases. The goal of the PE2001 workshop is to bring together experts from industries and research within the field of performance- and software-engineering.

PRELIMINARY WORHSHOP PROGRAM

M. Christiansen, H. Herting, E. Rohde: Overview and compatibility of the tools Strategizer - Best/1 - s_aturn within the software performance engineering
J. Luethi, C. M. Llado: Sensitivity Analysis of an EJB Performance Model using Interval Parameters
R. Gerlich, R. Gerlich: Performance and Robustness Engineering: A Potential Conflict
H. Eckardt: Bottleneck-analysis – a analytical method for the performance assessment
E. Dimitrov, A. Schmietendorf, K. T. Atanassov: Generalised Nets models for the performance analysis of multi-tier client/server systems
D. Stoll, P. Rauch, C. Janczewski, E. Lipper: Evaluating Architecture and Design Issues via Performance Modelling: A Case Study
R. Dumke, C. Wille: Performance engineering methods of agent-based software-systems
A. Schmietendorf, R. Hopfer: Overview to the use of benchmarks within the performance evaluation of hard- and software-systems
H. Ultsch: Web-Performance-Measurement with „HowAreYou“ (product presentation)

PROGRAM COMMITTEE

Prof. Dr. R. Dumke, Otto-von-Guericke-Universität Magdeburg
Dipl. Ing. Dipl. Inform. A. Schmietendorf, T-Nova, EZ Berlin
Prof. Dr. R. Hopfer, HS für Technik und Wirtschaft Dresden
Prof. Dr. F. Lehmann, Universität der Bundeswehr München
Prof. Dr. C. Rautenstrauch, Otto-von-Guericke-Universität Magdeburg
Dipl. Ing. H. Herting, DeTeCSM, Benchmarklabor Darmstadt
Dipl. Inform. A. Scholz, Accenture Unternehmensberatung
Prof. Dr. F. Victor, Fachhochschule Köln

ORGANISATION

The workshop will be held at the university of the German Federal Armed Forces in Munich. The workshop language will be German. Further information about the registration procedure can be found at the following web-side:

http://www-wi.cs.uni-magdeburg.de/pe2001/

Our 10th Workshop on Software Measurement took place in Berlin in October 2000. The following report gives an overview about the presented papers. Furthermore, the papers are published in the following Springer book:

Lecture Notes in Computer Science 2006

NEW APPROACHES IN SOFTWARE MEASUREMENT

Reiner Dumke and Alain Abran (Eds.)

10th International Workshop, IWSM 2000
Berlin, Germany,
October 4-6, 2000

Springer Publisher 2001

ISBN: 3-540-41727-3

Impact of Inheritance on Metrics for Size, Coupling, and Cohesion in Object -Oriented Systems

Dirk Beyer, Claus Lewerentz, Frank Simon
Software Systems Engineering Research Group
Technical University Cottbus, Germany
(db|cl|simon)@informatik.tu-cottbus.de

Abstract. In today's engineering of object oriented systems many different metrics are used to get feedback about design quality and to automatically identify design weaknesses. While the concept of inheritance is covered by special inheritance metrics its impact on other classical metrics (like size, coupling or cohesion metrics) is not considered; this can yield misleading measurement values and false interpretations. In this paper we present an approach to work the concept of inheritance into classical metrics (and with it the related concepts of overriding, overloading and polymorphism). This is done by some language dependent *flattening* functions that modify the data on which the measurement will be done. These functions are implemented within our metrics tool *Crocodile* and are applied for a case study: the comparison of the measurement values of the original data with the measurement values of the flattened data yields interesting results and improves the power of classical measurements for interpretation.

Measuring Object-Orientedness: the Invocation Profile

Peter Rosner[1], *Tracy Hall*[2], *Tobias Mayer*[1]
[1]Centre for Systems and Software Engineering, South Bank University,
Borough Rd, London SE1 0AA, UK
+44 207 815 7473
rosnerpe@sbu.ac.uk
tobias@sbu.ac.uk
[2]Department of Computer Science, University of Hertfordshire, Hatfield,
Hertfordshire, AL10 8AB, UK
hallt@herts.ac.uk

Abstract. This paper introduces the *invocation profile* as the basis for a suite of metrics to indicate the presence and mix of object-oriented mechanisms in a system written in an object-oriented language. This addresses concerns of practitioners and stakeholders that object-oriented mechanisms should be adequately exploited in such a system and gives an indication of the skills needed by developers for system enhancement and maintenance. An outline is given of plans to implement this metrics suite for systems written in Java.

CEOS - a Cost Estimation Method for Evolutionary, Object-Oriented Software Development

Siar Sarferaz[1], Wolfgang Hesse[2]
[1]microTOOL GmbH, Voltastr. 5, D–13349 Berlin, Germany
Tel.: +49-030-467086-0
Siar.Sarferaz@microTOOL.de
[2]FB Mathematik/Informatik, Universität Marburg, Hans Meerwein-Str.,
D-35032 Marburg, Germany
Tel.: +49-6421-282 1515, Fax: +49-6421-282 5419
hesse@informatik.uni-marburg.de

Abstract. In this article we present a method for estimating the effort of software projects following an evolutionary, object-oriented development paradigm. Effort calculation is based on decomposing systems into manageable building blocks (components, subsystems, classes), and assessing the complexity for all their associated development cycles. Most terms of the complexity calculation formulae carry coefficients which represent their individual weights ranging from factors for particular features up to general influence factors of the project environment. These coefficients can continuously be improved by statistical regression analysis.
Outstanding features of the method are its flexibility (allowing estimations for project portions of any size) and its capability to deal with dynamic adjustments which might become necessary due to changed plans during project progress. This capability reflects the evolutionary character of software development and, in particular, implies revision, use and evaluation activities.

A Measurement Tool for Object Oriented Software and Measurement Experiments with it

Li Xinke, Liu Zongtian, Pan Biao, Xing Dahong
Institute of Microcomputer Application, Hefei University of Technology,
Hefei 230009, P.R.C, China

Abstract. The research on software metrics has a long history for more than forty years, but the research on object-oriented (OO) software metrics has been going on for a few years only. C&K metrics is one of the most famous researches on OO software metrics. First, this paper analyses the shortcoming of the C&K metrics suite for object-oriented design and provides an improved metrics suite. Then the paper introduces a practical C++ measurement tool, SMTCPP, implemented by the authors based on improved metrics. SMTCPP parses C++ programs by the LL(1) method, extracts a lot of program information, such as classes, members and objects; counts the indications, such as the number of methods per class, the biggest complexity among methods, depth of inheritance tree, the number of children, coupling between object classes, response for class, and relative lack of cohesion in methods. The measure values are very useful to guide the software process. The tool may also put the values into a database to collect sufficient data for building a software quality evaluation model. Last, the paper analyses the experiments for three practical programs. The result shows that SMTCPP is useful.

Estimating the Cost of Carrying out Tasks Relating to Performance Engineering

Erik Foltin[1], *Andreas Schmietendorf*[1,2]
[1] Otto-von-Guericke-Universität Magdeburg, Fakultät Informatik, Institut für Verteilte Systeme,
Postfach 4120, D-39016 Magdeburg,
Tel.: +49-391-6712701, Fax: +49-391-6712810,
foltin|schmiete@ivs.cs.uni-magdeburg.de
[2] T-Nova Deutsche Telekom Innovationsgesellschaft mbH, Entwicklungszentrum Berlin,
Wittestraße 30N, D-13476 Berlin,
Tel.: +49-30-43577-633, Fax: +49-30-43577-460,
A.Schmietendorf@telekom.de

Abstract. The study presented here analyzes the methods currently used to estimate costs, and how these methods map the tasks related to Performance Engineering (PE) and the costs thereof. To create transparency and acceptance of these extremely important tasks within the context of software development, an approach is pursued which derives the required costs from a corresponding risk analysis and thus examines the business process to be supported, the software development and normal operation. Initial empirical studies are presented which highlight the general trends for possible costs of specific PE methods.

MEASUREMENT IN SOFTWARE PROCESS IMPROVEMENT PROGRAMMES: AN EMPIRICAL STUDY

Tracy Hall[1], *Nathan Baddoo*[1], *David Wilson*[2]
[1]University of Hertfordshire, UK
[2]University of Technology, Sydney, Australia

Abstract. In this paper we report on our empirical study of SPI programmes in thirteen UK software companies. We focus on companies' approaches to SPI and how measurement relates to SPI in those companies. We present quantitative data characterising SPI and measurement in the companies. We discuss how the use of measurement relates to the maturity of software processes and how measurement supports maturing processes.
Our results show that companies are generally enthusiastic about implementing measurement and they believe that SPI is impoverished without measurement. However our data shows that in reality companies have implemented very little substantive measurement. Indeed we suggest that companies find implementing measurement within SPI more difficult than they expect. Furthermore, we report on data from software personnel suggesting that companies are reluctant to implement measurement because it is difficult to justify in terms of quick pay backs. Overall our research suggests that despite companies knowing that measurement is fundamental to SPI, it is rarely implemented effectively.

Improving Validation Activities in a Global Software Development

Christof Ebert[1]*, Casimiro Hernandez Parro, Roland Suttels, Harald Kolarczyk*
Alcatel, Switching and Routing Division, Antwerp, Belgium / Madrid, Spain / Stuttgart, Germany
[1]Alcatel, SD-97, Fr.-Wellesplein 1, B-2018 Antwerpen, Belgium
Tel.: +32-3-240-4081, Fax: +32-3-240-9935
christof.ebert@alcatel.be

Abstract. Increasingly software projects are handled in a global and distributed project set-up. Global software development however also challenges traditional techniques of software engineering, such as peer reviews or design meetings. Especially validation activities during development, such as inspections need to be adjusted to achieve results, which are both efficient and effective. Effective teamwork and coaching of engineers highly contribute towards successful projects. We will in this article evaluate experiences made in the last 3 years with validation activities in a global setting within Alcatel's Switching and Routing business. We will investigate 3 hypotheses related to effects of collocated inspections, intensive coaching, and feature-oriented development teams on globally distributed projects. As all these activities mean initial investment compared to a standard process with scattered activities, the major validation criteria for the 3 hypotheses is cost reduction due to earlier defect detection and less defects introduced. The data is taken from a sample of over 60 international projects of various sizes from which we collected all type of product and process metrics in the past 4 years.

A Generic Model for Assessing Process Quality

Manoranjan Satpathy[1]*, Rachel Harrison*[1]*, Colin Snook*[2]*, Michael Butler*[2]
[1]School of Computer Science, Cybernetics and Electronic Engineering
University of Reading, Reading RG6 6AY, UK
{M.Satpathy, Rachel.Harrison}@reading.ac.uk
[2]Department of Electronics and Computer Science
University of Southampton, Highfield, Southampton SO17 1BJ, UK
{cfs98r, mjb}@ecs.soton.ac.uk

Abstract. Process assessment and process improvement are both very difficult tasks since we are either assessing or improving a concept rather than an object. A quality process is expected to produce quality products efficiently. Most of the existing models such as CMM, ISO 9001/9000-3 etc. intend to enhance the maturity or the quality of an organization with the assumption that a matured organization will put its processes in place which in turn will produce matured products. However, matured processes do not necessarily produce quality products. The primary reasons are: (i) In the process quality models, the relationship between the process quality and product quality is far from clear, and (ii) many of the process models take a monolithic view of the whole life-cycle process, and as a result, the idiosyncrasies of the individual processes do not receive proper attention.

In this paper, we first define an internal process model in a formal manner. Next, we define a generic quality model whose scope covers all the development processes and most of the supporting processes associated with the development phase. The generic quality model is a parametric template and could be instantiated in a systematic manner to produce the quality model for any individual process. We then show such a customization for the formal specification process and use this customized model to formulate a GQM-based measurement plan for the same process. We then discuss how the generic model would be useful in process assessment and process improvement.

Maturity Evaluation of the Performance Engineering Process

Andreas Schmietendorf, André Scholz
University of Magdeburg, Faculty of Computer Science, Germany
schmiete@ivs.cs.uni-magdeburg.de,
ascholz@iti.cs.uni-magdeburg.de

Abstract. This contribution presents a model for process improvement in the area of performance engineering, which is called performance engineering maturity model. The use of this model allows the evaluation of the level of integration and application of performance engineering. It leans against the well-established capability maturity model from the software engineering institute. The model is based on a questionnaire catalog, which was transferred into a web based evaluation form. The results of this anonymous evaluation are analyzed in this contribution.

