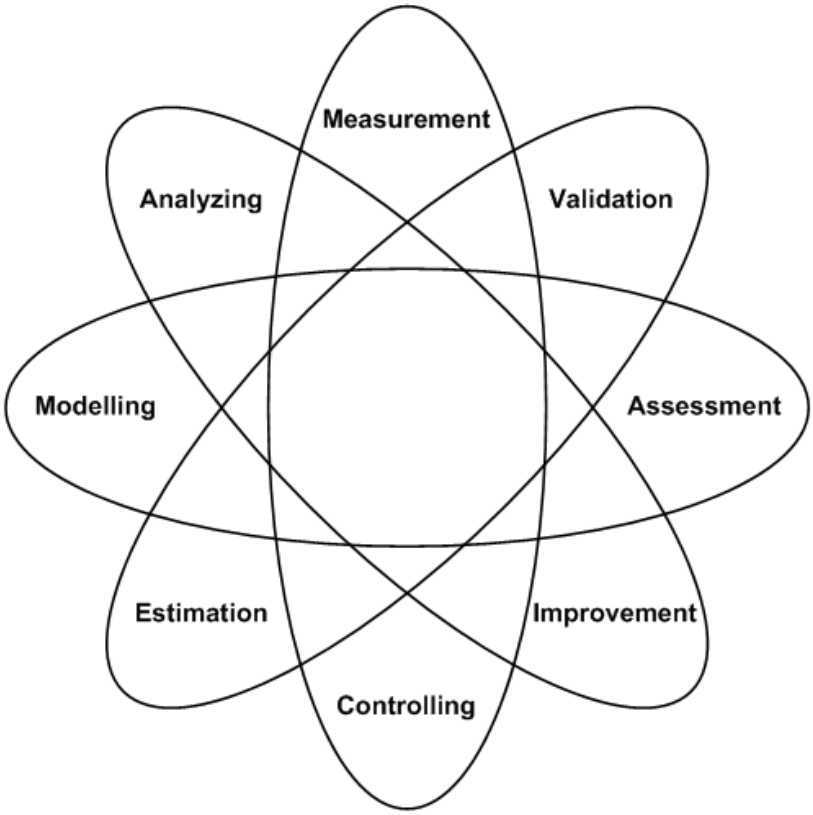
Volume 9, Number 2, December 2004

METRICS NEWS

Journal of the Software Metrics Community



Editors:

Alain Abran, Manfred Bundschuh, Reiner Dumke, Christof Ebert, Horst Zuse

Université du Québec
École de technologie supérieure

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

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.

METRICS NEWS

VOLUME 9 2004 NUMBER 2

CONTENTS

ISSN 1431-8008

==========

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

==========

15th International Workshop on Software Measurement
IWSM 2005

Co-sponsored by:

École de Technologie Supérieure - Université du Québec
&
Otto von Guericke University (Magdeburg, Germany)

In cooperation with:

SPIN-Montréal
and
COSMIC – Common Software Measurement International Consortium
German Interest Group on Software Metrics

Sept. 12-14, 2005

Montréal (Québec) CANADA
www.lrgl.uqam.ca/iwsm2005

==========

NEW this year:
An industry track + workshop on

GENERAL THEME & SCOPE: SOFTWARE 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.

Within the 2001-2003 timeframe, a significant number of key institutional documents have been brought into the public domain with a broad consensus based as ISO standards and technical reports. 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 topics (but not limited to):

A- Uses of measurements results in decision making:

- 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.).

B- Evaluation and assessment models:

- Performance assessment
- Quality assessment
- Maintenance assessment
- **Support systems assessment**

C- 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: UML, Web-based applications, Agent based systems, etc.
- Size attributes categories: Functional and non-functional, etc.

D- 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.

E- Measurement issues within international standards

- ISO software engineering new standards and work-in-progress on software measurement
- Working group 6: Software Products measurement: ISO 9126, 14598 and the new 25000 series
- Working group 12 : Functional Size Measurement – 14143 Parts 1 to 5
- Measurement program frameworks publicly available

Tutorials:

- ISO 19761: COSMIC-FFP – A Functional Size Measurement Method
- To be determined – call for proposals – deadline – May 1st

PROGRAM COMMITTEE

Alain Abran,	École de Technologie Supérieure - Université du Québec, Canada
Luigi Buglione,	AthosOrigin, Italy
Manfred Bundschuh,	DASMA, Germany
François Coallier,	ÉTS, Canada
Jean-Marc Desharnais,	ÉTS, Canada
Javier Dolado,	Universidad San Sebastian, Spain
Ton Dekkers,	Sogeti Nederland B.V., Netherlands
Reiner Dumke,	University of Magdeburg, Germany
Christof Ebert,	Alcatel, Paris, France
Nadine Hanebutte,	University of Idaho, USA
Franz Lehner,	University of Passau, Germany
Roberto Meli,	DPO, Italy
Andreas Schmietendorf,	FH Harz ,Wernigerode, Germany
Harry Sneed,	SES Munich/Budapest, Hungary
Charles Symons,	Software Measurement Service Ltd, Edenbridge, UK
Hannu Toivonen,	Nokia, Finland
Horst Zuse,	TU Berlin, Germany

SUBMISSIONS – <u>Industry Track</u> (presentations only)

- Abstract (max. 1 page): **June 15, 2005**,
- Notification of acceptance on: **June 30, 2005**
- Final Powerpoint presentation: **August 15, 2005**

SUBMISSIONS – <u>Research Track</u> – Full papers

- Authors should send proposed papers by e-mail by **April 10, 2005**
- Notification of acceptance on: **May 15, 2005**
- Final paper presentation: **July 15, 2005**

All proposals should be sent to:

Alain Abran or to
aabran@ele.etsmtl.ca
École de Technologie Supérieure
Dept. Génie électrique
1100 Notre Dame Ouest
Montreal (Quebec),
Canada H3C 1K3
Phone: +1-514-396-8632
Fax: +1-514-396-8684

Reiner Dumke
dumke@ivs.cs.uni-magdeburg.de
Otto-von-Guericke-Universitaet Magdeburg
Fakultaet fuer Informatik
Postfach 4120
39016 Magdeburg
Germany
Phone: +49-391-6718664
Fax: +49-391-6712810

FEES for authors: none

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

http://lrgl.uqam.ca/iwsm2005

==========

Deutschsprachige Anwendergruppe für Software-Metrik und Aufwandschätzung e.V.

MetriKon 2005

DASMA Software Metrik Kongress

15. / 16. November 2005

Fraunhofer IESE Kaiserslautern

GI-Fachgruppe 2.1.10
"Software Messung und Bewertung"

CALL FOR PAPERS

Veranstalter

DASMA e.V. und GI-Fachgruppe 2.1.10

Kontaktadressen für

MetriKon-Beiträge:

metrikon-beitraege@dasma.org

Ausstellungsreservierungen:

DASMA e.V.
c/o Romy Gampe
Lindenstraße 18
90542 Eckental
Fon: 09126 / 29 79 576
Fax: 09126 / 28 24 43
E-Mail: info@dasma.org

Programmkomitee

Manfred Bundschuh
AXA Service AG, Köln

Günter Büren
Büren & Partner, Nürnberg

Dr. Axel Dold
DaimlerChrysler AG, Ulm

Prof. Dr. Reiner Dumke
Universität Magdeburg

Dr. Christof Ebert
Alcatel, Paris

Bernd Gebhard
Bayerische Motoren Werke, München

Prof. Dr. Hans-Georg Hopf
GSO-Fachhochschule Nürnberg

Dr.-Ing. Marek Leszak
Lucent Technologies Bell Labs, Nürnberg

Prof. Dr. Klaus Lewerentz
Technische Universität Cottbus

Prof. Dr. Peter Liggesmeyer
Fraunhofer IESE, Kaiserslautern

Dr. Dirk Meyerhoff
Schüco-Service GmbH, Bielefeld

Dr. Jürgen Münch
Fraunhofer IESE, Kaiserslautern

Dr. Andreas Schmietendorf
T-Systems International, Berlin

Dr.-Ing. habil. Horst Zuse
Technische Universität Berlin

Wichtige Termine

30. Mai 2005
Abgabeschluss der Kurzfassung

25. Juli 2005 (spätestens)
Benachrichtigung über die Annahme

04. Oktober 2005
Abgabe des druckfertigen Beitrages

Ziele und Zielgruppe

Software-Metriken sind eine der Schlüsseltechniken für das Management und die Führung von Software-Entwicklungsprozessen. Die praktische Anwendbarkeit von Metriken und die Effektivität von Mess-Programmen sind dabei immer wieder Gegenstand von engagierten Diskussionen und wissenschaftlichen Betrachtungen. Die DASMA verfolgt seit mehr als 10 Jahren das Ziel, diese Themen aufzugreifen und damit den Erfahrungsaustausch über die Anwendung von Software-Metriken und die Anforderungen der Praxis zu fördern.

Mit der jährlich ausgerichteten MetriKon fördern und gestalten DASMA und GI-Fachgruppe 2.1.10 „Software Messung und Bewertung" gemeinsam diesen Erfahrungsaustausch zwischen den Beteiligten aus Industrie und Forschung.