COSMIC FFP and the World-Wide Field Trials Strategy

Alain Abran[1], S. Oligny[1], Charles R. Symons[2]
[1]Software Engineering Management Research Laboratory
Université du Québec à Montréal
C.P. 8888, Succ. Centre-Ville
Montréal, Québec, Canada
Tel: +1 (514) 987-3000 (8900), Fax: +1 (514) 987-8477
abran.alain@uqam.ca
[2]Software Measurement Service Ltd., St. Clare's, Mill Hill
Edenbridge, Kent TN8 5DQ, UK
Tel: +44 (0) 1732 863 760, Fax: +44 (0) 1732 864 996
charles_symons@compuserve.com

Abstract. Building on the strengths of previous work in the field of software functional size measurement, the Common Software Measurement International Consortium (COSMIC) proposed a set of principles in 1998 onto which a new generation of functional size measurement methods could be built. The COSMIC group then published version 2.0 of COSMIC-FFP, in 1999, as an example of a functional size measurement method built on those principles. Key concepts of its design and of the structure of its measurement process are presented, as well as the strategy of its world-wide field trials.

Extraction of Function-Points from Source-Code

Harry M. Sneed (MPA)
CaseConsult, Wiesbaden
Software Data Service, Vienna
Software Engineering Service, Budapest
Harry.Sneed@T-online.de

Abstract. In spite of the efforts of the IFPUG group to standardise the counting of Function-Points, there is still a lot of room left for interpretation. This is especially true when it comes to counting Function-Points in modern client server or web-based applications. There is no standard means of identifying inputs and outputs in such systems. The author proposes here a tool supported method for extracting Function-Point counts from C++ and Java Source-Code. This method has been applied and calibrated to the GEOS Stock Brokerage system under development in Vienna, where the author is currently engaged.

Early & Quick COSMIC-FFP Analysis using Analytic Hierarchy Process

Luca Santillo
Data Processing Organisation, 00196 Roma, v. Flaminia, 217, Italy
Tel.: +39-06-3226887, Fax: +39-06-3233628
luca.santillo@iol.it

Abstract. COSMIC-FFP is a rigorous measurement method that makes possible to measure the functional size of the software, based on identifiable functional user requirements allocated onto different layers, corresponding to different levels of abstraction. The key concepts of COSMIC-FFP are software layers, functional processes and four types of data movement (sub-processes). A precise COSMIC-FFP measure can then be obtained only after the functional specification phase, while for forecasting reasons the Early & Quick COSMIC-FFP technique has been subsequently provided, for using just after the feasibility study phase.
This paper shows how the Analytic Hierarchy Process, a quantification technique of subjective judgements, can be applied to this estimation technique in order to improve significantly its self-consistency and robustness. The AHP technique, based on pair-wise comparisons of all (or some of) the items of the functional hierarchical structure of the software provided by E&Q COSMIC-FFP, provides the determination of a ratio scale of relative values between the items, through a mathematical normalization. Consequently, it is not necessary either to evaluate the numerical value of each item, or to use statistical calibration values, since the true values of only one or few components are propagated in the ratio scale of relative values, providing the consistent values for the rest of the hierarchy.
This merging of E&Q COSMIC-FFP with AHP results in a more precise estimation method which is robust to errors in the pair-wise comparisons, and self-consistent because of the redundancy and the normalization process of the comparisons.

Measuring the Ripple Effect of Pascal Programs

Sue Black, Francis Clark
Centre for Systems and Software Engineering, South Bank University,
103 Borough Road, London SE1 0AA, UK
Tel.: ++44(0)702 815 7471
blackse@sbu.ac.uk
clarkfg@hotmail.com

Abstract. Recent acquisition of a half million LOC telephone switching system TXE4 written in Pascal has provided a unique opportunity for software measurement. This paper discusses the software implementation of ripple effect measure - REST (Ripple Effect and Stability Tool) focusing on a recent attempt to produce a Pascal parser for REST which will be used to measure the TXE4 system. Ripple effect is a measure of impact analysis: the effect that a change to one part of a system will have on other parts of a system. It can be used in software engineering development to compare different versions of software or during maintenance to highlight software modules which may need attention. The implementation of the Pascal parser has highlighted several significant differences between Pascal and C source code, which are discussed and investigated.

An Assessment of the Effects of Requirements Reuse Measurements on the ERP Requirements Engineering Process

Maya Daneva
Clearnet Communications, 200 Consilium Place, Suite 1600
Toronto, Ontario M1H 3J3, Canada
mdaneva@clearnet.com

Abstract. Setting realistic expectations for a requirements measurement exercise and assessing the real benefits resulting from the implementation of metrics in Requirements Engineering (RE) is a key challenge for many information systems (IS) organizations. This paper describes how a project team can demonstrate a connection between efforts invested in requirements reuse measurement and business results in Enterprise Resource Planning (ERP) projects. We provide an approach to analyzing and assessing the benefits gained from integrating requirements reuse measurement practices in the ERP RE process. Dependencies between requirements measurement activities and RE activities and deliverables are studied in the context of SAP R/3 implementation projects.

A New Metric-Based Approach for the Evaluation of Customer Satisfaction in the IT Area

Reiner R. Dumke, Cornelius Wille
University of Magdeburg, Faculty of Computer Science, Postfach 4120,
D-39016 Magdeburg, Germany
Tel.: +49-391-6718664, Fax: +49-391-6712810,
{dumke,wille}@ivs.cs.uni-magdeburg.de

Abstract. For the existence and growth of enterprises the protected sales of goods and performances are of decisive opinion. In order to ensure this, the acceptance of the products and performances by the customers is depended on indirect criterion mainly: the customer satisfaction. A lot of criteria and methods with help of the science were or are worked out to "measure" satisfaction or discontent of the customers to the inquiry.
Our paper describes the general satisfaction aspects and their coherence between customer satisfaction, quality and customer loyalty as well as their significance for the development of an enterprise. After the classification of methods for measuring customer's satisfaction different methods for the recording of customer satisfaction are shown. A basic model for customer satisfaction is introduced for recording and assessment at software products using software metrics related to the product, process and resources aspects. This method attemps to measure directly the causes to evaluate their effect to the customer satisfaction.
In order to evaluate customer satisfaction, a tool COSAM was implemented that allows a metrics-based assessment besides the traditional assessment of the customer satisfaction by customer interviews. On the other hand, the tool can be used for experiments of a given level of customer satisfaction to analyse the effects of successful measured aspects such as ISO 9000 certification, a high level of the developer skills or a high level in the CMM evaluation.

Utility Metrics for Economic Agents

Dirk Schmelz[1], Margitta Schmelz[1], Julia Schmelz[2]
[1]Thüringer Kompetenzzentrum eCommerce
tranSIT GmbH Ilmenau, Germany
c/o Friedrich-Schiller-Universität Jena, Germany
mms@informatik.uni-jena.de
[2]Technische Universität München, Germany
schmelz@mathematik.tu-muenchen.de

Abstract. In this paper, a metric view for operating software agents is developed and explained by way of an example of economic trader agents utility. Here, the role of simulation models as a helpful technology for construction and validation of agents in artificial environments is propagated.

QF^{2}D: a Different Way to Measure Software Quality

Luigi Buglione[1], *Alain Abran*[2]
Software Engineering Management Research Laboratory
Université du Québec à Montréal
C.P. 8888, Succ. Centre-Ville
Montréal, Québec, Canada
[1]Tel: (39) 338.95.46.917, Fax: (39) 06-233.208.366
luigi.buglione@computer.org
[2]Tel: +1 (514) 987-3000 (8900), Fax: +1 (514) 987-8477
abran.alain@uqam.ca

Abstract. Quality Function Deployment (QFD) technique has been developed in the context of Total Quality Management, and it has been experimented in the software engineering domain. This paper illustrated how key constructs from QFD contributed to an development of a second version of a Quality Factor (QF) for a qualitative software evaluation, considering three distinctive but connected areas of interest, each of them representing dimension of performance:

- economic dimension, the perspective is the managers' viewpoint;
- social dimension, the perspective is the users' viewpoint;
- technical dimension, the perspective is the developers' viewpoint.

This new version of the original QF technique, referred to as QF2D (Quality Factor through QFD), has the following features: it can be used for both a priori and a posteriori evaluations of the software product; it makes usage of the set of quality sub-characteristics proposed in the new upcoming ISO/IEC 9126:2000 standard it has a variable number of elements taken into account the three viewpoints for the evaluation; it offers the visual clarity from QFD for external and internal benchmarking. An implementation of this new version of this technique in quality models is also discussed.

Using FAME Assessments to Define Measurement Goals

Dirk Hamann[1], *Andrew Beitz*[1], *Markus Müller*[2], *Rini van Solingen*[1]
[1]Fraunhofer IESE, Sauerwiesen 6, Technopark II, 67661 Kaiserslautern, Germany
[2]Fraunhofer IESE, Luxemburger Str. 3, 67657 Kaiserslautern, Germany
{hamann, beitz, markus.mueller, solingen}@iese.fhg.de

Abstract. Although assessment-based approaches and measurement-based approaches are often considered as competitors, they compliment each other very well. Assessments are strong in identifying improvement objectives within a relatively short time frame, but are weak in guiding the actual implementation of the proposed changes. Measurement, however, supports very well in supporting actual changes and providing feedback on the effects of these changes, but has a difficulty with selecting the right goals. In this paper, we suggest an approach in which focused assessments are used to identify improvement goals and to use goal-oriented measurement to guide the implementation of the actual changes.

Mapping Processes Between Parallel, Hierarchical and Orthogonal System Representations

Francis Dion[1], Thanh Khiet Tran[2], Alain Abran[3]
[1] Epsilon Technologies inc., 1200, Boul. Chomedey, Laval (QC) Canada H7V 3Z3
fdion@xpertdoc.com
[2] tkhiet@yahoo.com
[3] Professor and director of the Research Lab. in Software Engineering Management
Université du Québec à Montréal, Département d'informatique, C.P. 8888, Succ. Centre-ville
Montréal (Québec), Canada H3C 3P8
abran.alain@uqam.ca

Abstract. The importance of software system representation through models and visual diagrams is increasing with the steady growth of systems complexity and criticality. Since no single representation is best suited to address all the documentation, communication and expression needs of a typical software development project, the issues related to conversion and coherence between different representations are having a significant impact on team productivity and product as well as process quality. This paper explores the types of relationships that exist between representations and the impact they have on mapping, generation and synchronization processes. We propose a characterization of those relationships as being parallel, hierarchical or orthogonal. Examples and comments on mapping or transformation processes and automation prospects in the context of software size measurement are also provided.

Analyzing Software Design using a Measurable Program Design Language

Nadine Hanebutte[1] and Reiner R. Dumke[2]
[1]University of Idaho, Dept. of Computer Science, Moscow, ID 83843, USA
hane@cs.uidaho.edu

[2]Otto-von-Guericke-University of Magdeburg, Dept.of Computer Science,
Postfach 4120, D-39016 Magdeburg, Germany,
Tel.: +49-391-6718664, Fax: +49-391-67-1812810
dumke@ivs.cs.uni-magdeburg.de

Abstract. To estimate the quality or the number of faults in a future source code, design metrics are generated from the design documentation written in a Program Design Language. The document is analyzed for defined tokens and structures. Their occurrences or order is counted using previously defined unambiguous metrics. Metrics are highly correlated. Therefore the structure of the observed measures is to be analyzed and processed to be valid input to further analysis like multi-regression models and hypothesis testing for prediction of external software attributes.

Keywords: design metrics, fault estimates, software quality maintenance, Program Design Language

1 Introduction

The key to any attempt to make changes is the ability to measure the effort of those changes. There are about 200 software metrics in use (Munson and Khoshgoftaar, 1993). But only a subset of them are used in a particular software development project or a particular project stage. Measures are known for all stages of the development process as project, process or

environment measures. Software metrics are mostly applied as product measures during the implementation, usually because other measurable or structured information about the software are not available.

1.1 Recent Works

The number of measures that can be extracted from a project design depends on the degree of the availability of documentation of the design of a software. According to this degree there are low-level (or architectural) and the high-level (or detailed) design. Different metrics are available for each level.

During the low-level design phase extracted measures are based on information like hierarchical module diagrams, data flow, functional and interface description.

One of the suggested metrics for this phase is D_e, which provides information about the necessity of redesign after an outlier analysis. (Zage, 1993)

One other approach is the mapping of the cyclomatic complexity approach (McCabe, 1976) from implementation onto design. (McCabe, 1989) While the metrics of low-level design can only capture some of the project attributes, depending on the used tool (Swann, 1978), it is possible to map almost all measurement strategies for source code onto the detailed design. But to extract metrics from design documentation, it is necessary to use a suitable description language. (Heitkoetter et al., 1990) There is no general solution for extending software metrics onto design because of the wide range of tools that can be used for the documentation of design (Oman and Curtis, 1990). Because of the different level of design details that can be visualized with different tools a mapping metrics from different tools is sometimes not possible.