Wir laden Sie herzlich ein, sich an diesem Austausch aktiv zu beteiligen.

MetriKon-Beiträge

Gewünscht werden **technische Beiträge** zur fachlichen Fundierung, Anwendung und Validierung von praxisrelevanten Softwaremetriken. Ebenfalls willkommen sind **Erfahrungsberichte** zur Einführung von Messprogrammen und zur Umsetzung von Messmodellen in die industrielle Praxis.

Ebenso können am Vortag der Tagung (14.November 2005) **Tutorien** zur Einführung oder Vertiefung komplexer Themen oder Verfahren der Software-Metrik angeboten werden.

Darüber hinaus wird eine **Ausstellung** organisiert, in der Dienstleister und Werkzeughersteller den Teilnehmern ihre Angebote zur Unterstützung von Softwaremetriken und Aufwandschätzung vorstellen. Aussteller sollten möglichst frühzeitig ihren Teilnahmewunsch beim Veranstalter äußern.

Themenkreis

Generell sind Beiträge zu allen Themen rund um Software-Metriken und Aufwandschätzverfahren erwünscht. Das Programmkomitee behält sich bei der Auswahl der eingereichten Beiträge vor, auf eine ausgewogene Mischung von Praxis und Theorie zu achten.

Zur Orientierung hier einige Stichworte zu erwünschten Vortragsthemen:

- Software-Metriken, Vergleich von Metriken, Einführung von Metriken etc.
- Metriken als Erfolgsfaktor für die Verfolgung von Projektzielen
- Einsatz, Einführung und Erfahrungen mit Aufwandschätzverfahren und -Tools
- Quantitatives Projektmanagement, Projektcontrolling
- CMMI- / SPICE-konforme Messprogramme und Metriken
- verwandte Themen wie Benchmarking von Prozessen und Projekten
- Erfahrungen beim Einsatz von Metriken/Aufwandschätzverfahren im Zusammenhang mit embedded Systems, Web-Anwendungen, DWH, OO, etc.

Weitere Informationen

Zur DASMA e.V. und zur MetriKon 2005 -Tagung finden Sie weitere Informationen im Web unter **http://dasma.org** und **http://www.metrikon.de**, zur GI-Fachgruppe 2.1.10 unter **http://ivs.cs.uni-magdeburg.de/sw-eng/us/giak**.

Beachten Sie bei der Erstellung eines Vortrages für die MetriKon 2005 bitte die „Richtlinien zur Einreichung von Beiträgen zur MetriKon 2005". Sie finden diese Richtlinien und auch den Call for Papers im Internet auf den Seiten der MetriKon unter der oben angegebenen Adresse.

Aufruf zur Einreichung von Beiträgen zum 6. Workshop des Arbeitskreises PEAK der GI FG 2.1.10 am 10. Juni 2005 bei T-Systems Berlin

---

Der Performance Engineering Arbeitskreis (kurz PEAK) der GI-Fachgruppe 2.1.10 (Software-Messung und -Bewertung) beschäftigt sich mit dem Performance Engineering in der System- und Softwareentwicklung. Im Folgenden finden sich einige Themen zu potentiellen Beiträgen des kommenden Workshops.

Performance Management und Performance Messungen:

- Performance Management von Softwareanwendungen
- Techniken der Performancemessung für Services
- Performanceadaptive Lösungen (Komponenten/Services/Agenten)

Techniken der Modellierung und verfügbare Werkzeuge:

- Modell driven Architecture (OMG MDA) und SPE
- Modellierung von Nutzerverhalten und QoS-Anforderungen
- XML-basierte Performance Spezifikationen

Industrielle Software Performance Engineering Prozesse

- Performance-orientierte Phasen der Softwareentwicklung
- Mehrwertpotentiale und Aufwände für das Performance Engineering
- Management von Performanceanforderungen

Unterschiede und Gemeinsamkeiten der System- und Softwareentwicklung

- Einsatz von Hardware vs. Entwicklung/Einsatz von Software
- Performanceeigenschaften beim System- und Architekturentwurf
- Standardisierung von Hard- und Softwarekomponenten

Wie in jedem Jahr wird sich der Workshop wieder als Diskussionsforum zu aktuellen Herausforderungen im Umfeld des Performance Engineering verstehen und auf aktuelle Trends eingehen. Dabei werden insbesondere die Inhalte internationaler Workshops (CMG-Conference, WOSP, UKPEW, …) im Umfeld des Performance Engineering beachtet.

WORKSHOP-BEITRÄGE

Praktiker und Wissenschaftler, die auf dem Gebiet des Performance Engineering und artverwandter Aufgabenbereiche aktiv sind, werden gebeten, Beiträge in einem der Formate *doc*, *rtf*, *pdf*, *ps* einzureichen. Der Umfang der Beiträge sollte 3000 Wörter nicht übersteigen. Die Formatierungsrichtlinien der Webseite des Arbeitskreises zu entnehmen. Die Beiträge sollen in Kurzvorträgen (Vortragslänge ca. 30 Minuten zzgl. 10 Minuten Diskussion) präsentiert werden. Angenommene Beiträge werden in einem Tagungsband veröffentlicht.

Bitte senden Sie ihre Beiträge per E-mail an:

andreas.schmietendorf@t-systems.com

TEILNAHME / ANMELDUNG

Interessenten werden gebeten, sich auf der Webseite des Workshops elektronisch anzumelden. Bei Anmeldung bis zum 1. Juni 2005, wird eine Teilnahmegebühr von 50.- Euro (GI-Mitglieder: 40.- Euro) erhoben, danach 60.- Euro bzw. 50.- Euro.

PROGRAMMKOMITEE	**TERMINE**
Prof. Dr. R. Dumke, *Otto-von-Guericke Universität Magdeburg* Dr. R. Gerlich, *BSSE System and Software Engineering* Prof. Dr. G. Henselmann, *Fachhochschule Schmalkalden* H. Herting, *T-Systems Darmstadt* Prof. Dr. R. Hopfer, *Hochschule für Technik und Wirtschaft Dresden* Prof. Dr. K. Kabitzsch, *Technische Universität Dresden* Prof. Dr. F. Lehmann, *Universität der Bundeswehr München* Prof. Dr. C. Rautenstrauch, *Otto-von-Guericke-Universität Magdeburg* Dr. S. Rugel, *Siemens München* Prof. Dr. A. Schmietendorf, *Hochschule Harz* Dr. E. Dimitrov *T-Systems Berlin* Dr. A. Scholz, *Booz Allen Hamilton* Dr. D. Stoll, *Lucent Technologies Nürnberg* Prof. Dr. F. Victor, *Fachhochschule Köln*	15.04.2005: Einreichung von Beiträgen 05.05.2005: Annahme/Ablehnung 11.05.2005: finales Workshop-Programm 20.05.2005: druckreife Beiträge 10.06.2005: Workshop in Berlin

WEBSEITE DES ARBEITSKREISES:

http://ivs.cs.uni-magdeburg.de/~gi-peak

Bericht der Fachgruppensitzung der GI-FG 2.1.10 Software-Messung und -Bewertung

Unsere Fachgruppensitzung fand im Rahmen der IWSM/Metrikon 2004 am zweiten Konferenztag, dem 4. November 2004, in Berlin statt.

Einschätzung bisheriger Ergebnisse:

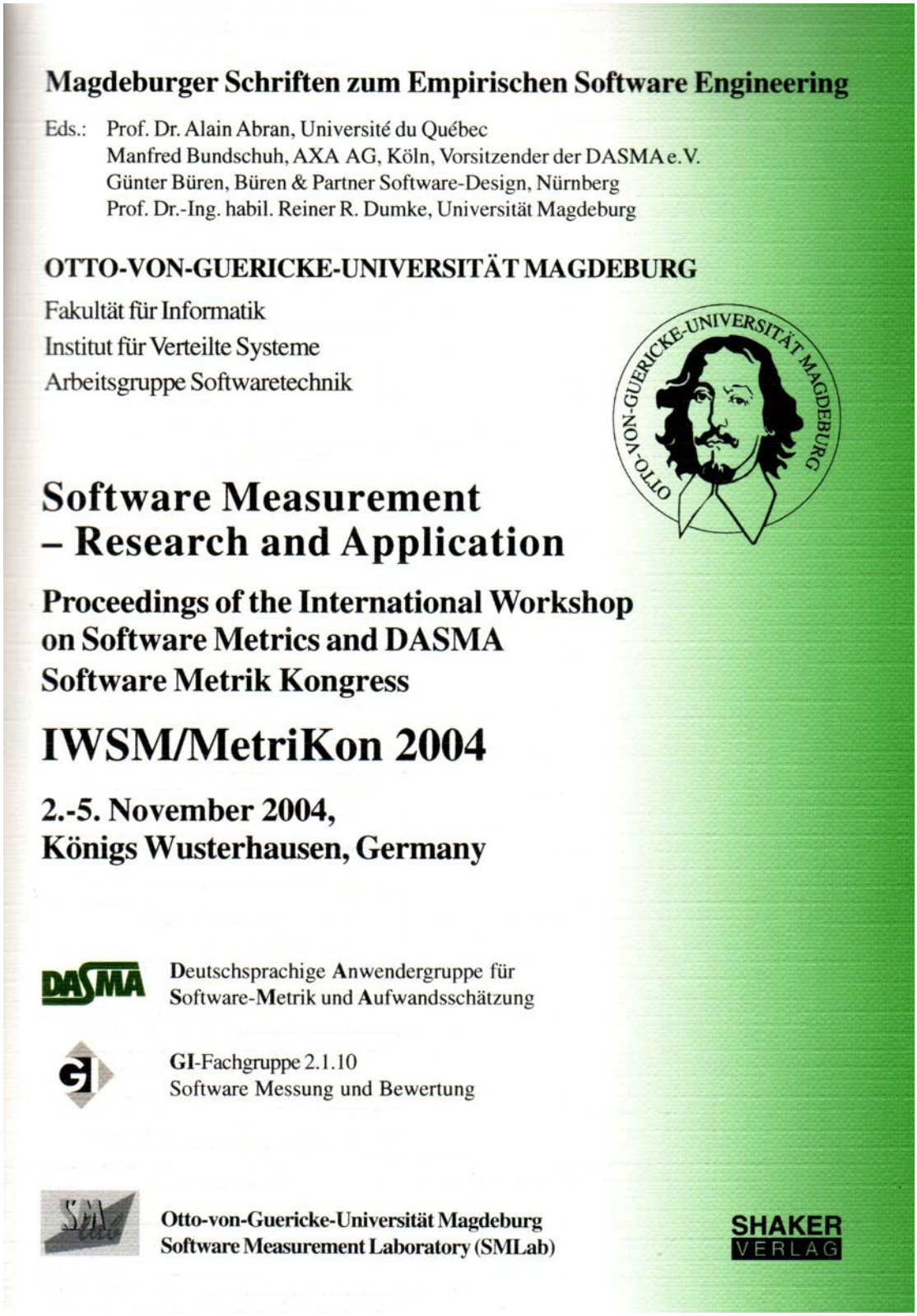
- Die Arbeit der Arbeitskreise wird insbesondere beim *Performance Engineering* als intensiv und sehr erfolgreich eingeschätzt. Für das Curriculum zur Softwaremessung sollten konkrete Lehransätze für themenbezogene Lösungen initiiert werden, um damit einen breiteren Interessentenkreis zu erschließen.
- Die Orientierung auf gemeinsame Metriken-Konferenzen mit den jeweiligen Communities, insbesondere der DASMA, erwies sich als sehr sinnvoll und erfolgreich.
- Insbesondere die *IWSM/Metrikon 2004* verhalf mit ihren Teilnehmern aus Australien, Belgien, Brasilien, Finnland, Frankreich, Griechenland, Großbritannien, Irland, Italien, Japan, Kanada, Niederlanden, Österreich, Schweiz, Spanien, USA und Deutschland zu interessanten Kontakten, Gesprächen und Initiativen.

Mögliche Initiativen und Vorhaben:

- Prof. Lewerentz schlug die Bildung eines Arbeitskreises *Software-visualisierung* vor, da der theoretische und vor allem empirische Hintergrund der Softwaremessung eine sinnvolle Ausrichtung für den Erkenntnisgewinn aus einer Visualisierung darstellen kann.
- Die FG 2.1.10 sieht ihr Wirken innerhalb der nationalen und internationalen Metriken-Communities und orientiert weiterhin auf die *gemeinsame Durchführung von Veranstaltungen* und Initiativen.
- Für die Mitglieder und Interessenten unserer Fachgruppe wird auf die Möglichkeit verwiesen, das *Virtuelle Software-Kompetenzzentrum* (ViSEK) zu nutzen bzw. durch eigene Beiträge zu erweitern und zu bereichern.

Reiner Dumke
(Sprecher der FG 2.1.10)

- Our 14th Workshop on Software Measurement (IWSM 2004) and Metrik Kongress (MetriKon 2004) took place in Berlin, Germany in November 2004. The following report gives an overview about the presented papers. Furthermore, the papers are published in the following Shaker book (ISBN 3-8322-3383-0):



Magdeburger Schriften zum Empirischen Software Engineering

Eds.: Prof. Dr. Alain Abran, Université du Québec
Manfred Bundschuh, AXA AG, Köln, Vorsitzender der DASMA e.V.
Günter Büren, Büren & Partner Software-Design, Nürnberg
Prof. Dr.-Ing. habil. Reiner R. Dumke, Universität Magdeburg

OTTO-VON-GUERICKE-UNIVERSITÄT MAGDEBURG

Fakultät für Informatik
Institut für Verteilte Systeme
Arbeitsgruppe Softwaretechnik

Software Measurement
– Research and Application

Proceedings of the International Workshop
on Software Metrics and DASMA
Software Metrik Kongress

IWSM/MetriKon 2004

2.-5. November 2004,
Königs Wusterhausen, Germany

DASMA Deutschsprachige Anwendergruppe für Software-Metrik und Aufwandsschätzung

GI-Fachgruppe 2.1.10
Software Messung und Bewertung

Otto-von-Guericke-Universität Magdeburg
Software Measurement Laboratory (SMLab)

SHAKER VERLAG

- **Move from function point counting to better project management and control!**

Pekka Forselius

Software Technology Transfer Finland Oy

pekka.forselius@sttf.fi

Abstract. It was 25 years ago, when the Function Point Analysis method was introduced to the software development community. It was the first technology independent way to measure the size of a piece of software. The main reason to develop a size measurement method was the need to manage software development better. The projects failed too often in late 1970s. All the other industries were used to base their management practices and contracting on measurable and comparable deliveries, but the software development business didn‘t even have any applicable metrics.
Since 1979 technologies have improved, platforms have changed and use of software has increased everywhere, but still most of the managers are waiting for better measurement methods. And the projects fail too often.
In this paper the hidden development of Functional Size Measurement and other measurement methods is explained. Although none has invented anything better for size measurement in 25 years, the software project management concepts, knowledge databases and best practices have improved a lot, and finally it‘s time to adopt them into every day‘s software development business. The first serious pilots in Australia and Finland have been extremely successful. It is time to make software development industry the leading project management domain.

Measuring the size of application software overheads

Eberhard Rudolph

School of Computing and Information Technology,
Griffith University, Brisbane, Australia

erudolph@gmx.net

Abstract. Functional size has emerged as the ISO standard to quantify the size of application software. Functional size is derived from functional user requirements and specifically excludes technical or quality requirements and implementation style. In practice, however, the delivered software will have to include additional software functionality introduced by the IT environment, economics, user expectations, quality constraints, or developer's practices. All of this additional software functionality is excluded from the functional size count, remains unmeasured and therefore eludes management.
Using operational size as a size measure of all implemented software functionality this paper proposes to measure software overhead as the difference between operational (gross) and functional (net) software size. It is shown that minor adjustments to current functional size measurement methods will enable these methods to measure gross software size in addition to net software size.
As a result estimation of software development costs, particularly for maintenance and enhancement projects, will be more precise. Quantification of software overheads will assist the evaluation of different software implementations and the analysis of fatware.

The Functional Size eMeasurement Portal (FSeMP) A Web-based Approach for Effort Estimation, Benchmarking and eLearning

Mathias Lother, René Braungarten, Martin Kunz, Reiner R. Dumke

Software Engineering Group, Institute for Distributed Systems,
University of Magdeburg, Germany

{lother, braungar, makunz, dumke}@ivs.cs.uni-magdeburg.de
http://ivs.cs.uni-magdeburg.de/sw-eng/us/
http://fsemp.cs.uni-magdeburg.de

Abstract. According to a widely accepted consensus software measurement combined with belonging continuous training can make a substantial contribution to software development including aspects of process improvement and increasing software quality.
Beside other areas the application of measures in order to support software management decisions is of particular importance. Especially effort and size estimations as well as benchmarks are ways to assure the delivery of software in time and budget. Since these techniques are nontrivial training and consulting features are necessary in order to assist software developers and project managers in their work.
The growing trend of distributed software development leads to a growing demand of Web-based support of these methods. This paper introduces a Functional Size eMeasurement Portal using the Internet.s potential in order to provide an effort estimation, benchmarking and eLearning environment.

Software complexity evaluation based on functional size components

Luca Santillo

Data Processing Organization

luca.santillo@dpo.it

Abstract. This work introduces a self-consistent model for software complexity, based on information which can be typically collected at early stages of software lifecycle, e.g. in the functional specification phase, when a functional size measurement is also usually performed. The proposed model considers software complexity as structured into three bottom-up stages of the software architecture (from internal complexity of each base functional component to the overall structural complexity of the software system). Conceptually, complexity can be considered as proportional to dimensionality = scale × diversity, hence any complexity factor can be seen as an issue either of scale or of diversity, originating a corresponding specific factor of implementation difficulty, or complexity driver; such mapping suggests which factors could be referred to software structure and size, while other factors should be referred more adequately to the software development process. For each stage of the model, a description an one or more specific metrics – at different detail level – are proposed in order to provide an overall software complexity indicator to support in distinguish software systems from a complexity perspective. Moreover, it's shown how some or all of the complexity factors – at various stages – can be integrated with typical functional size measures (COSMIC, IFPUG) in order to normalize the latter in a more realistic software estimation process. A weighted profile scheme, or complexity mask, is also proposed to take into account the intrinsic differences when evaluating the complexity for systems of different type or domain.

Validating Metrics for UML Statechart Diagrams through a Family of Experiments

José A. Cruz-Lemus, Marcela Genero, Mario Piattini

ALARCOS Research Group
Computer Science Department, University of Castilla – La Mancha
Paseo de la Universidad 4, 13071 Ciudad Real (Spain)

{JoseAntonio.Cruz, Marcela.Genero, Mario.Piattini}@uclm.es

Abstract. In this paper we present a family of experiments that we carried out for validating a set of metrics that were defined for measuring the size and structural complexity of UML statechart diagrams, as early understandability indicators. The family consists of a controlled experiment and two replications performed with different Computer Science students and teachers.
The obtained results reveal that the metrics Number of Activities (NA), Number of Simple States (NSS), Number of Guards (NG) and Number of Transitions (NT) are highly correlated with the understandability time of the UML statechart diagrams.
Nevertheless, further validation is needed for obtaining stronger results.

A V&V Measurement Management Tool for Safety-Critical Software

Edgardo Palza, Alain Abran*, Christopher Fuhrmann*, Eduardo Miranda***

* École de Technologie Supérieure – ETS
1100 Notre-Dame Quest, H3C 1K3 Montréal, Québec, Canada

** Ericsson Research Canada
8500 Blvd. Decarie H4P 2N2, Montréal, Québec, Canada

edgardo.palza-vargas1@ens.etsmtl.ca, aabran@ele.etsmtl.ca, christopher.fuhrmann@etsmtl.ca, eduardo.miranda@ericsson.com

Abstract. This paper presents a V&V Measurement Management Tool (V&V MMT) to support the Management of V&V activities in the context of safety-critical software. We illustrate how V&V MMT can facilitate the quantification of the V& V processes, activities and tasks in projects recommended in the IEEE Standard for Software Verification and Validation (IEEE Std. 1012-1998) for facilitating the establishment of V&V measurement indicators: (1) Software Verification and Validation Plan (SVVP), (2) Baseline change assessment, (3) Management Review of V&V, (4) Management and Technical Review Support, (5) Interface with Organizational and Supporting Process.

Evaluating the Sensitivity of Coupling Metrics to Evolving Software Systems

F.G. Wilkie, M.P. Ware*, B.A. Kitchenham**, T.J. Harmer*[+]

* Centre for Software Process Technologies, Faculty of Engineering, University of Ulster, Newtownabbey, Co.Antrim BT37 0QB, Northern Ireland

** Department of Computer Science, Keele University, Keele, Staffordshire ST5 5BG, National ICT Australia, Locked Bag 9013 Alexandria, NSW 1435, Australia

[+] School of Computer Science, Queen's University of Belfast, Belfast, BT7 1NN, Northern Ireland

fg.wilkie@ulster.ac.uk, mp.ware@ulster.ac.uk, barbara@cs.keele.ac.uk, barbara.kitchenham@nicta.com.au, t.harmer@qub.ac.uk

Abstract. This paper is concerned with the use of coupling metrics and their sensitivity to change during software system evolution. Three coupling metrics are applied to a commercial C++ application across six releases over an eight year period. It is shown that one of the metrics is more sensitive to the underlying changes in the evolving software than the other two. The results appear to be contradicting Lehman's laws of systems evolution for e-type applications.

A System of References for Software Measurements with ISO 19761 (COSMIC-FFP)

Adel Khelifi, Alain Abran, Luigi Buglione

École de Technologie Supérieure – ETS, 1100 Notre-Dame Ouest, Montréal, Canada H3C 1K3

adel.khelifi.1@ens.etsmtl.ca, aabran@ele.etsmtl.ca, luigi.buglione@computer.org

Abstract. Software measurement is still emerging as a field of knowledge, and, most often, traditional quality criteria of measurement methods such as repeatability, reproducibility, accuracy and convertibility are not even investigated by software measurement method designers. In Software Engineering, the Functional Size Measurement (FSM) community has been the first to recognize the importance of such quality criteria for measurement, as illustrated in the recently adopted ISO document 14143-3; these criteria represent, however, only a subset of the metrology criteria which includes, for instance, measurement units and internationally recognized measurement references (e.g. .etalons.). In this paper, a design for building a set of normalized baseline measurement references for COSMIC-FFP (ISO 19761), the 2nd generation of FSM methods, is proposed. The goal is to design, for the first time in Software Engineering, a system of references for software FSM methods.

Implementing COSMIC-FFP as a replacement for FPA

Frank W. Vogelezang

Sogeti Nederland B.V.

frank.vogelezang@sogeti.nl

Abstract. This article shows the first results of the adoption of COSMIC Full Function Points as a sizing method replacing function point analysis. The main arguments why COSMIC-FFP was chosen will be explained, the transformation plan will be shown together with the first results of the use of COSMIC-FFP. Next to the management requirement that the new functional sizing method had to be a standard a number of practical requirements were essential before the transformation could start: to find a correlation between Cosmic functional sizing units and function points so that the existing figures for size and product delivery rate could be reused, (early) estimation possibilities and the use of COSMIC-FFP for sizing maintenance projects.

The Second Level Input Variables for Software Cost Estimation Models

Juan J. Cuadrado-Gallego[1], *Javier Dolado*[2], *Daniel Rodríguez*[3], *Miguel-Angel Sicilia*[4]

[1] Departamento de Informática, Universidad de Valladolid, 40005, Plaza de Santa Eulalia, 9, Segovia, España
jjcg@infor.uva.es

[2] Departamento de Lenguajes y Sistemas Informáticos, Universidad del País Vasco, 20009, P.M. Lardizabal, 1, San Sebastian, España
dolado@si.ehu.es

[3] Departament of Computer Science, University of Reading Reading, RG6 6AY, UK
d.rodriguezgarcia@rdg.ac.uk

[4] Departamento de Ciencias de la Computación, Universidad de Alcalá 28871, Ctra. de Barcelona, 33.6, Alcalá de Henares, España
msicilia@uah.es

Abstract. Among the main factors to obtain successful results making software cost estimations with Parametric Mathematical Models are the definition of the equations, the revision and the correct calibration of the input parameters, and finally, but not less significant, a wise selection of the numerical values of defined parameters for each project. Ever thinking that the accuracy of the model was solved working over the first three factors, he last factor has not been researched at the same level. But, if an erroneous numerical value is chosen for the input variables, the estimations done with the model will be erroneous, no matter how fine the model have been built. This paper point out this problem and give a way to solved it by a redefinition of the parameters in two levels.

ERP-Standard-Softwareanbieter im magischen Dreieck von Arbeitsweise, Kosteneffizienz und Produktqualität

Frank Schmeißner; Lutz Winkler

Imbus Rhain-Main GmbH (www.imbus.de)

Frank.Schmeissner@imbus.de, Lutz.Winkler@imbus.de

Zusammenfassung. Im Rahmen einer Benchmarkstudie werden in einem ersten Teil aussagekräftige Kennzahlen und Metriken erhoben zu:

- Produktqualität
  Ein Beispiel für eine entsprechende Kennzahl: Welchen Prozentsatz an Fehlern finden Sie **vor** der Freigabe und welchen Prozentsatz findet der Kunde im ersten halben Jahr **nach** Auslieferung seines Produktes?
- Kosteneffizienz im Test- und Qualitätsmanagement-Bereich
  Beispiele für Kennzahlen: Verhältnis Anzahl Tester zu Entwickler, und: durch-schnittlicher Aufwand für das Finden eines Fehlers vor Auslieferung (Summe aller Fehlervermeidungs- und Prüfaufwände).

Bezüglich des Einflusses des Reifegrades der verschiedenen SWEntwicklungsprozesse gemäß SPICE auf die Restfehlerrate eines SW-Produktes werden ergänzend zu dem Konzept der Benchmarkstudie die empirischen Ergebnisse eines EU-Forschungs-projektes vorgestellt (siehe http://www.imbus.de/produkte/pets.html).
Basierend auf 16 untersuchten Projekten und Anwendung von statistischen Methoden (Goodmans's und Kruskal's Gamma) konnte empirisch nachgewiesen werden, welche Aspekte für die Restfehlerrate ausschlaggebend sind.