Additionally to selecting the metrics to be applied one has to model the relation of these measures to the things that one wants to know: How faulty is the future code going to be? Is it from the intented quality? Based on the answer the best design can be selected. (Shepperd and Ince, 1989)

There are different ideas on how to find the relation between measured numbers and the formulation of the answers, i.e. factor analysis (Coupal and Robillard, 1990; Munson and Khoshgoftaar 1992) or outlier analysis (Shepperd and Ince, 1989; Zage, 1993].

1.2 Product Measurement during Software Design

The point of measuring software is to retrieve a number of quantities that enable us to make statements about the quality of a piece of documentation or code or to perform a comparison between different solutions for the same problem.

Many of the commonly used measures are directly or indirectly dependent on each other. For instance, increasing the lines of code (LOC) is accompanied by an increase in the number of statements. Splitting a module in smaller piece would reduce the number of statements but would shift the problem to an other measuring domain - coupling, because the amount of exchanged variables between the new modules would be increased. Regarding this, it is useful to analyze the applied set of metrics about their dependencies and develop an understanding for the nature of those metrics.

Mostly source code is measured to extract quality information. But it is possible to measure other steps in the software development process like the design, as well. Design measurement lets us capture important aspects of product and process early in the software development life cycle, so that corrective actions can be taken earlier (Rombach, 1990). For doing so it is necessary to create well-structured design documentation.

1.3 The University-of-Idaho-Design-Language (UofI-DL)

The design language was developed to describe the algorithm of future software as well as its data structure. Furthermore, as part of the project documentation, it should link the different parts of the design project together with the later product – the source code. It consists of a verbal description of a module, information about its place in the calling structure and a reference of all used variables, types and constants. For the algorithm - called the Precode - a keyword set is used to describe the problem by providing structures like loops, sequences, selections, calls and simple equations. Any separate design document to describe a piece of code is called a module. A module written in UofI-DL has the form as in Figure 1.

The structure and keyword set have the advantage of being measurable, to quantify the design results on module level. For each module the design documents includes a description about its place within the calling hierarchy and information about the data flow to and from other modules. This allows to capture metrics at module-level which are usually only measurable at project-level when all source code files, in particular the definition of all functions, types, etc., are available.

An experiment was conducted where sixteen metrics were extracted from the design of 48 programs or program parts written in UofI-DL. Therefore sample algorithms were taken, from books on algorithms in C or PASCAL as well as from older design projects, to capture a wide variety of algorithm types. Since the general applicability of these metrics is to be analyzed and the goal is to draw conclusions that are valid beyond this experiment, the randomization of the input data is required (Wohlin et al., 2000).

The design documents have been written for the chosen algorithms. The design of a module is stored as ASCII text. A syntax check is performed with each file to ensure a valid input to the measuring process. Thereafter the text is analyzed for the appearance or order of certain tokens. For each metric an unambiguous definition was made to ensure the repeatability of the experiment. These definitions were implemented in a metric tool.

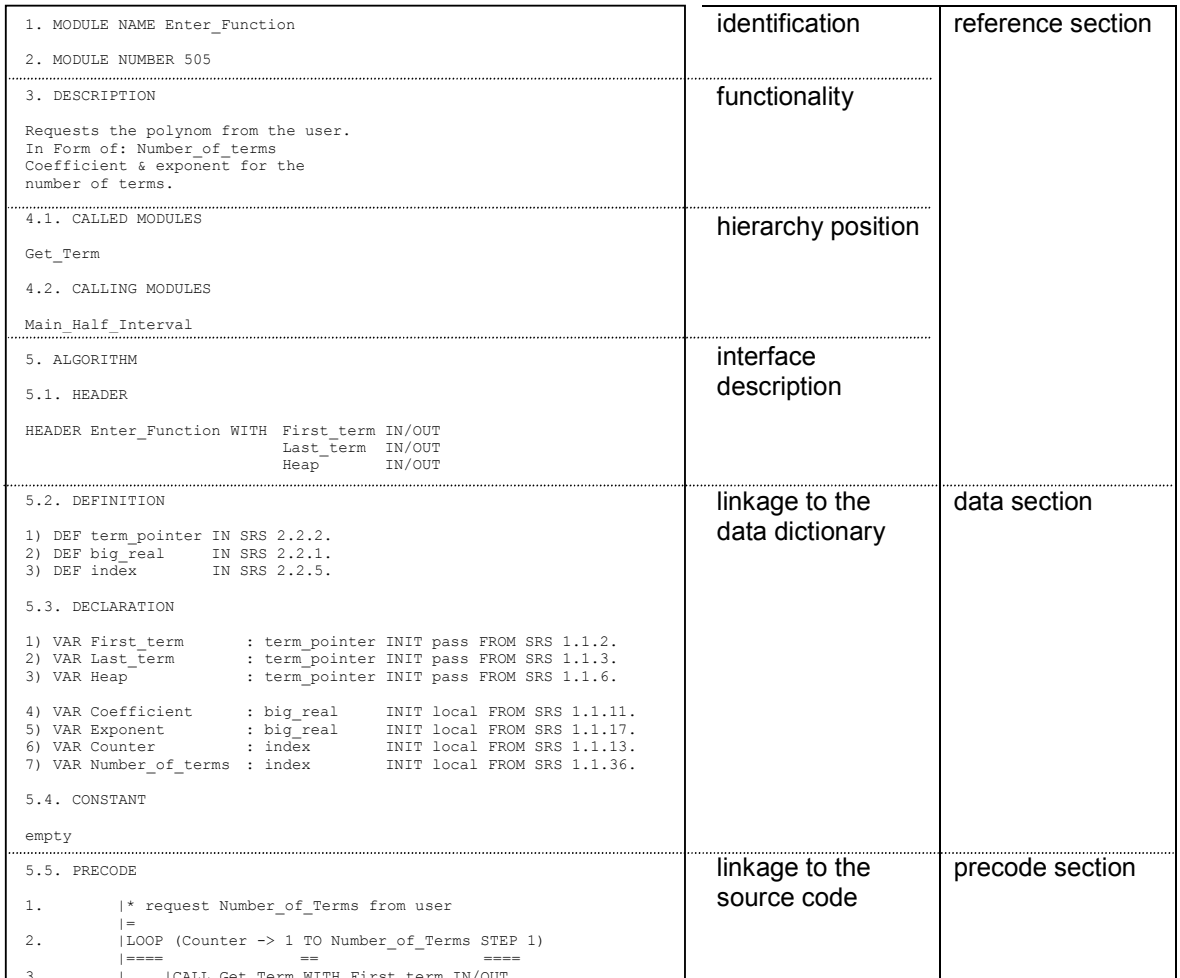

Figure 1: Example for the UofI-DL Module Structure

2 The Design Measurement Approach

Using the measurement tool the following six-teen measures have been extracted from the module:

Nodes, Edges, Cycles, Variables, Types, Constants, Statements, Maximum Nesting, Returned Arguments, Number of Paths, Maximum Path Length, Average Path Length, Calling Modules, Called Modules, Incoming Variables, Outgoing Variable.

Those measures are supposed to capture the most influence factors to the introduction of faults. All chosen metrics are simple ones, captured by counting up certain attributes of design documentation:

Variables, Constants and Types
The idea of measuring variables, constants and types is taken from the Halstead metric (Zuse, 1997). Halstead identified the number of operands as a characteristic mark for the programming effort. The approach is mapped onto the declaration and definition part of a module. The types are seen as the unique operands. Accordingly, the variables and constants are the total number of operands, and can thus be grouped by their types.

Incoming and Outgoing Variables, Returned Arguments
The coupling increases directly with the complexity of a module interface (Troy and Zweben, 1993). The mediums of the coupling are the passed data and the return arguments named in the module header. Those are measured in numbers of incoming and outgoing variables.

Called variables are manipulated outside. Outside means that according to the status of a module and the setting of the arguments, a variable of this module is changed inside of another module. The manipulation of data by other modules than the one where it is defined in

the first place and the passing of information is discussed in (Yux and Lamb, 1995) by Henry and Kafura. Their information flow metric for design accounts the fact that the complexity increases with extending the data flow. Chapin recognized with his Q-metric for design the different kinds of variables and the necessity to differ between them (Yu and Lamb, 1995).

Calling Modules
An external design metric D_e as suggested by Zage (Zage, 1993) is based on data flow and call structure. This composite metric includes fan-in, fan-out, inflow and outflow.

Called Modules, Nodes and Edges
McCabe used connected components, nodes and edges as base for his cyclomatic complexity (McCabe, 1976). There are doubts on the usefulness of the McCabe metric itself (Dumke and Foltin, 1999). But these three measures are commonly identified as the influence factors for the algorithm complexity. A similar approach is introduced by McClure (Yu and Lamb, 1995), which is calculates the modules complexity in design from the invoking and invoked modules and the according control structures.

Statements
The LOC is probably one of the oldest and most often used metrics (Zuse, 1997). There are several definitions on how to count lines of code. Because formatting plays a major role in the editing of a well-structured design document, simply counting the new lines would not give a comprehensive result. The number of complete statements in the algorithm should therefore be counted.

Paths
McCabe came to the conclusion that a cyclomatic complexity of ten should be the maximum for a maintainable program. Assuming that maintainability and complexity are dependent, there is a maximum path number before an algorithm becomes to complex. The cyclomatic complexity is the number of linear independent paths in a program. Therefore it can be concluded that the number of paths is a significant measure of software.

Maximum and Average Path Length, Maximum Nesting and Cycles
Those are measures on how the statements are distributed in the different algorithm paths. Complexity of the control structures can be used to weight the information available on fan-in, fan-out, inflow or outflow.

These metrics can be roughly divided into two groups: one capturing the attributes related to the module's size, the other one the communication between a module and other ones.

3 The Software Measurement Exploration

3.1 Analysis and Statement of the Results

After measuring all 48 modules, the result is in the format of 48 x 16 matrix containing 48 vectors of the 16 metrics. It is difficult to draw any immediate conclusions or even make predictions from the amount of different numbers with different meanings. For the analysis process the aim is to place all measured numbers in a multi-regression model (Equation 1) to approximate external metrics. With this type of model the actual process of discovering the relationship between internal and external metrics starts. The model than is to be verified using hypothesis testing methods, e.g. the t-test (Wohlin et al., 2000).

Eq. 1: $y=b_0m_0+b_1m_1+b_2m_2+...+b_im_i+c$

This model is based on the assumption, that the independent variables of the analysis are not linear compounds of each other, nor share an element of common variance. But usually software metrics do not own this features (Munson, 1995). All metrics are more or less correlated with each other. As a result it can not be determined how the effect of a change to one metric effects the other ones. The measures as they are, can not be used for any prediction of the future software systems attributes immediately. For example, an increase in LOC can cause the definition of new variables or maybe a reduction in the amount of needed variables.

Under the assumption that the measures have an underlying structure and that some of them are highly correlated, factor analysis and principal component analysis are performed on the z-scores of the measures to explore the nature of this underlying structure.

If there is a structure, it can be concluded that instead of 16 dependent metrics there is a smaller set of independent metrics represented by the factors, each combining a number of the original measures, that can be used to describe each module.

To analyze the matrix's structure the eigenvalues, as representation of the variance within the input matrix, are calculated. According to the Kaiser

Criterion[1] five factors would be extracted, according to the scree-test[2] four. Because in case of five factors the result include a trivial factor[3] four was chosen.

This leads to a grouping of the input measures. According to their correlation to one single factor and by analyzing the corresponding input metric a domain is defined that describes the nature of the grouped metrics.

FACTOR	ONE	TWO	THREE	FOUR	
NODES	0.96276	0.05329	-0.00660	0.12768	
EDGES	0.93830	0.10428	0.01844	0.23732	
CYCLES	0.42570	-0.06092	-0.10061	0.75005	
VARIABLES	0.66313	0.27124	0.39342	0.19673	
TYPES	0.40250	0.50972	0.20342	0.29158	to be removed
CONSTANTS	0.21945	0.14471	-0.61806	-0.33772	to be removed
STATEMENTS	0.93411	0.09083	0.02708	-0.03540	
MAXIMUM NESTING	0.54965	0.50932	0.25872	0.18160	
CALLING MODULES	0.04399	0.34559	0.62539	-0.15196	
CALLED MODULES	-0.05742	0.94406	0.01214	-0.03235	
INCOMING VARIABLES	0.06384	0.18414	0.84762	0.06090	
OUTGOING VARIABLES	0.05313	-0.01726	0.51501	-0.42189	
CALLED VARIABLES	0.19073	0.85324	0.13907	0.03079	
NUMBER OF PATHS	0.29814	0.17591	0.12566	0.73327	
MAXIMUM PATH LENGTH	0.90907	0.12056	-0.09052	0.22046	
AVERAGE PATH LENGTH	0.87933	0.05807	-0.10244	0.18891	
Domain	Length	Fan-Out	Fan-In	Width	

Figure 2: Rotated Factor Pattern; Four Factors selected

[1] Number of factors equals the number of eigenvalues greater than one.