Identifikation und Aufwandsschätzung neuer Anforderungen im Rahmen etablierter Integrationsarchitekturen

Andreas Schmietendorf[#], Daniel Reitz[#]*, Jens Lezius[#], Tillmann Walter[#]*

[#] T-Systems GmbH, Entwicklungszentrum Berlin . eSC/IS,
Wittestraße 30H, D-13509 Berlin
{andreas.schmietendorf, daniel.reitz, jens.lezius, tillmann.walter}@t-systems.com

* Otto-von-Guericke-Universität Magdeburg, Fakultät Informatik,
Institut für Verteilte Systeme, Postfach 41 20, D-39016 Magdeburg
{schmiete, reitz}@ivs.cs.uni-magdeburg.de

Zusammenfassung. Der vorliegende Artikel greift eine aus Sicht der Autoren bisher wenig beachtete Themenstellung auf. Dabei geht es um die Identifikation und Aufwandsschätzung von Integrationsanforderungen. In immer stärker werdendem Umfang gilt es, neue Anforderungen mittels Konfiguration und Integration bereits bestehender Anwendungs-architekturen zu realisieren und weniger darum, neue Anwendungssysteme zu entwickeln. Im Rahmen des Artikels werden bestehende Methoden und Verfahren zur Prognose des Projektaufwands hinsichtlich der Verwendbarkeit untersucht, der Zeitpunkt für eine notwendige Aufwandsschätzung aufgezeigt und eine bereits etablierte Vorgehensweise (Impact Assessment) zur Identifizierung von Integrationsanforderungen bzw. einer daraus resultierenden Aufwandsschätzung vorgestellt. Darüber hinaus wird eine optimierte und vor allem strukturierte Vorgehensweise zur Diskussion gestellt.

Estimating maintenance projects using COSMIC-FFP

Tom Koppenberg, Ton Dekkers

Sogeti Nederland B.V.

tom.koppenberg@sogeti.nl, ton.dekkers@sogeti.nl

Abstract. A large number of software projects are enhancement projects of existing software. For estimating new projects acceptance of COSMIC Full Function Points is rapidly growing because it has already proven to be a good alternative for Function Point Analysis. Estimating enhancements using classic Function Point Analysis has always been somewhat controversial, but we believe that COSMIC can be a very good alternative in the very near future.

A Field Study of Software Functional Complexity Measurement

De Tran-Cao, Ghislain Lévesque, Jean-Guy Meunier

University of Quebec in Montreal

Tcde@cit.ctu.edu.vn, Levesque.ghislain@uqam.ca, Meunier.jean-guy@uqam.ca

Abstract. Numerous measurement methods (metrics) have been proposed to measure software complexity, but these have been criticized for their lack of a theoretical model which would serve as a guide for measurement methods. To fill this gap, we use Wood.s task complexity model as a theoretical model which will make it possible to capture and quantify software functional complexity. We propose a framework that captures three aspects of software complexity: data movement complexity, data manipulation complexity and system complexity. Data movement complexity is measured by the number of data groups (NOD) that move in and out of a functional process. Data manipulation complexity is measured by the number of conditions (NOC) on inputs to produce different expected outputs and the system complexity is measured by the entropy of system (EOS). An empirical study using 15 software maintenance projects of a telecom company shows that the three proposed measures are relevant. Moreover, the effort estimation model, which takes into account both data movement complexity and data manipulation complexity, is better than the model built on the linear regression between maintenance effort and COSMIC-FFP size.

Estimation Models based on Functional Profiles

Alain Abran, Blanca Gil, Éric Lefebvre

École de Technologie Supérieure - Université du Québec, Montréal (Canada)

aabran@ele.etsmtl.ca, blanca.gil.1@ens.etsmtl.ca, elefebvre@ele.etsmtl.ca

Abstract. As a software functional size method, Function Point Analysis (FPA) has been used in many organizations for measuring productivity and building estimation models. FPA is based on a number of function types (external inputs, external outputs, external inquiries, internal logical files and external interface files), the set of which we refer to as a functional profile. It has been observed that a majority of projects within a sample are close to having an average functional profile, while some, of course, can be considered as outliers. This paper investigates the functional profiles within the ISBSG international repository, and whether or not productivity varies with such profiles. The results of the statistical analyses lead to distinct estimation models, depending on whether or not a project functional profile is within a reasonable range of the average functional profile for any particular sample.

Eine einheitliche Kohäsionsmetrik für Methoden, Klassen und Komponenten

Johannes Drexler, Francesca Saglietti

Lehrstuhl für Software Engineering, Universität Erlangen-Nürnberg

{jdrexler|saglietti}@informatik.uni-erlangen.de

Abstract. The cohesion metric derived in this article takes into account cohesion properties on different hierarchical levels with respect to the underlying building block composition. In particular, cohesion measures at the high level of components are based on corresponding properties at the intermediate class level, the latter ones being derived by considering cohesion at the lower level of methods. In this sense, this article offers a novel contribution to cohesion measurement. Based on this approach, a tool was implemented supporting the automatic determination of the class cohesion metric proposed.

Mit Code-Metriken Wartungskosten senken: Controlling technischer Qualität

Ute Richter (MMS), Frank Simon (SQS)

T-Systems Multimedia Solutions GmbH, Dresden;
Software Quality Systems AG, Cologne

Ute.Richter@t-systems.com; Frank.Simon@sqs.de

Zusammenfassung. In heutigen Projekten mit immer kürzeren Time-To-Market-Zyklen und einer zunehmend wichtigen Mitarbeiteraustauschbarkeit (z.B. für Offshoring) ist die technische Qualität, die Eigenschaften wie Wartbarkeit, Code-Lesbarkeit oder Einhaltung einer technischen Architektur zusammenfaßt, ein wichtiger Schlüssel für langfristig erfolgreiche Software-Projekte. Ohne ein entsprechendes Controlling-Instrument bleibt die aktuelle technische Qualität aber unbekannt und kann daher weder konkret bestimmt noch optimiert werden.
In diesem Papier wird die erfolgreiche Einführung eines solchen Controlling-Instruments beschrieben: Mittels sehr spezifischer, von verschiedenen Werkzeugen errechneter Metriken und deren kontinuierlichen, projektübergreifenden Betrachtung konnte die technische Quali-tät transparent gemacht werden; die darauf aufbauende Optimierung hat bereits nach einem Jahr 20% der Wartungskosten bei einem Großteil der im Code-Control-Cockpit enthaltenen Projekte eines großen Software-Hauses eingespart.
Beschrieben wird in diesem Papier die notwendige organisatorische Vorarbeit eines solchen Controlling-Instruments, die technische Umsetzung und die konkreten erfahrenen Ergeb-nisse.

Software Maintenance Productivity measurement: How to assess the readiness of your organization

Alain April[1], *Alain Abran*[1], *Reiner R. Dumke*[2]

[1] École de Technologie Supérieure, Montréal, Canada
aapril@ele.etsmtl.ca, aabran@ele.etsmtl.ca

[2] Otto-von-Guericke-University of Magdeburg, Germany
dumke@ivs.cs.uni-magdeburg.de

Abstract. Software maintenance constitutes an important part of the total cost of the life cycle of software. Some even argue this is the most important fraction of the cost (5080% according to Tony Scott, 75% according to Rand P. Hall. The added value of software maintenance is often not **fully** understood by the customer, however, leading to a perception that software maintenance organizations are costly and inefficient. A common view of maintenance is that it merely fixes bugs. However, studies over the years have indicated that in marry organizations the majority of

the maintenance effort, over 80% is devoted to value-added activities (Sommerville, Pressman, Pigoski. To improve customer perceptions of software maintenance, it is important to provide them with better insight into the activities performed by the maintenance organization and to document such performance with objective measures of software maintenance activities.
In this paper, the prerequisites for software maintenance productivity analysis are described with the use of experiences of the Bahrain Telecommunications Company (Batelco) during the year 2001-2. First, the differences between software maintenance activities and IS development projects are described. Then, a basic trend model is applied as well as ways to manage the expectations of customers. To conclude, some remarks are made regarding the application of productivity analysis for software maintenance managers.

Web Tomography
Towards e-Measurement and e-Control

Dumke, R., Lother, M., Schäfer, U., Wille, C.

Otto-von-Guericke-Universität Magdeburg,
Postfach 4120, 39016 Magdeburg, Germany

{dumke,lother,schaefer,wille}@ivs.cs.uni-magdeburg.de

Abstract. The following paper considers the general aspects of software e-measurement and the special forms and approaches of Web-based measurement as Web tomography. Following a discussion of some theoretical characteristics we present an example of tomographical Web analysis using our tool implementation WEBTOMIX. While describing the possibilities of the determination of Web technologies, we demonstrate some useful approaches of Web measurement and evaluation of Web-based systems.

Software Measurement Body of Knowledge – Initial Validation using Vincenti's Classification of Engineering Knowledge types

Alain Abran, Luigi Buglione, Asma Sellami

École de Technologie Supérieure – ETS,
1100 Notre-Dame Ouest, Montréal, Canada H3C 1K3

aabran@ele.etsmtl.ca, Luigi.Buglione@computer.org,
asma.sellami.1@ens.etsmtl.ca

Abstract. The Guide to the SWEBOK (2001 Trial version) currently contains ten distinct software engineering Knowledge Areas (KAs) and three common themes: Quality, Tools and Measurement. The Measurement topic is pervasive throughout all the KAs (in both the 2001 and 2004 editions). An initial taxonomy for a new specific KA on Software Measurement had been proposed in 2003. To improve this initial proposal, the Vincenti classification of engineering knowledge types was used as an analytical tool. This paper presents a revised breakdown for a body of knowledge on Software Measurement.