[2] Plot of the eigenvalues. Plot is observed for "jumps": Number of factors equals number of eigenvalues before the change in continuity.

[3] This is a factor with only one member from the original set.

The metrics are expected to group according to their nature and can therefore be used to characterize each domain.

Metrics which do not group conclusively need be removed for this analysis to reduce their influence of their noise to the analyzed data, since these metric do not help to draw conclusions about the modules. In the first factor nodes, edges, variables, statements, maximum nesting, maximum and average path length are grouped which are all measures of the size of a program. The longer the source code "text" is, the higher is the count of these metrics. The second factor includes called modules, called variables and types. The first two are the number of other programs that are called by the measured one and the variables that are passed to them; manipulated within this other module and are returned with its value changed to the calling one.

Both are highly correlated to the factor with values from above 0.8. The third variable types seems to have nothing in common with the other two. The usage and definition of types is a question of programming style of the programmer and is sometimes not even supported by the programming language. It is possible to write two almost exact program, by using self-defined types in one of them and only standard types in the other. Therefore it is assumed that number of types is more a measure of development strategies than of code/design itself. The modules were written without a restriction on the usage of predefined or self-defined types. Therefore metric types is removed from the data set and will not be included in further analysis.

The third factor shows a similar problem. Calling modules, incoming variables and outgoing variables, which are measures of the communication flow from other modules are in the same group as constants. As for types in the last factor the number of constants seems to be again more a measure of a writing style than the content quality. Constants can be substituted by variables completely as long as the programmer respects the restriction not to change the value once it is set. Constants is removed from the analyzed set.

The last factor with number of path and cycles measures the complexity of the control flow or the amount of different ways from the beginning of the program to its end.

These observations lead to the following conclusions: The number of factors in the future analysis is four. Variables types and constants are removed. A naming convention for each factor according to the grouped metrics can be introduced.

- Name of domain one: *Length*
- Name of domain three: *Fan -in*
- Name of domain two: *Fan- out*
- Name of domain four: *Width*

Size		Communication	
Length	Width	Fan-Out	Fan-In
Nodes *Edges* *Variables* *Statements* *Maximum Nesting* *Maximum Path Length* *Average Path Length*	*Cycles* *Paths*	*Called Modules* *Called Variables*	*Calling Modules* *Incoming Variables* *Outgoing Variables*

Table 1: The Structure of the Metrics

The factors are the metric domains. Domain length and width are module size domains. Domain fan-in and fan-out are module-communication domains. Therefore there are two super domains: size and communication. The metrics structure as in Table 1.

The correlation between the domains is overall low, while the correlation in-between the size related domains and in-between the communication related domains is slightly higher while it is extremely low between the size domains and the two communication domains.

The super domains are independent. The correlation between the communication related domains is very low. The correlation between two remaining domain is to high for absolute independence but to low to actually conclude dependence. The domains can therefore be treated as if they were independent, which was not possible with initial metrics (Table 2.)

There are several ways to continue with the goal to scale the modules and to describe their likelihood to fail with one single number for each module.

	Domain length	Domain fan-out	Domain width	Domain fan-in
Domain length	1.00	0.27	0.52	0.12
Domain fan-out	0.27	1.00	0.09	0.28
Domain width	0.52	0.09	1.00	-0.02
Domain fan-in	0.12	0.28	-0.02	1.00

Table 2: Correlation of the Domains

One would be to add up the scores, another to weight each score with the corresponding eigenvalue as in equation 2.

$$\textbf{\textit{Eq. 2:}}\ \textit{Fault index} = f_L * \square_L + f_{FO} * \square_{FO} + f_{FI} * \square_{FI} + f_W * \square_W$$

To be able to compare the fault indices of one project with others the indices should be standardized. Usually the modules with indices outside the standard deviation are analyzed with the target of redesign. Static limits can be defined which are adjusted as knowledge about the nature of the index emerges. If there are no outliers, the modules closest to the limits are analyzed, mainly to see if the limits are valid. If extreme outliers occur, the X-less algorithm could be used (Zage, 1993).

Further analysis could be done to relate this number to quality or number of faults for example to estimate coefficients for an univariant relation. If this relation could be estimated, the limits could be refined, according to the numbers of faults. I.e. if no faults occur to an index of 0.5 this could be the upper limit.

3.2 Result Interpretation

The factor scores are computed to structure all information and ease the observation process. The domains as well as the fault index are understood as an indicator of problems. There are no fixed numbers on limit values for the index or the domains yet. The numbers are a ranking system. If the relation to an external metric like number of faults could be estimated, this metric can be used for qualifying and defining static limits. The interpretation of the fault index is context dependent. For each domain the highest and the lowest scores are shown in the Tables 3 to 6. The meaning and conclusion drawn from each score are different within each domain.

Module 002 - maximum	Module 113 - minimum
Cycles 10 Paths 5032	Cycles 0 Paths 1

Table 3: Maximum and Minimum Results for Domain Width

3.2.1 Domain Width

The reason for a high score in width of a program is a high amount of decisions that have to be made when tracing the algorithm, such as in constructs like loops and decision statements. An increase in the number of selections leads to the increase of the number of paths in a module. A high number of paths indicates that there is a good chance, that some of those paths will never be passed or that this module is faulty, because the high number of conditions often have hard to predict dependencies. A closer look should be taken at modules, which score high in this domain. They are good candidates for alternative solutions. Modules with a high number of paths are almost unstestable, because there is no chance to cover all paths. A lower score in this domain does not necessarily mean the module is of better quality, but it indicates, that this module can be tested easier.

3.2.2 Domain Length

Module 002 - maximum		Module 504 - minimum	
Nodes	52	Nodes	9
Edges	75	Edges	10
Variables	17	Variables	2
Statements	45	Statements	5
Maximum Nesting	7	Maximum Nesting	1
Maximum Path Length	31	Maximum Path Length	5
Average Path Length	22,27	Average Path Length	3,00

Table 4: Maximum and Minimum Results for Domain Length

Compared with the entire measured modules module 002 has the almost highest score in number of nodes, edges and statements (Table 4). A high number of statements have to be executed every time the source code to this module is in use. Additionally, a high number of variables is used. A module with a high score in the length domain will have a high amount of memory in use. The module with the lowest score in length was 504. So it can be expected, that this module has very low space usage and the number of executed commands per module call is small. Indeed show those numbers that this module is really "small" with just three average executed statements and two variables.

3.2.3 Domain Fan-Out

Data about external factors of influence to one module like called modules and called variables are collected in this domain (Table 5).

Module 114 - maximum		Module 101 - minimum	
Called Modules	6	Called Modules	0
Called Variables	11	Called Variables	0

Table 5: Maximum and Minimum Results for Domain Fan-Out

The module 114, the one with the highest score in this domain has the highest number in those two raw measures by far. This could mean that the granularity is already too high and tasks from called modules with a low fan-in could be included.

The module (101) with the minimum score in this domain does not call any other modules. This module is taken as an example since there is more that one module that has the same score in this domain and since they all do not call sub routines. Therefore there is no data return to this module either.

A minimum in this domain indicates a one-block program with no modularity, when there is a very small number in the fan-in domain, too.

3.2.4 Domain Fan-In

Module 005 - maximum	Module 900 - minimum
Calling Modules 4	Calling Modules 0
Incoming Variables 4	Incoming Variables 0
Outgoing Variables 1	Outgoing Variables 0

Table 6: Maximum and Minimum Results for Domain Fan-In

A high score in the fan-in domain it could mean, that the module stands in the lowest level in a calling hierarchy or it is to universal. It might be solving to many different tasks and should be split into subtasks. If too many other modules are using this one, the task it is solving is either often repeated and important or it is representing more than one functionality, which should lead to a redesign with a higher granularity. A low score in this domain means that the functionality represented by this module is almost not used and a too high granularity is indicated or this module stands on the top of the calling hierarchy, which should be accompanied by high score in fan-out. Module 900 seems to be of the one block design type, having a low granularity, since its fan-out score is one of the lowest. This can indicate a monolithic programming style.

4 Summary and Conclusions

The suggested design language allows a comprehensive way of writing design documents. Its major advantages are its structure that allows to link design documents together with other project part's documentation and the fact that it can be measured with metric similar to the one often applied to source code. Therefore first quality statements can be made about the future software before the first line of code is written. To put a meaning to the measured numbers the metrics are analyzed using factor analysis. This leads to a grouping of the metrics and allows a first interpretation of the results. The metrics group in a way as one would group them intuitively. The grouping allows a reduction and therefore eases further analysis. A module can therefore be described in its length, its width[4], its fan-out, as the data sent from

[4] Width in the sense of complexity. One can imagine this as all the different ways across the module next to each other.

the module, and fan-in, as the data send to this module from others. This allows quality statements early during the development not only in terms of likelihood to fail, but also for functionality splitting and granularity, which are more design than code issues. Furthermore each module can be measured independently from all others.

References

Coupal, D. and Robillard, P.N., Factor Analysis of Source Code Metrics. *The Journal of Systems and Software*. v12, 263 -269 (1990)

Dumke, R.R. and Foltin, E., An Object-Oriented Software Measurement and Evaluation Framework. *Proceedings of the FESMA '99*, Amsterdam, 59 -68 (1999)

Heitkoetter, U.,et al, Design Metrics and Aids to Their Automatic Collection. *Information and Software Technology*. v32 n1, (1990)

McCabe, T.J., A Complexity Measure. *IEEE Transactions on Software Engineering*. vSE2 n4, (1976)

McCabe, T.J. and Butler, C.W., Design Complexity Measurement and Testing. *Communications of the ACM*. v32 n12, (1989)

Munson, J.C. and Khoshgoftaar T.M., Measuring Dynamic Program Complexity, *IEEE Software*. v11, 48-55 (1992)

Munson, J.C. and Khoshgoftaar, T.M., Measurement of Data Structure Complexity. *The Journal of Systems and Software*, 20, 217-225, (1993)

Munson, J.C., Software Measurement: Problems and practice. *Annals of Software Engineering*. 1, 255-285 (1995)

Oman, P.W. and Curtis, R.C., Design and Code Traceability Using a PDL Metrics Tool. *The Journal of Systems and Software*. v12, 189-197 (1990)

Rombach, D.H., Design Measurement: Some Lesson Learned. *IEEE Software*. v3, 17-25 (1990)

Shepperd, M. and Ince, D., Metrics, Outlier Analysis and the Software Design Process. *Information and Software Technology*. v31 n2, 91-98 (1989)

Swann, G.H. *Top-Down Structured Design Techniques*, Petrocelli Books, New York, Princeton, 1978, p. 17-33

Troy, D.A. and Zweben, S.H., Measuring the Quality of Structured Design, *in Software Engineering Metrics. - Vol. 1: Measures and Validations* (Sheppard, M. ed.), McGraw-Hill Book Company, 1993

Wohlin, C., Rueson, P., Höst, M., Ohlsson, M.C., Regnell, B. and Wesslen, A., *Experimetation in Software Engineering: An Introduction*, Kluwer Academic Publishers, 2000

Yu, X. and Lamb, D.A., Metrics applicable to software design. *Annals of Software Engineering*. 1, 23-41 (1995)

Zage, W.M and Zage D.M., Evaluating Design Metrics on Large-Scale Software. *IEEE Software*. v7, (1993)

Zuse, H., *A Framework of Software Measurement*, DeGruyter Publisher, Berlin, 1997

Software Reuse and Metrics within a Process Model for Object-oriented Development

Evgeni Dimitrov[1], Andreas Schmietendorf[1,2], and Reiner Dumke[2]
[1]T-Nova, Deutsche Telekom Innovationsgesellschaft mbH, Entwicklungszentrum
Berlin, Wittestraße 30N, 13509 Berlin
Evgeni.Dimitrov|A.Schmietendorf@telekom.de
[2]Otto-von-Guericke-Universität Magdeburg, Fakultät Informatik,
Institut für verteilte Systeme, Postfach 41 20, D-39016 Magdeburg
schmiete|dumke@ivs.cs.uni-magdeburg.de

Abstract: *Like other engineering disciplines, the development of industrially viable information systems requires a procedure that can be planned and followed. A procedural model for object-oriented software development is presented, which also particularly takes account of software reuse and the use of metrics. The appropriate project-specific procedural model can be derived from this procedural model for every concrete object-oriented software project. In addition, initial concepts and implementations of tool support for this procedural model are presented, and general experience in practice is described.*

Keywords: *Metrics, object orientation, tools, reuse, procedural model*

1 Introduction

In the context of "software production", the process of software development is determined, throughout all its phases, by the extreme complexity of the product to be created, the need to design the entire system for different work phases (which may also be geographically scattered) and the high level of quality and safety requirements that have to be incorporated. This process, which is typically carried out in phases, must be duly planned, controlled and analyzed [Dumke 1993], and questions of economic viability, quality assurance and compliance with regulations must also be incorporated.