Portfolio-Management for Software Projects

Christof Ebert

Alcatel, Paris, France

Christof.ebert@alcatel.com
www.alcatel.com

Abstract. Portfolio Management targets the delivery of the right products at the right time for the right markets. It applies to IT development as well as software development. The main success factor for successful portfolio management in IT or software development is to understand the own position within the enterprise, while considering the intrinsic challenges in software management, such as resource dependencies, process capability and productivity, and quality impacts. This article presents a simple yet effective approach to portfolio management. It consists of three steps, namely the extraction of project information, the evaluation of these projects, and finally the decision in which projects to further invest. For each step we present insight how to make it a success and concrete practical experiences on how to implement for software projects. We present several real-world scenarios, which show how to practically line up software portfolio management with market evolution, risk management or technology management. We look especially on the integration of portfolio management with product life-cycle management. A software product release portal is presented to visualize project results and allow impact analyses in their mutual dependencies. Finally we offer some hints on presenting key figures.

Metrics for Cooperative Development Processes

Dr. Thomas Fehlmann

Euro Project Office AG, Zurich, Switzerland

Thomas.Fehlmann@e-p-o.com

Abstract. The /ch/open Process is a collection of development methodologies and software development best practice. It is available under the GNU Open Documentation licence and has become popular among software developers in Switzerland.
The methodology integrates software metrics, combinatory metrics, the "Six Steps to Completion" metric for project tracking, and Design for Six Sigma. This framework is particularly useful for agile software development, where metrics are key for successful communication within the development group and between developers, users and sponsors.
This paper summarizes experiences with metrics in collaborative software development environments.

Verification and validation of a knowledge-ased system

Jean-Marc Desharnais[1], *Alain Abran*[1], *Julien Vilz*[2],
François Gruselin[2], *Naji Habra*[2]

[1] Départment de génie électrique,
École de Technologie Supérieure (ETS)
jmdeshar@ele.etsmtl.ca, aabran@ele.etsmtl.ca

[2] Facultés Universitaires Notre Dame de la Paix à Namur,
Université de Namur
jvi@info.fundp.ac.be, FGruselin@xpectis.com,
nha@info.fundp.ac.be

Abstract. In the real world, a knowledge-based system (KBS) must often accommodate a considerable number of references which support the particular knowledge domain. The size of such a knowledge repository makes its detailed verification challenging and subsequent maintenance onerous. New technology can help improve both the verification and maintenance of these knowledge repositories. To investigate the effectiveness of new technologies for verification and maintenance, we developed two subsequent versions of a KBS designed to improve the consistency of software measurement using ISO 19761 (the COSMIC-FFP measurement method for software functional size) and COSMICFFP guide.
The COSMIC-FFP KBS consists of a hybrid knowledge system built on casebased and ruled-based approaches. The first prototype was built in 2000 using Microsoft technology (Visual Basic 6, Access 2000, hyperlink facilities for RTF files). This prototype included 105 case problems and almost 800 files (hyperlinks) for the required references. Because of the high number of files, the verification and validation of this KBS was, of course, very challenging. This led us to design a second KBS, a Web-based prototype (XML, XSL, Java Server Page) which is much easier to verify and validate, leading to considerably improved maintainability and expandability.
This paper presents an overview of the selected hybrid KBS approach and of the first and second prototypes. It also illustrates the transitioning and quantitative benefits for the detailed KBS verification and validation process and the lessons learned during this process.

Evaluation of the Agent Academy: Measurement Intentions and Results

Cornelius Wille[1], *Reiner R. Dumke*[1], *Nick Brehmer*[2]

[1] Otto-von-Guericke-Universität Magdeburg, Institut für Verteilte Systeme,
Universitätsplatz 2, 39106 Magdeburg, Germany
[wille|dumke]@ivs.cs.uni-magdeburg.de

[2] M-BIS GmbH, Sandtorstrasse 23, 39106 Magdeburg, Germany
nick.brehmer@m-bis.de

Abstract. In the recent years the agent technology has emerged as a new paradigm for software development. Starting from personal user agents today intelligent and autonomous agents often interact in multi agent system with dynamic changing environments. To become a part of the mainstream software development process the agent-oriented software needs a strong quality evaluation to be verified. The presented work introduces the Agent Academy as a multi agent based expert system for developing and training of intelligent agents. Outgoing from existing object-oriented metrics and quality models an agent-oriented approach will be presented. Examples introduce how software measurement can be used to evaluate the maintainability of agent-oriented software.

Lack of Consensus on Measurement in Software Engineering: Investigation of Related Issues

Pierre Bourque[1], *Sibylle Wolff*[2], *Robert Dupuis*[3], *Asma Sellami*[1], *Alain Abran*[1]

[1] École de Technologie Supérieure – ETS,
1100 Notre-Dame Ouest, Montréal, Canada H3C 1K3

[2] SAP Labs Canada

[3] Université du Québec à Montréal

pbourque@ele.etsmtl.ca, sibylle.wolff@sap.com,
dupuis.robert@uqam.ca, asma.sellami.1@ens.etsmtl.ca,
aabran@ele.etsmtl.ca

Abstract. Even though measurement is considered an essential concept in recognized engineering disciplines, measures in software engineering are still far from being widely used. To figure out why software measurement has not yet gained enough peer recognition, this paper presents a set of issues that still have to be addressed adequately by the software measurement community. These issues were derived from the analysis of comments obtained during two Delphi studies and a Web-based survey conducted to identify and reach a consensus on the fundamental principles of the discipline within the international software engineering community. The paper also discusses the application of metrology concepts as a research direction to address some of the measurement issues identified.

The Versatility of Software Defect Prediction Models (or why it's so hard to replicate related Case Studies)

Marek Leszak

Lucent Technologies Bell Labs, Thurn-und-Taxis-Str. 10, 90411 Nuernberg, Germany

mleszak@lucent.com

Abstract. This paper is based on the author's previous paper at IWSM2002 in which several process metrics of an industrial large-scale embedded software system, the Lucent product LambdaUnite™ MSS, have been analyzed. This product is a large embedded hardware/ software system for the metropolitan and wide-area transmission and switching market. The previous paper has been expanded in the following directions: The analysis of defect data includes all major (i.e. feature) releases till mid of 2004 (the IWSM2002 paper covered releases till 2002), so that with the metrics considered, the persistence of the validity of the previously published results can be checked. Several defect metrics on file-level (previous paper: subsystem-level) have been defined and analyzed, as basis for a defect prediction model. Main analysis results: Faults and code size per file show a very weak or no correlation. Portion of faulty files per release tend to decrease across releases. Size and error-proneness in previous release alone is not a good predictor of defects per release, a more sophisticated statistical model, e.g. multivariate analysis, would be needed. Customer-found defects are strongly correlated with number of delivered systems and with pre-delivery MRs per subsystem.

Software Reliability – Grundlagen und Berechnung

Hans-Georg Hopf

Georg-Simon-Ohm-Fachhochschule Nürnberg

Hans-goerg.hopf@fh.nuernberg.de

Abstract. This article introduces a method, originally suggested by John Musa, to estimate the software reliability of a program. The reliability calculation is based on an evaluation of the test effort. At first a reliability goal is defined specifying a final failure intensity. Based on the present failure intensity the amount of test required and the failure correction workload is calculated. The estimation could be compared to the test experience.

Measuring Components' Unused Members

Hamdan Msheik[1], *Alain Abran*[1], *Hamid Mcheick*[2], *Pierre Bourque*[1]

[1] Electrical Engineering Department, École de Technologie Supérieure,
1100 Notre-dame Ouest, Montréal (Québec), Canada, H3C 1K3
hamdan.msheik.1@ens.etsmtl.ca, aabran@ele.etsmtl.ca,
pierre.bourque@ele.etsmtl.ca

[2] Department of Computer Science, Université du Québec à Chicoutimi
555, Boulevard de l'Université, Chicoutimi (Québec), Canada, G7H 2B1
hamid_mcheick@uqac.ca

Abstract. Currently, components technology represents a major step in the evolution of software technology as a whole. Although it has been undergoing continuous enhancement, this technology suffers from a number of limitations: in particular, components' unused functionalities. For instance, a software component incorporates a set of functions of which a size-varying subset is actually used to satisfy the functional requirements of a particular software application. Consequently, a subset of unused functionalities will persist in the deployed application. This subset of unused functionalities provides no functional value to the hosting application. Furthermore, these unused functionalities consume memory and network resources and might compromise application security if they are exploited inappropriately. In this paper, we propose CUMM (Components' Unused Members Measurement), a method to measure components' unused members (attributes and functionalities), and their memory consumption inside a software application. Furthermore, we present a set of analysis models which use the results of the CUMM to determine percentages of unused members as well as the degree of generality of a component's members.

A Measurement Service for Monitoring the Quality Behaviour of Web Services offered within the Internet

Andreas Schmietendorf [#]*, *Reiner Dumke**

[#] T-Systems International, Entwicklungszentrum Berlin, Wittestraße 30G,
D-13476 Berlin, Germany
andreas.schmietendorf@t-systems.com

* Otto-von-Guericke-Universität Magdeburg, Fakultät Informatik,
Institut für Verteilte Systeme, Postfach 4120, 39016 Magdeburg, Germany
{schmiete, dumke}@ivs.cs.uni-magdeburg.de

Abstract. Within this paper we propose the idea of an independent measurement service. For this we explain the developed concept and a first prototypical implementation. By the help of the measurement service everybody is able to evaluate the quality behavior, of within the internet provided Web Services. Based on such evaluations it is possible to certificate offered Web Services and to support the selection process during the implementation of Web Service based solutions. Furthermore the measurement service should provide his functionalities as Web Service, therefore it is possible to support a dynamic choice of offered services.

An Analysis of the Mc Cabe Cyclomatic Complexity Number

Alain Abran[1], *Miguel Lopez*[2], *Naji Habra*[3]

[1] ETS – U. of Québec, Canada
aabran@ele.etsmtl.ca

[2] Cetic, Belgium
vp@cetic.be

[3] University of Namur, Belgium
nha@info.fundp.ac.be

Abstract. On basis of a framework defined to understand, structure, compare and analyze the different measurement approaches presented in software engineering literature, this paper develops a detailed analysis of the well-known McCabe Cyclomatic complexity number. It highlights in particular some misconceptions underlying this measurement approach and also points out the necessity to have well grounded definitions and models for the measurement methods the practitioners are applying in the software industry.

Usability-Metriken als Nachweis der Wirtschaftlichkeit von Verbesserungen der Mensch-Maschine-Schnittstelle

Bela Mutschler, Manfred Reichert*[+]

* DaimlerChrysler Forschung, REI/SP, 89013 Ulm

[+] Universität Ulm, Abteilung Datenbanken und Informationssysteme, 89069 Ulm

bela.mutschler@daimlerchrysler.com
reichert@informatik.uni-ulm.de

Abstract. The adequate design of user interfaces is crucial for the acceptance of any software system. Methods for usability engineering (including usability metrics for measuring the impacts of applied changes) gain an increasing interest in both academia and industry. In this context usability metrics can be applied to evaluate whether the integration of usability engineering methods in existing software development processes results in a better usability and user acceptance. In many cases an economic-oriented evaluation is only based on a single business ratio like return on investment. This paper gives a more systematic motivation regarding the correlation between usability engineering methods and the added value to a software project. Usability metrics have proven as a suitable method for this purpose.

Kreative Software-Messung – Kleiner Leitfaden Statistischer Tricks

Prof. Dr.-Ing. Jörg Robra

Georg-Simon-Ohm-Fachhochschule Nürnberg

joerg.robra@t-online.de

Abstract. Software measurements are expensive in time and cost; thus they are performed only if any advantages are expected, e.g. the proof that some product or artefact has been improved by new

measures, development times have been shortened, costs reduced, a process is better to control, or important predictions are available earlier and more precise. Thus every software measurement is a candidate for manipulations with the aim to produce the expected results – may be unconsciously, by ignorance of statistic pitfalls, or with intent. Goal of this contribution is to sharpen the eyes for these measures to control the own sub-conscious, to avoid own statistic mistakes and to recognise else's manipulations.

Bewertung der Gebrauchstauglichkeit mit Metriken

Prof. Dr. Rüdiger Liskowsky

Technische Universität Dresden, Fakultät Informatik,
Lehrgebiet Programmierumgebungen und Werkzeuge

rl2@inf.tu-dresden.de

Abstract. This paper describes approaches to measure the usability of software based on how the user is confronted with the user interface. In order to obtain reliable results, the requirements of the measurement theory build the foundation for these approaches, as to obtain reliable results.
The knowledge collected in the area of software metrics is the base for the creation of significant usability metrics. The term usability metrics follows directly from the definition of usability and the measurable criterions postulated from it. These criterions build the basis for the systematic classification of usability metrics according to the metrics for software design. From this classification further metrics can be derived which represent a reliable, quantified criterion for tests of standard conformance. The paper considers tools to support determination of usability metrics as well.
The final part of the paper contains the derivation of several still incomplete metrics based on eye movement recording with an eyetracker device. A short reflection on measurement uncertainty of metrics closes the paper.

Guideline for the application of COSMIC-FFP for sizing Business applications Software

Arlan Lesterhuis

Sogeti Nederland B.V.

arlan.lesterhuis@sogeti.nl

Abstract. The COSMIC-FFP functional sizing method can be applied to several software domains (such as business applications and 'real-time' software). The Measurement Manual offers a theoretical skeleton, together with examples from those domains. For each specific software domain a Guideline will be developed. A Guideline aims to describe detailed rules and to provide extensive examples for the sizing of software from that specific domain.
The first Guideline to be published gives a characterisation of the business applications domain. To apply the COSMIC-FFP method to business applications software the measurer requires a good

understanding of data analysis. The Guideline gives a short explanation of data analysis and its relation with the COSMIC-FFP method. Finally, the application of the COSMICFFP method in the domain is explained for the End-user Measurement Viewpoint and the Developer Measurement Viewpoint, as defined in the COSMIC-FFP Measurement Manual.
The Guideline discusses the materials of the COSMIC-FFP method (boundary, layer, object of interest, identification of data movements etc.), their manifestation in the business applications domain together with many examples.
Measurement of maintenance, the extended definition of object of interest and its implications for sizing are treated. Besides, several issues as sizing authorisation, help, logging, menus and layouts are treated.

Information Theory-based Functional Complexity Measures and Functional Size with COSMIC-FFP

Alain Abran, Olga Ormandjieva**, Manar Abu Talib***

* École de Technologie Supérieure - ETS,
1100 Notre-Dame Quest, Montréal, H3C 1K3, Canada
aabran@ele.etsmtl.ca

** Concordia University,
1455 de Maisonneuve Blvd. W., Montréal, Quebec H3G 1M8, Canada
ormandj@cse.concordia.ca, m_abutal@cse.concordia.ca

Abstract. This paper presents an exploratory study of related concepts across information theory-based measures and functional size measures. Information theory-based software measurement has been used in the design of an entropy-based measure of functional complexity in terms of an amount of information based on some abstraction of the interactions among software components. As a functional size measurement method, COSMIC-FFP, adopted in 2003 as the ISO/IEC 19761 standard, measures software functionality in terms of the data movements across and within the software boundary. In this paper, we explore some of the links between the two types of measures, and, in particular, the similarities (and differences) between their generic model of software functionality, their detailed model components taken into account in their respective measurement processes and, finally, their measurement function. Some further investigation avenues are also identified for extending the use of functional size measures for reliability estimation purposes and for scenario-based black-box testing.

Function Points Analysis based on requirement specification, a Case Study

Carlos Granja[1], *Angel Oller*[2]

[1] Research Group on Languages and Information Systems and Software Engineering,
Granada University, C/ D. Saucedo A. s/n. E18071- Granada, Spain
jcgranja@ugr.es

[2] Telefonica Moviles S.A.
oller_a@tsm.es

Abstract. In the function points analysis based on requirement specification, is a case study. This study is based on three real-world projects; a comparative analysis of these projects was possible because they were carried out for a single problem domain, followed the same development methodology and were subject to equivalent restrictions. Each project, performed by different groups, addressed and solved the same problem, but from different perspectives. Thus, the comparative dimension of the analysis enabled significant results to be achieved, in software metrics. The author include your experience in ISO JTC1 SC7 Working Group 12 in functional size measurement series and related international standards, and as coeditor in SQuaRE and COTS projects.

Zwischen Wunsch und Wirklichkeit: Einführung von Kennzahlen-systemen in die IT-Projekt- und Unternehmenssteuerung

Dipl.-Math. Jürgen Bach, Dipl.-Kfm. Björn Petersdorf

bach consulting GmbH, Bundeskanzlerplatz 2-10, 53113 Bonn, Germany

juergen.bach@bachconsulting.de,
bjoern.petersdorf@bachconsulting.de

Abstract. This lecture will focus on the possibilities of metrics (i.e. function points) to serve as connection between IT and other departments in companies. Focussing on IT cost control and IT added value, the authors speak about their successful implementations of key figure systems in companies of all industries. To support organisations in their process of gaining more in-depth understanding and knowledge of their IT, bach consulting has developed METORI, the maturity model for the usage and implementation of key figure systems. It integrates the different learning phases organisations go through when implementing key figure methodologies.

Reusage Knowledge on Process Flexibility for Developing Measurement Programs

Olga Jaufman

DaimlerChrysler AG, Ulm, Germany

Olga.Jaufman@DaimlerChrysler.com

Abstract. At present, software development often has to be carried out in a highly dynamic environment characterized by new innovations, new business relationships, and other frequent changes. In order to quickly react to the changes, software engineers desire to have flexible software development processes. In order to define a sufficiently flexible software development process, the criteria and measures for process flexibility are needed. The challenge is the different understanding of process flexibility by project stakeholders. Therefore we support the characterization of process flexibility by proposing knowledge related to process flexibility and a process for definition of the needed process flexibility from project stakeholders' point of view.