Within Deutsche Telekom AG, a standard procedural model based on the V model is used for the development and maintenance of complex software systems [VM Basis 1996]. This non platform related description for the development work itself and for the form in which the results are presented has been successfully used for years. Figure 1 shows the phases of the VM Basis graphically.

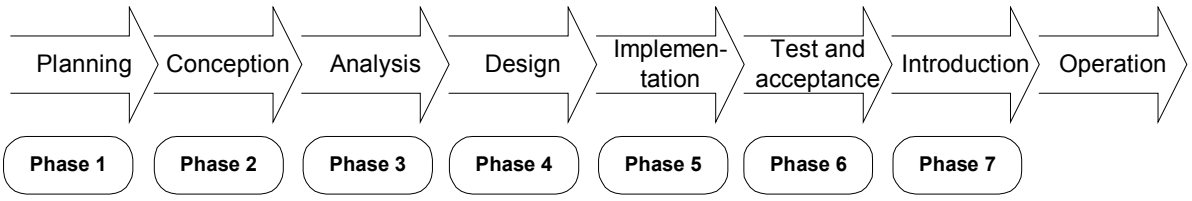

Figure 1: **Deutsche Telekom AG's procedural model [VMBasis 1996]**

As with other engineering disciplines, the added value of applying procedural models comes from the use of tried and tested and, as far as possible, standardized methods, notations and tools for software development. This ensures, amongst other things, that reliable, continuously

improved methods are used for development, that sufficient documentation (including documentation of the uniform structure) is available for further development and maintenance, that the process of development can be planned as a whole and the input required can be assessed and that regulations/laws can be taken into account. In addition, large projects can be structured on the basis of structural levels and specific quality assurance becomes possible.

2 The aims of the object-oriented procedural model

The procedure for software development, i.e. the definition of the software development process, is depicted on the basis of descriptions and instructions through structuring from various viewpoints as a model or procedural model and is thus made transparent and plannable.

A procedural model [Biskup 1996]

1. defines:
   - a role model,
   - rules for the areas of activity of software system development, project management, configuration management and quality management,
2. prescribes methods and tools that support the preparation of results.

However, the standard procedural model used by Deutsche Telekom, VM-BASIS, does not take the object-oriented concepts into consideration and can therefore not be used effectively in the realization of OO projects. To make this possible, the project experience existing in DTAG's Berlin Development Centre in this area was combined to form a **generic** procedural model for OO development (***genVMOO***) [Dimitrov 1998] and made available to all future projects. The idea was to develop a procedural model based on the Unified Process (UP) which would take into account, in particular, the characteristics of VM-BASIS in relation to structural levels and notation. In addition, the following additional objectives were pursued:

- Defining the use of metrics for the quantitative and qualitative evaluation of the resources used, necessary processes and the actual product.
- Successive support of the procedural model by appropriate tools so that an efficient application is guaranteed.
- Consideration of assets for comprehensive reuse of the software and linking of the processes necessary for this with those of the development process.

In order to be able to support different project types in OO development, such as business process and data-driven development or prototyping, a **generic VM-OO** (reference model) is taken as a basis. The generic VM-OO serves as a framework for deriving *project-type-specific* procedural models on the basis of process patterns ([Coplien 1995]) and/or using company specifications ([Noack 1997]). Through the more specific formulation of marginal conditions and general project specifications, further *project-specific* modifications (known as 'tailoring') can be undertaken (Figure 2).

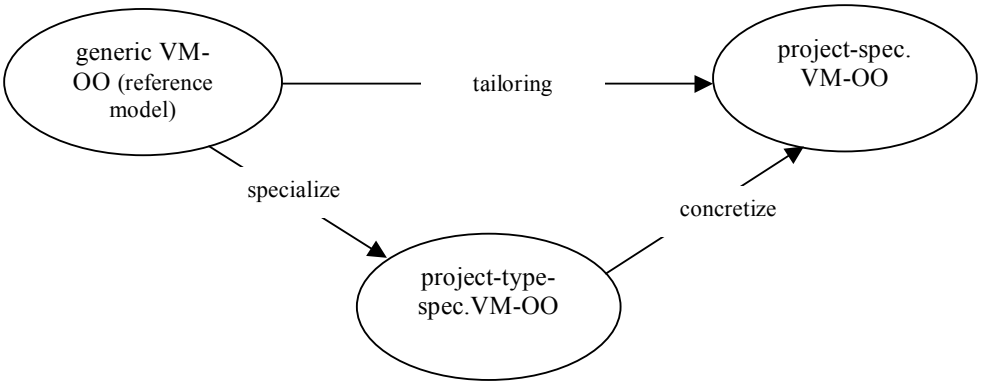

Figure 2: **Classification of OO procedural models**

3 Generic procedural model for object orientation

3.1 Basic structures and characteristics of genVMOO

The following features characterize genVM-OO:

- It complies with the Unified Process (UP) [Jacobson 1999] (although it was developed at the same time as UP in part),
- It is based on the widely used "nearly" standard notation UML,
- It can be used in a controlled way for iterative, incremental development,
- It is application case driven,
- It can be modified for each specific project.

genVMOO uses the static structure and the numbering system of VM-BASIS, consisting of phase, segment and activity. The main focus here is to describe what is to be done and how it is to be done.

The following main goals are associated with the individual phases:

- *Conception:* Laying down the basic requirements for the system. The ideas that the developers, analysts and end users have are described in overall terms and the general conditions are outlined.
- *Analysis*: The description of all the functional and operational characteristics of the future system.
- *Design*: Laying down the system architecture and mapping the analysis model for the solution area.
- *Implementation*: Transforming the design model into programs that are coded in an (object-oriented) programming language.
- *Test*: Proof that the implemented system behaves as required and complies with all the specifications.

The following standard template is used to represent the segments within a phase (or the activities within a segment) (here using the example of the phase "Object-oriented analysis"):

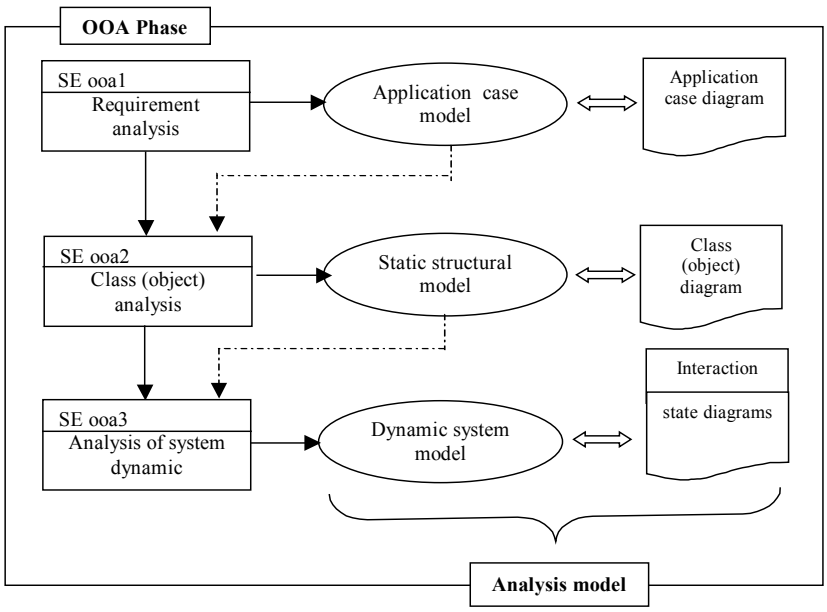

Figure 3: **Segments within the object-oriented analysis (OOA) phase**

The dynamic structure (project management view) - milestone plans are used to describe **when** something must be done and **who** should do it - is taken from the Unified Process and consists of the following stages (see also [Müller-Ettrich 1999]):

- *Inception*: for the fundamental orientation and rough planning for the project. At the end, sufficient information should be available to decide whether the project is to be continued or not.
- *Elaboration*: for the analysis of the area of application and the development of a viable architectural model.
- *Construction*: for incremental generation, testing all software components and integrating them into a product.
- *Transition*: for acceptance of the application, delivery and launch on the user's premises.

3.2 Software reuse in genVMOO

genVMOO not only defines the procedure for development of object-oriented software systems, but also supports project staff in the development of *component-based* software architectures. For this, it is important to clarify, on the one hand, the link between the reuse processes and the genVMOO-supported software development processes and, on the other hand, the assignment of possible re-usable candidates to the phases of the genVMOO [SW-WiVe 1999]. These interrelations are shown in Figure 4.

The fundamental thesis here is that reuse is justified in all phases of the software development. Not just codes should be reused, but all the products (assets) of the software development, such as project plans, specifications, templates, analysis and design models, test cases and test plans, documentations, etc.

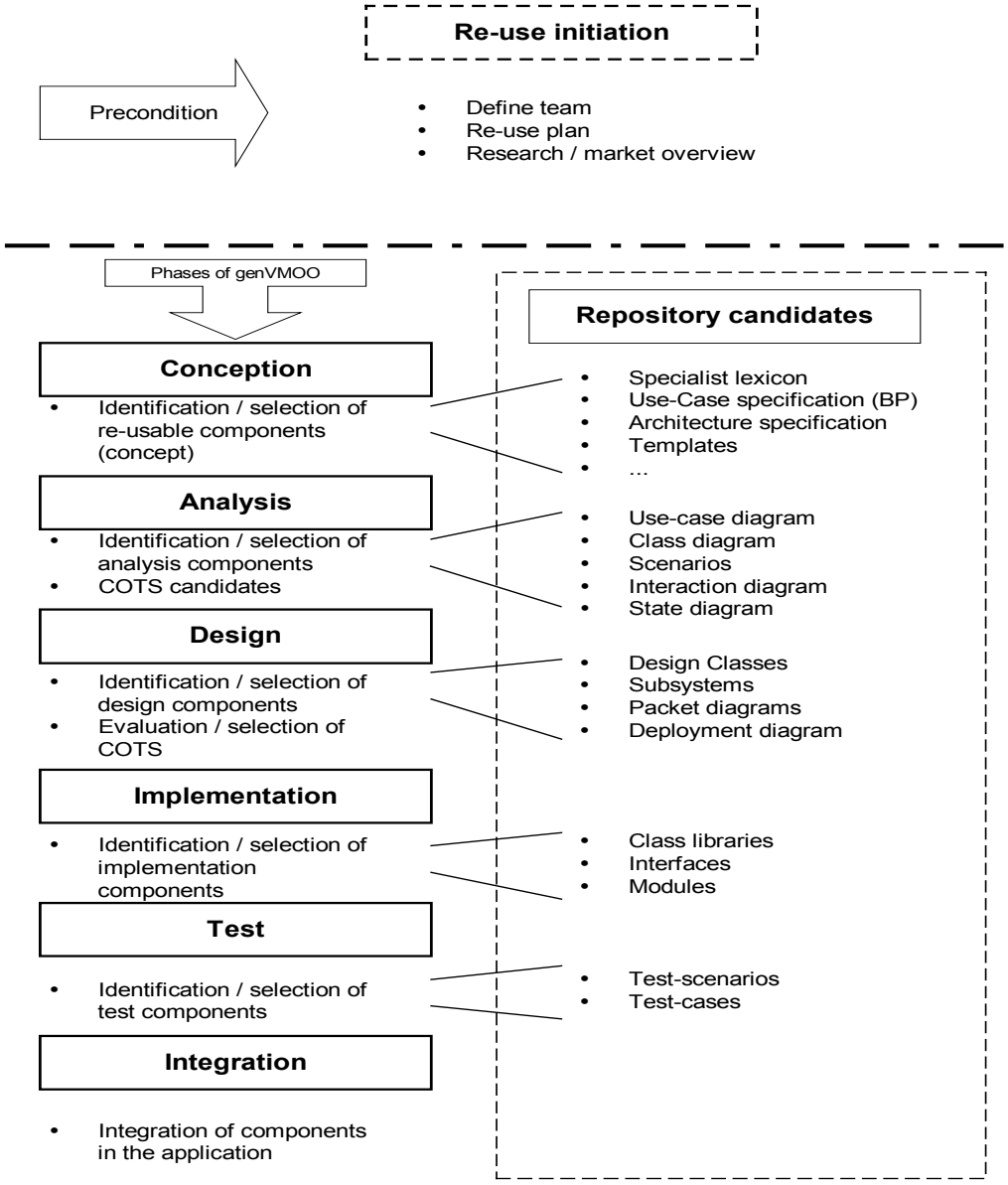

Figure 4: **Integration of the reuse process and the OO software development process (genVMOO)**

3.3 Metrics and measurement points in genVMOO

Since metrics unrelated to paradigms (input requirement metrics, error statistics, customer satisfaction, etc.) have been determined for many years, the application of metrics specially for object-oriented paradigms was started in parallel to the introduction of the object-oriented procedural model. On the basis of these metrics, both the quantitative and the qualitative evaluation of the models prepared (UML in Rational Rose), the program code (C++ and Java) and, indirectly, the procedural model itself are possible. In addition, it is also possible to carry out a partly automated input requirement estimate using Object Point [Sneed 1996].

However, it is also proposed to expand the application of metrics in relation to the conception and maintenance phases. In the case of conception, recording the metrics mainly affects the text-based descriptions or business process models on the basis, for example, expanded process chains. Maintenance would use such metrics as modification or port input requirements.

Determination of metrics has been defined for the first time in the object-oriented procedural model. To do this, defined measurement points were laid down, which allow projects to be compared, even within the ongoing software generation process, taking the marginal conditions (scope, complexity, quality...) into account.

The following two possibilities are the most rational when defining the measurement point:

1. measurement points to be determined by reference to segments, phases and ongoing cycles,
2. measurement points to be determined by milestones or milestone events.

The input required in the case of the former is naturally greater, but more precise conclusions can be drawn about the process, the resources and the actual software product. However, in addition, it also involves difficulties resulting from the fact that the conclusion of a segment or a phase cannot always be clearly identified, or the transitions between phases and cycles become blurred. If this is the case, the measurement points should be determined by reference to milestone events.

In the case of the former, in genVMOO, 11 significant measurement points were defined for the moment, and the initial experiments were carried out. For OO projects with few cycles (2 to 3 cycles), this approach is quite good. On the other hand, the second approach seems to be more suitable for more complex projects with more than 3 cycles.

4 genVMOO tool support

4.1 Process Management Tool

The efficient use of procedural models depends, in our opinion, largely on tool support. In a first approach, a process management tool should support teamwork, the project manager and changes to the genVMOO itself. For the teamwork, the procedural model should be displayed on screen as an ongoing element.

Each person involved in the project should thus be taken individually through the activities that the project manager has assigned to him or her. At every point, everyone involved knows what concrete products /results documents are expected of them. The aim is to release software developers from any overheads that a procedural model inevitably brings with it, and thus to support the acceptance and spread of genVMOO. A further aspect is the support of the project manager, who should be given a type of checklist of all the tasks that are to be dealt with in principle. In addition, it should be possible to plan the project in terms of time, resources and functions (phase structure, milestones, etc.). Modifications to the genVMOO should also be carried out with the support of tools so that the system can react flexibly to new requirements. This means that the tool will support the continuous improvement of the entire software development process as it moves towards a higher level of organizational maturity. These requirements thus produce the functional tool characteristics shown below.

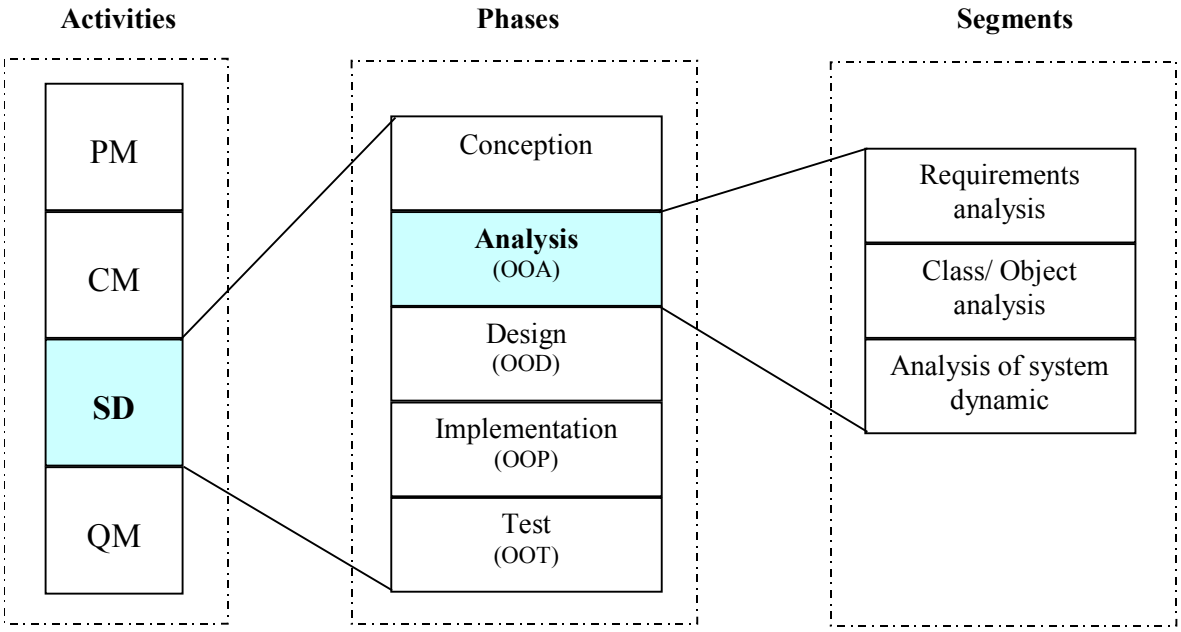

Figure 5: **Presentation using the example of system development**

1. Visualizing the components of genVMOO in the form of a tree structure (cf. Figure 5) according to the following areas of activity:
   - Project management,
   - Configuration management,
   - System development,
   - Quality management.

   For system development, the following should also be visualized:
   - the development phases, divided into segments and activities,
   - the results documents (description forms) and
   - instructions/ explanations needed to carry out the activities.

   For the results documents, pre-produced templates will be provided and visualized.

2. Visualizing the workflow in the team, computer administration of all results documents, (automatic) incorporation of the tools used in the project for carrying out activities and generation of the results documents.

3. Control of the software development process on the basis of genVMOO, i.e. those involved in the project should be taken reliably through the process of development.

4. Automatic modification (*tailoring*) of genVMOO to a concrete project. As a result of this, a project-specific VMOO is created, since not all the activities /results documents proposed in the genVMOO are required for every specific project.
   Marginal conditions and project characteristics are offered to the user in a dialogue. The actual *tailoring* is therefore reduced to a series of mouse clicks.

5. Setting up the specific project – planning of times and functions (phases, increments, or cycles, milestones, etc.)

6. Maintenance and carrying out changes to the genVMOO, resulting from project experience. The modifications are carried out interactively on the genVMOO.
   The following modifications can be undertaken by the project manager:
   - definition of segments or activity and product types (description forms) and indication of relationships (product flow) between segments, activities and results documents
   - laying down roles and processing states
   - formulation of tailoring rules

In an initial development phase, the project management tool could be realized on the basis of 3 components: a visualizer (partly realized already) to carry out tasks 1 to 3, a project manager to carry out tasks 4 and 5, and a model manager to carry out task 6. The extent to which the company's own tools could be implemented or standard applications such as MS Project could be used for project planning tasks has not yet been conclusively discussed. Tasks 4 and 5 are already partly supported within the framework of a metrics database that has now been implemented; this will be explained in the following.

4.2 Using a metrics database

The incorporation of metrics within the software lifecycle requires, as already stated, defined measurement points and, for efficient processing as part of statistical evaluations, these must be saved in a database. To do this, the MetricDB application was implemented on the basis of a multilevel C/S architecture, the client interface of which is shown in Figure 6.

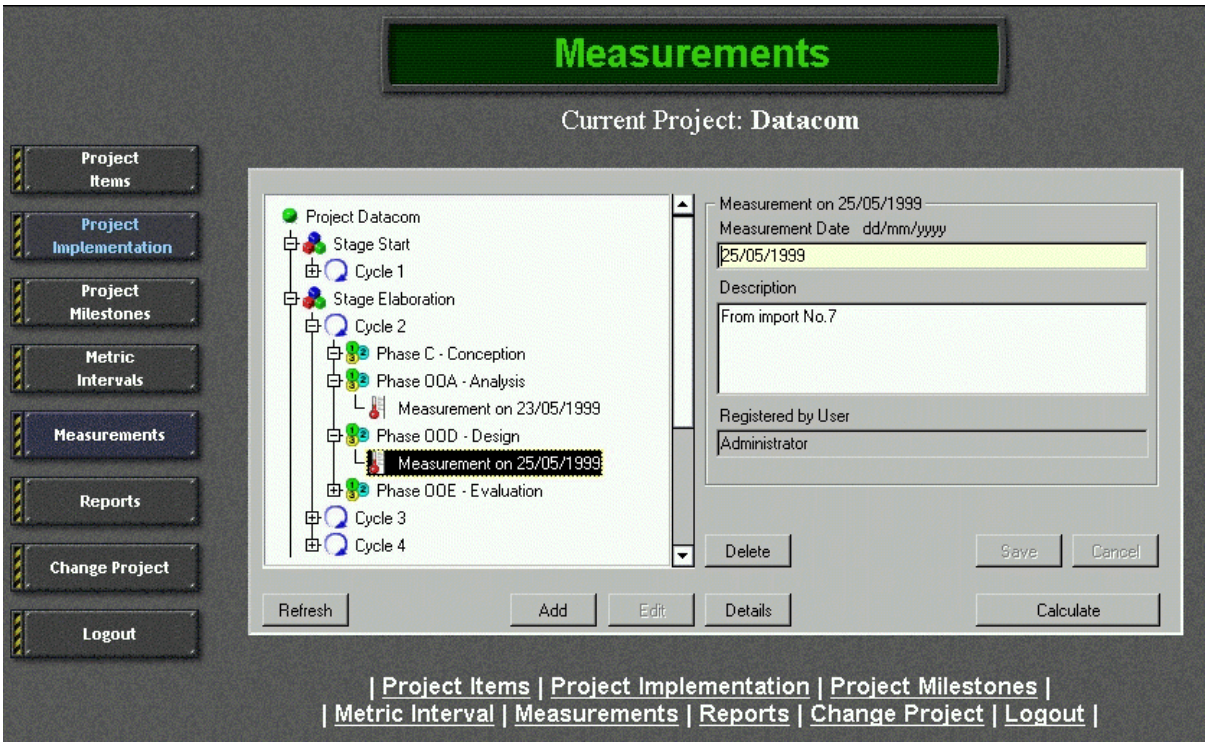

Figure 6: **Browser dialog of the metrics database**

The significant advantages of this application are its possible incorporation of every type of procedural model (on the basis of templates), the automatic takeover of measured values from corresponding measurement tools, the use of many types of metrics, including configurable threshold values and a corresponding textual and graphic report, for example, for comparing the metrics of two projects over a period of time to be defined. Figure 6 clearly shows the

procedural model that was briefly described, with its stages, cycles and phases, with the measurements carried out always being assigned to the relevant phases. A more detailed exploration of this subject is given in [Schmietendorf 1999].

5 Summary and outlook

In the past, complex and, above all, extensive procedural models have caused major problems in practical use. It cannot be expected that all developers should be familiar with "Paperware" running into a few thousand pages. Accordingly, a procedural model must be a guideline for the creation of software products and must follow a learning approach integrated into the development. At the moment, project managers and developers receive just 50 pages of text to help them find their way around "genVMOO". Another important factor for the successful, efficient use of procedural models is their granularity, i.e. special marginal conditions should only be supported by the procedural model if they have to be taken into account within development; otherwise, this type of complexity should be kept well away from the user.

In addition to these fundamental questions, the procedural model that has been selected or developed requires tool support. The question is whether a standard tool that takes everything into consideration is preferable or whether existing tool approaches should be integrated independently, including the appropriate CASE tools and programming environments. Even if integration of this type seems more time-consuming at first glance, a considerable level of dependency is likely with a procedural model and associated tool chain developed solely by one company. One possible solution would be the definition of a standard interface (such as CORBA-IDL or XMI), via which all the tools used as part of development could exchange information. In addition, another advantage of individual integration is that practical experience can be taken into account in the corresponding procedural models, which should give a corresponding market advantage for the software development house in question in the medium term.

To complete the genVMOO we are planning to integrate the necessary performance engineering processes. Taking into account the performance of an information system in terms of response times, throughput and process flow times requires an approach that takes the complete software development cycle into consideration. A number of viable integration models are already available for this (e.g. [Smith 1990]); the processes necessary for these must be integrated into current procedural models. In addition to the realization of the tool support already described, this theme is at the heart of further tasks that have to be dealt with.

References

[Biskup 1996] *Biskup, H.; Fischer, T.:* Vorgehensmodelle - Versuch einer begrifflichen Einordnung (Procedural models - an attempt at a classification of terms). In:

Leitungsgremium des GI-FA 5.1 (Hrsg.): Memorandum 2/96 of Technical Committee 5.1 "Management of Application Development and Maintenance". Karlsruhe 1996.

[Coplien 1995] *Coplien, J.:* A development Process generative Pattern Language. Bell Laboratories, 1995 http://portal.research.bel-labs.com/orgs/ssr/people/ cope/Patterns/ Process)

[Dimitrov 1998] *Dimitrov, E.:* Ein Vorgehensmodell für die Objektorientierte Entwicklung (A procedural model for object-oriented development). Internal Report, Deutsche Telekom AG, Entwicklungszentrum Berlin, 1998

[Dumke 1993] *Dumke, R.:* Modernes Software Engineering. Vieweg Lehrbuch Informatik, Braunschweig/Wiesbaden 1993

[Jacobson 1999] *Jacobson, I.; Booch, G; Rumbaugh, J.:* The Unified Software development Process. Addison-Wesley, Reading (Mass.) 1999

[Müller-Ettrich 1999] *Müller-Ettrich, G.:* Objektorientierte Prozeßmodelle. UML einsetzen mit OOTC, V-Modell, Objectory (Object-oriented process models. Using UML with OOTC, V-model, objectory). Addison-Wesley, 1999

[Noack 1997] *Noack, J.; Schienmann, B.; Kittlaus, H.-B.:* Ein Leitfaden für die objektorientierte Anwendungsentwicklung in der Sparkassenorganisation (A guideline for object-oriented application development in the savings bank organization). OBJECTspektrum 6/97, S.52-59, 1997

[Rational 1999] *Rational:* UML Unified Modeling Language, Version 1.1 and 1.3, http://www.rational.com

[Schmietendorf 1999] *Schmietendorf, A.; Stoyanov, S.; Mourdjeva, A.:* Implementation of a Metrics Database for Industrial Use. Metrics News Vol. 4, No. 1, July 1999

[Smith 1990] *Connie U. Smith, C. U.:* Performance Engineering of Software Systems, Addison-Wesley, New York 1990

[Sneed 1996] *Sneed, H. M.:* Schätzung der Entwicklungskosten von objektorientierter Software (Estimating the development costs of object-oriented software). Informations Spektrum 19, Springer Verlag, Heidelberg 1996, S. 133-140

[SW-WiVe 1999] Projekt Software-Wiederverwendung (SW-WiVe) (The software reuse project), Final report, T-Nova, Deutsche Telekom Innovationsgeselschaft, Entwicklungszentrum Berlin, 1999

[VMBasis 1996] *Pullwitt, S.; Tannenbaum, K. G.:* Vorgehensmodell der Deutschen Telekom: Entwicklung und Instandhaltung von komplexen Softwaresystemen (Deutsche Telekom procedural model: the development and maintenance of complex software systems). *A. Ganser* (publ.), Munich, Vienna, Oldenbourg 1996

Dumke, R.; Lehner, F.:

Software-Metriken - Entwicklungen, Werkzeuge und Anwendungsverfahren
DUV Publisher, Wiesbaden, 2000 (229 pages)

ISBN 3-8244-7120-5

The includes the papers of the 9th German Workshop on Software Measurement in Regensburg in September 1999. The contents is

Michael Jacobsen-Rey
AUTOMATED SOFTWARE INSPECTION - Attaining New Levels of Software Quality

Thomas Fetcke
TWO PROPERTIES OF FUNCTION POINT ANALYSIS
Erik Foltin, Reiner Dumke, Andreas Schmietendorf
ENTWURF EINER INDUSTRIELL NUTZBAREN METRIKEN-DATENBANK
Projekt: metricDB-2 V 0.8

Claus Lewerentz, Heinrich Rust, Frank Simon
QUALITY - METRICS - NUMBERS - CONSEQUENCES

Reiner Dumke
ERFAHRUNGEN IN DER ANWENDUNG EINES ALLGEMEINEN OBJEKT-ORIENTIERTEN MEASUREMENT FRAMEWORK

Andreas Schmietendorf, Evgeni Dimitrov, Reiner Dumke, Erik Foltin, Michael Wipprecht
KONZEPTION UND ERSTE ERFAHRUNGEN EINER METRIKENBASIERTEN SOFTWARE-WIEDERVERWENDUNG

Patricia Mandl-Striegnitz
UNTERSUCHUNG EINES NEUEN ANSATZES ZUR PROJEKTMANAGEMENT-AUSBILDUNG

Hans Windpassinger
MÖGLICHKEITEN DER METRIK-BASIERTEN MODELLIERUNG UND AUSWERTUNG VON QUALITÄTSVORGABEN MIT DEM WERKZEUG LOGISCOPE

Silvio Löffler, Frank Simon
SEMIAUTOMATISCHE, KOHÄSIONSBASIERTE SUBSYSTEMBILDUNG

Ulrich Schweikl, Stefan Weber, Erik Foltin, Reiner Dumke
APPLICABILITY OF FULL FUNCTION POINTS AT SIEMENS AT

Harry M. Sneed
TESTMETRIKEN FÜR OBJEKTORIENTIERTE BANKENANWENDUNGEN

Christof Ebert
PROCESS CHANGE MANAGEMENT IN LARGE ORGANIZATIONS

Angelika Mittelmann
MESSEN VON WEICHEN FAKTOREN - Ein Erfahrungsbericht

Wohlin,Claes et al.:

Experimentation in Software Engineering - An Introduction
Kluwer Academic Publishers Boston/Dordrecht/London, 2000 (204 pages)

ISBN 0-7923-8682-5

The purpose of EXPERIMENTATION IN SOFTWARE ENGINEERING: *An Introduction* is to introduce students, teachers, researchers, and practitioners to experimentation and experimental evaluation with a focus on software engineering. The objective is, in particular, to provide guidelines for performing experiments evaluating methods, techniques and tools in software engineering. The introduction is provided through a process perspective. The focus is on the steps that must be taken to perform experiments and quasi-experiments. The process also includes other types of empirical studies.

The motivation for the book emerged from the need for support the authors experienced when making their software engineering research more experimental. Several books are available that either treat the subject in very general terms or focus on some specific part of experimentation; most focus on the statistical methods in experimentation. These are important, but there are few books elaborating on experimentation from a process perspective; none addressing experimentation in software engineering in particular.

The scope of EXPERIMENTATION IN SOFTWARE ENGINEERING: *An Introduction* is primarily experiments in software engineering as a means for evaluating methods, techniques and tools. The book provides some information regarding empirical studies in general, including both case studies and surveys. The intention is to provide a brief understanding of these strategies and in particular to relate them to experimentation.

EXPERIMENTATION IN SOFTWARE ENGINEERING: *An Introduction* is suitable for use as a textbook or a secondary text for graduate courses, and for researchers and practitioners interested in an empirical approach to software engineering.

Bundschuh, M.; Fabry, A.:

Aufwandschätzung von IT-Projekten
MITP Publisher, Bonn, 2000 (331 pages)
ISBN 3-8266-0534-9

This new book about software effort and costs estimation, includes a description of the current used methods in practice. A detailed presentation considers the Function Point methods and their different approaches. The book includes some case studies and is directed for a general practical use in the IT area.

Menascé, D.A.; Almeida, V.A.F.:

Scaling for E-Business – Technologies, Model, Performance, and Capacity Planning

Prentice Hall Publ., 2000 (449 pages)
ISBN 0-13-086328-9

This book teaches you how to approach website performance problems in a methodical and quantitative way. It introduces a methodology to analyze the way websites are used (behavior model graphs) and how work flows through them (interaction diagrams). The book shows you how to build these models from web logs or from a system analysis. It then shows you how to use these models to analyze your current system's behavior, and also to predict how much capacity you will need as demand grows and changes.

The book gives a very readable treatment of each step in this process, giving background tutorials on networking, web servers, server-side scripts, and database servers. It also gives quantitative measures of each of these components, telling you how to size servers and networks for each step of the interaction diagram. For example, it shows the relative cost of ordinary HTTP transactions, and then progresses to SSL/TOS secure transactions, and then SET transactions. In each case it explains the technology, then it explains the performance implications, and finally it considers the pros and cons of using hardware accelerators for the cryptographic steps. Each concept is exemplified by a specific example worked out in detail.

The web is unpredictable: it is very hard to guess what will happen next. What new technology will appear next month? What new security hole will pop up? What feature will create explosive growth on your site? This book cannot answer those questions – no book can. But, once you know what you want to do, this book gives you the quantitative tools to estimate the capacity needed to provide the new features and to estimate what they will cost, and also to estimate the new system's performance and response time.

Professors Menascé and Almeida have developed a pragmatic approach to website performance modeling. This practitioner's handbook abstracts the current research articles and textbooks – giving you clear advice on how to approach performance problems. The result is a very readable and useful tutorial on how to scale up a website from a single server to a site handling millions of transactions per day.

Dumke, R.; Abran, A. (Eds.):

New Approaches in Software Measurement

10th International Workshop, IWSM 2000, Berlin, Germany, October 4-6, 2000

LNCS 2006, Springer-Verlag, Heidelberg 2001 (244 pages)
ISBN 3-540-41727-3

OBJECT-ORIENTED SOFTWARE MEASUREMENT

Impact of Inheritance on Metrics for Size, Coupling, and Cohesion in Object -Oriented Systems
D. Beyer, C. Lewerentz, F. Simon
Measuring Object-Orientedness: the Invocation Profile
P. Rosner, T. Hall, T. Mayer
CEOS - A Cost Estimation Method for Evolutionary, Object-Oriented Software Development
S. Sarferaz, W. Hesse
A Measurement Tool for Object Oriented Software and Measurement Experiments with it
L. Xinke, L. Zongtian, P. Biao, X. Dahong

INVESTIGATIONS IN SOFTWARE PROCESS IMPROVEMENT

Estimating the Cost of Carrying out Tasks Relating to Performance Engineering
E. Foltin, A. Schmietendorf
Measurement in Software Process Improvement Programmes: An Empirical Study
T. Hall, N. Baddoo, D. Wilson
Improving Validation Activities in a Global Software Development
C. Ebert, C. Hernandez Parro, R. Suttels, H. Kolarczyk
A Generic Model for Assessing Process Quality
M. Satpathy, R. Harrison, C. Snook, M. Butler
Maturity Evaluation of the Performance Engineering Process
A. Schmietendorf, A. Scholz

FUNCTION-POINT-BASED SOFTWARE MEASUREMENT

COSMIC FFP and the World-Wide Field Trials Strategy
A. Abran, S. Oligny, C.R. Symons
Extraction of Function-Points from Source-Code
H.M. Sneed
Early & Quick COSMIC-FFP Analysis using Analytic Hierarchy Process
L. Santillo

SOFTWARE MEASUREMENT OF SPECIAL ASPECTS

Measuring the Ripple Effect of Pascal Programs
S. Black, F. Clark
An Assessment of the Effects of Requirements Reuse Measurements on the ERP Requirements Engineering Process
M. Daneva

A New Metric-Based Approach for the Evaluation of Customer Satisfaction in the IT Area
R.R. Dumke, C. Wille
Utility Metrics for Economic Agents
D. Schmelz, M. Schmelz, J. Schmelz

IMPROVING THE SOFTWARE MEASUREMENT PROCESS

QF^2D: a Different Way to Measure Software Quality
L. Buglione, A. Abran
Using FAME Assessments to Define Measurement Goals

D. Hamann, A. Beitz, M. Müller, R. van Solingen
Mapping Processes Between Parallel, Hierarchical and Orthogonal System Representations
F. Dion, T.K. Tran, A. Abran

Dumke, R.; Rautenstrauch, C.; Schmietendorf, A.; Scholz, A. (Eds.):

Performance Engineering. State of the Art and Current Trends

LNCS 2047, Springer-Verlag, Heidelberg 2001

One of the most critical non-functional quality factors of a software system is the performance characteristic. The main idea of performance engineering is to consider the performance as a design target throughout the whole software development process and especially in its early phases.

The objective of this book is to bring researchers and industry experts together in describing the state of the art as well as current trends of performance engineering. Thereby a major part of all facets of this innovative development technique can be discussed. Each paper will provide insight into the effective use of performance engineering through methods, models, case studies, experience reports, or experiments.

The contributions of the book are based on:

- The first German Workshop Performance Engineering within the Software Development PE2000 May, 17 2000 in Darmstadt, Germany
- The second international Workshop on Software and Performance – WOSP 2000, September, 17.-20. 2000 in Ottawa, Canada
- and a separate Call for Book Chapters

The book contains the following articles:

INTRODUCTION

Historical Roots of Performance Engineering
Aspects of Performance Engineering – An Overview

RELATIONS BETWEEN SOFTWARE AND PERFORMANCE ENGINEERING

Conception of a Web-based SPE Development Infrastructure
Dumke, R., Koeppe, R.
Performance and Robustness Engineering and the Role of Automated Software Development
Gerlich, R.
Performance Engineering of Component-Based Distributed Software Systems
Gomaa, H., Menascé, D.A.
Conflicts and Trade-offs between Software Performance and Maintainability
Lundberg, L., Häggander, D., Diestelkamp, W.
Performance Engineering on the Basis of Performance Servic Levels
Rautenstrauch, C., Scholz, A.
Possibilities of Performance Modelling with UML
Schmietendorf, A., Dimitrov, E.
Origins of Software Performance Engineering: Highlights and Outstanding Problems

Smith, C.U.
Performance Parameters and Context of Use
Stary, C.

PERFORMANCE MODELING AND PERFORMANCE MEASUREMENT

Using Load Dependent Servers to Reduce the Complexity of Large Client-Server Simulation Models
Curiel, M., Puigjaner, R.
Performance Evaluation of Mobile Agents: Issues and Approaches
Dikaiakos, M.D., Samaras, G.
UML-based Performance Modeling Framework for Component-Based Distributed Systems
Kähkipuro, P.
Scenario-based Performance Evaluation of SDL/MSC-specified Systems
Kerber, L.
Characterization and Analysis of Software and Computer Systems with Uncertainties and Variabilities
Majumdar, S., Lüthi, J., Haring, G., Ramadoss, R.
The Simalytic Modeling Technique
Norton, T.R.
Resource Function Capture for Performance Aspects of Software Components and Sub-systems
Woodside, M., Vetland, V., Courtois, M., Bayarov, S.

PRACTICAL EXPERIENCE

Shared Memory Contention and its Impact on Multi-Processor Call Control Throughput
Drwiega, T.
Performance and Scalability Models for a Hypergrowth e-Commerce Web Site
Gunther, N.J.
Performance Testing for IP Services and Systems
Huebner, F., Meier-Hellstern, K., Reeser, P.
Performance Modelling of Interaction Protocols in Soft Real-Time Design Architectures
Juiz, C., Puigjaner, R., Jackson, K.
A Performance Engineering Case Study: Software Retrieval System
Merseguer, J., Campos, J., Mena, E.
Performance Management of SAP® Solutions
Schneider, T.

METRICS 2001 & ESCOM 2001:

7th International Symposium on Software Metrics
April 2 - 6, 2001, London, England
see: http://www.mmhq.co.uk/2001/

PE2001:

2. German Workshop of Performance Engineering in Software Development
April 19, 2001, Munich, Germany
see: http://www-wi.cs.uni-magdeburg.de/pe2001/

FESMA/DASMA 2001

4th European Conference on Software Measurement and ICT Control
May 9 - 11, 2001, Heidelberg, Germany
see: http://www.ti.kviv.be/conf/fesma.htm

QualWeek 2001

14th Annual International Internet & Software Quality Week 2001
29 May - 1 June 2001, San Francisco, California
see: http://www.soft.com/QualWeek/QW2001/

IFPUG 2000, Fall:

International Function Point User Group Fall Conference,
September 11-15 , 2000, San Diego, USA
see: http://www.ifpug.org/conferences/conf.html

IWSM'2001:

11th International Workshop on Software Measurement
August 28 - 29, 2001, Montreal, Canada
see: http://lrgl.uqam.ca/workshop2001/

PROFES 2001:

3rd International Conference on Product Focused Software Process Improvement
September 10 - 13, 2001, Kaiserlautern, Germany,
see: http://www.ele.vtt.fi/profes2001/

CONQUEST 2001:

Conference on Quality Engineering in Software Technology
September 19 - 21, 2001, Nuremberg, Germany
see: http://www.asqf.de/

UML 2001:

Fourth International Conference on the Unified Modeling Language
October 1 - 5, 2001, Toronto, Canada
see: http://www.cs.toronto.edu/uml2001/

WCRE 2001:

8th Working Conference on Reverse Engineering
October 2 - 5, 2001, Stuttgart, Germany
see: http://www.reengineer.org/wcre2001/

ICSM'2001:

IEEE International Conference on Software Maintenance
November 6 - 10, 2001, Florence, Italy

see: http://www.dsi.unifi.it/icsm2001/

EuroSTAR 2001:

9th European International Conference on Software Testing Analysis & Review,
November 19 - 23, 2001, Stockholm, Sweden
see: http://www.eurostar.ie/

see also: **OOIS, ECOOP** and **ESEC** European 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. Commercial site offering products and services for software developers (i. e. Y2K, Testing or Quality Assurance)

- **http://www.sei.cmu.edu/**
  Software Engineering Institute of the U. S. Department of Defence at Carnegie Mellon University. Main objective of the Institute is to identify and promote successful software development practices.
  Exhaustive list of publications available for download.

- **http://dxsting.cern.ch/sting/sting.html**
  Software Technology INterest Group at CERN: their WEB-service is currently limited (due to "various reconfigurations") to a list of links to other information sources.

- **http://www.spr.com/index.htm**
  Software Productivity Research, Capers Jones. A commercial site offering products and services mainly for software estimation and planning.

- **http://fdd.gsfc.nasa.gov/seltext.html**
  The Software Engineering Laboratory at NASA/Goddard Space Flight Center. Some documents on software product and process improvements and findings from studies are available for download.

- **http://www.qucis.queensu.ca/Software-Engineering/**
  This site hosts the World-Wide Web archives for the USENET usegroup comp.software-eng. Some links to other information sources are also provided.

- **http://www.esi.es/**
  The European Software Institute,Spain

- **http://saturne.info.uqam.ca/Labo_Recherche/lrgl.html**
  Software Engineering Management Research Laboratory at the University of Quebec, Montreal. Site offers research reports for download. One key focus area is the analysis and extension of the Function Point method.

- **http://www.SoftwareMetrics.com/**
  Homepage of Longstreet Consulting. Offers products and services and some general information on Function Point Analysis.

- **http://www.utexas.edu/coe/sqi/**
  Software Quality Institute at the University of Texas at Austin. Offers comprehensive general information sources on software quality issues.

- **http://wwwtrese.cs.utwente.nl/~vdberg/thesis.htm**
  Klaas van den Berg: Software Measurement and Functional Programming (PhD thesis)

- **http://divcom.otago.ac.nz:800/com/infosci/smrl/home.htm**
  The Software Metrics Research Laboratory at the University of Otago (New Zealand).

- **http://ivs.cs.uni-magdeburg.de/sw-eng/us/**
  Homepage of the Software Measurement Laboratory at the University of Magdeburg.

- **http://www.cs.tu-berlin.de/~zuse/**
  Homepage of Dr. Horst Zuse

- **http://dec.bournemouth.ac.uk/ESERG/bibliography.html**
  Annotaded Bibliography on Object-Oriented Metrics

- **http://www.iso.ch/9000e/forum.html**
  The ISO 9000 Forum aims to facilitate communication between newcomers to Quality Management and those who, having already made the journey have experience to draw on and advice to share.

- **http://www.qa-inc.com/**
  Quality America, Inc's Home Page offers tools and services for quality improvement. Some articles for download are available.

- **http://www.quality.org/qc/**
  Exhaustive set of online quality resources, not limited to software quality issues

- **http://freedom.larc.nasa.gov/spqr/spqr.html**

Software Productivity, Quality, and Reliability N-Team

- **http://www.qsm.com/**
  Homepage of the Quantitative Software Management (QSM) in the Netherlands

- **http://www.iese.fhg.de/**
  Homepage of the Fraunhofer Institute for Experimental Software Engineering (IESE) in Kaiserslautern, Germany

- **http://www.highq.be/quality/besma.htm**
  Homepage of the Belgian Software Metrics Association (BeSMA) in Keebergen, Belgium

- **http://www.cetus-links.org/oo_metrics.html**
  Homepage of Manfred Schneider on Objects and Components

- **http://dec.bournemouth.ac.uk/ESERG/bibliography.html**
  An annotated bibliography of object-oriented metrics of the Empirical Software Engineering Research Group (ESERG) of the Bournemouth University, UK

News Groups

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

Software Measurement Associations

- **http://www.aemes.fi.upm.es**
  AEMES Association Espanola de Metricas del Software

- **http://www.asqf.de**
  ASQF Arbeitskreis Software-Qualität Franken e.V., Nuremberg, Germany

- **http://www.cosmicon.com**
  COSMIC Common Software Measurement International Consortium

- DANMET: Danish Software Metrics Association

- **http://www.dasma.de**
  DASMA Deutsche Anwendergruppe für Software Metrik und Aufwands-schätzung e.V.

- **http://www.esi.es**
  ESI European Software Engineering Institute in Bilbao, Spain

- **http://www.fesma.org/**
  FESMA Federation of European Software Metrics Associations

- **http://www.sttf.fi**
  FiSMA Finnish Software Metrics Association

- FFPUG: French Function Point User Group

- FPUGA: Function Point User Group Austria

- **http://www.iese.fhg.de**
  IESE Fraunhofer Einrichtung für Experimentelles Software Engineering

- **http://www.isbsg.org.au**
  ISBSG International Software Benchmarking Standards Group, Australia

- **http://www.nesma.nl**
  NESMA Netherlands Software Metrics Association

- **http://www.sei.cmu.edu/**
  SEI Software Engineering Institute Pittsburgh

- **http://www.spr.com/**
  SPR Software Productivity Research by Capers Jones

- **http://fdd.gsfc.nasa.gov/seltext.html**
  SEL Software Engineering Laboratory - NASA-Homepage

- **http://www.vrz.net/stev**
  STEV Vereinigung für Software-Qualitätsmanagement Österreichs

- **http://www.sqs.de**
  SQS Gesellschaft für Software-Qualitätssicherung, Germany

- **http://www.ti.kviv.be**
  TI/KVIV Belgish Genootschap voor Software Metrics

- **http://www.uksma.co.uk**
  UKSMA United Kingdom Software Metrics Association

Software Metrics Tools (Overviews and Vendors)

Tool Listings

- **http://www.pitt.edu/~ddarcy/isprof/intotool.html#intro**
  Metrics Tool Listings by Dace Darcy

- **http://www.cs.umd.edu/users/cml/resources/cmetrics/**
  C/C++ Metrics Tools by Christopher Lott

- **http://davidfrico.com/mettools.htm**
  Software Metrics Tools by Dave

- **http://mdmetric.com/meastl1.htm**
  Maryland Metrics Tools

- **http://cutter.com/itgroup/reports/function.html**
  Function Point Tools by Carol Dekkers

Tool Vendors

- **http://www.mccabe.com**
  McCabe & Associates

- **http://www.scitools.com**
  Scientific Toolworks, Inc.

- **http://zing.ncsl.nist.gov/webmet/**
  Web Metrics

- **http://www.globalintegrity.com/csheets/metself.html**
  Global Integrity

- **http://www.spr.com/**
  Software Productivity Research (SPR)

- **http://jmetric.it.swin.edu.au/products/jmetric/**
  JMetric

- **http://www.imagix.com/products/metrics.html**
  Imagix Power Software

- **http://www.verilogusa.com/home.htm**
  VERILOG (LOGISCOPE

- **http://www.qsm.com/**
  QSM

METRICS NEWS

VOLUME 5 *2000* *NUMBER 2*

CONTENTS

Call for Papers **3**

Call for Participation **7**

Workshop Report **11**

Position Papers **23**

Hanebutte, N. and Dumke, R.R.:
Analyzing Software Design using a Measurable Program Design Language **23**

Dimitrov, E., Schmietendorf, A., and Dumke, R.:
Software Reuse and Metrics within a Process Model for Object-oriented Development **34**

New Books on Software Metrics **45**

Conferences Addressing Metrics Issues **51**

Metrics in the World-Wide Web **53**

ISSN 1431-8008