How do we measure process improvement? – Examples From Industry

Melanie Ruhe

SIEMENS AG, CT SE3, Munich, Germany

melanie.ruhe@siemens.com

Abstract. The systematic application of development processes is the key to long-term project success. This involves the selection of suitable processes as well as a cyclic analysis and improvement of the selected process. The sponsor of process improvement activities – usually the management - requires an analysis on the return of investment, i.e. the measurement of the success of process improvement. How do we measure process improvement? The literature provides some guidelines and theory to the information needs. However, in practice it is usually difficult to satisfy the management with suitable as well as expressive measures and reports. The paper provides practice examples of both objective and subjective measures implemented at Siemens AG on a detailed level. From these experiences it can be said that organizations with low CMMI level have generally more problems with evaluating the benefits of process improvement. Mature organizations have developed tailored measurement models. In general the experiences show that it is important to implement easy measures, which relate to the basic project planning & controlling goals, i.e. quality, costs and time, also in the area of evaluating process improvement benefits.

IT Governance requires quantitative (project)management

Ton Dekkers

Sogeti Nederland B.V.

ton.dekkers@sogeti.nl

Abstract. Instead of a demand market for IT services we now facing a supply market. Return on investment is more relevant for business then before. Financial affairs in some areas make managers cautious, value for money and transparency are now keywords. Business demands more and more governance and the same applies for IT nowadays. We need IT-governance as much as we need corporate governance. And at best, both governance practices arise from the same shared model for managing information and related technology (CobiT). The operationalisation of certain IT-governance aspects requires quantitative (project)management.

Patterns of success or failure in ERP requirements engineering: an empirical study

Maya Daneva

TELUS Mobility

mdaneva@computer.org

Abstract. The application of standard maturity models, that indicate both qualitative and quantitative assessments, can help organizations identify more successful and less successful practices impacting the outcome of requirements engineering (RE) process adoption. This paper uses maturity assessments to examine variations in instantiations of a generic off-the-shelf RE process. We built on previously published results that reflect the lessons learnt from five years of experience in using generic off-the-shelf RE processes in Enterprise Resource Planning (ERP) projects.

Proposals for project collection and classification from the analysis of the ISBSG Benchmark 8

GUFPI-ISMA SBC (Software Benchmarking Committee):
Domenico Natale[1], *Luca Santillo*[2], *Italo Della Noce*[3], *Monica Lelli*[2], *Stefania Lombardi*[4], *Guido Moretto*[5], *Simonetta Ortona*[1]

[1]SOGEI, [2]DPO, [3]Provincia Autonoma di Trento, [4]FINSIEL, [5]InfoCamere

dnatale@sogei.it, luca.santillo@dpo.it, italo.dellanoce@provincia.tn.it, monica.lelli@dpo.it, s.lombardi@finsiel.it, guido.moretto@infocamere.it, sortona@sogei.it

Abstract. This work presents statistical analysis scenarios of a sample of software development and enhancement projects contained in the Benchmark 8 by ISBSG (International Software Benchmarking Standards Group, 2003). The research – started in the second half of 2003 – is an on-going, incremental process based on the voluntary participation of the members of the SBC (Software Benchmarking Committee) of GUFPI-ISMA (Italian Software Metrics Association).
This work is focused on the distribution of development and enhancement projects in the ISBSG sample with regard to: sizing method, project size, completion date, productivity, and primary programming language.
Some considerations and hints arise from this research to improve the data collection process (by ISBSG as well as by any private organization within its own benchmarking database). Categorization criteria and taxonomies are identified and suggested for standardized use in order to clarify the definition and the comparison of future software projects.

Building fault prediction models from abstract cognitive complexity metrics – analyzing and interpreting fault related influences

Roland Neumann[1], *Stamatia Bibi*[2]

[1] Hasso-Plattner-Institute for Software Systems Engineering GmbH
at the University Potsdam
neumann@hpi.uni-potsdam.de

[2] Aristotle University of Thessaloniki, Greece
sbibi@csd.auth.gr

Abstract. Finding software defects as early as possible is a critical task that saves values. Fault prediction models identify faults by spotting error prone components, exploring design guidelines and thus directing test effort. These models gain knowledge from past mistakes to prevent future ones. This paper describes, applies, evaluates and compares modelling techniques of fault related code structures based on abstract complexity metrics. These complexity metrics are calculated from base metrics for all complexity aspects assuring statistical independence. Modelling techniques applied are MARS, Classification and Regression Trees, Association Rules and Bayesian Belief Networks. Their ability to interpret fault reasons and predict future possible faults is compared in this paper.

Metrics Based Project Governance

Pam Morris (BSc.Grad Dip Comp.Dip Ed)

Total Metrics (Australia)

Pam.Morris@Totalmetrics.com

Abstract. This paper describes a rigorous approach to software development project control by introducing functional size measurement at the planning stage and objectively quantifying the status and scope of the project and its deliverables throughout its lifecycle. The Scope Manager's role is both that of 'quantity surveyor' and project auditor. The paper defines this role and describes the process and the benefits of the metrics they bring to a project. The paper discusses how organisations typically do not harness the potential of the metrics personel in their organisation to pro-actively assist project teams manage risk. The engagement of independent Scope Managers on a project assists in assuring successful projects or providing early warning of projects in trouble.

Halstead's Software Science in Today's Object Oriented World

Da Yu Li, Victoria Kiricenko, Olga Ormandjieva

1 Introduction

Software measurement is a discipline emerging from the engineering character of developing software in early 1970's. The aim of the software measurement is to provide an objective feedback on the quality characteristics of software development artifacts.

From 1970's, efforts were paid to establish empirically predictive theories to support quality assurance, quality control, and quality prediction [1]. Earliest work was carried out by Halstead [6] and McCabe [10]. The common object of their research was the source code of programs. However, their approaches were completely distinct from each other, although they are both categorized as structural measurement of software. The two typical measurements form a watershed of such sort of methods.

Halstead's Software Science is more physical. In his theory, he directly measured the number of symbols N, the number of sorts of unique operators η_1, the number of sorts of unique operands η_2, and the number of modules M, deduced a set of metrics such as program volume V, language level L, etc, and estimated programming effort E and implementation time T. Halstead tried to explain all programs at the height of language level.

Alternatively, McCabe concentrated on the internal logical relationship between program statements. He believes a program is not just as simple as a combination of symbols, the organic interaction between statements makes the program more complex than it appears. McCabe's complexity measure includes cyclomatic complexity measure $V(G)$ and essential complexity measure $EV(G)$. Based on graph theory, Fenton and Whitty (1986) proposed the theory of decomposition tree, in which a complex flow graph is decomposed into a unique hierarchy of primes.

Back at the early days of the software measurement field McCabe's and Halsted's works were equally important and popular. However, today while the McCabe's cyclomatic complexity measure is still in use, even though, it is said to be too fine grinded to be powerful as before, the Halstead's software science measures are hardly seen in use.

In this paper we examine some of the vast number of the software science measures presented by Maurice H. Halstead and discuss their applicability and potential usefulness in the today's environment. Over the years, many researchers and practitioners criticized Halstead's work and, while some of the points made by them are valid, many are irrational and lack evidence to support the conclusions drawn. The critiques mainly fall in the following 5 categories: (1) developed in the context of assembly languages and too fine grained for modern programming languages; (2) the treatment of basic and derived measures is somewhat confusing; (3) the notions of time to develop and remaining bugs are arguable; (4) unable to be extended to include the size for specification and design; and (5) the experiments used to test many of hypotheses were very poor. Peter G. Hamer and Gillian D. Frewin [7] stated

that foundations of Software Science are very week, both theoretically and experimentally and questioned its usefulness for even POL. Carper Jones [8] pointed out that Halstead's measures were out of use. But he presented his point of view without formal prove.

The emerging of object-oriented languages (OOL) has greatly changed the way of programming, substituting the procedure-oriented languages (POL) of the near past. Consequently, new software measurements are being developed to be applied to the new OO paradigm. For instance, a branch of new measures for OO structure were developed by Chidamber and Kemerer [3] to characterize the OO design quality attributes, namely: weighted methods per class (WMC); depth of inheritance tree (DIT); number of children (NOC); coupling between object classes (CBO); response for class (RFC); and lack of cohesion metric (LCOM). As OOL is originated from POL, some OO measures are inherited from the existing work.

Is Halstead's theory really out of time in OO world? This is the question we are addressing in this paper.

The paper is organized as follows. In the Section 2 we will briefly enumerate part of Halstead's work on software measurement and analyze some of the critiques of his work. In the Section 3 we discuss the potential that the Halstead's measurements have in the OO environment and outline proposed modifications to adapt them to today's OOL. Section 4 concludes.

2 Halstead's Models of Software Metrics

Although there are thousands of languages in the world, each language has a variety of distinct symbols. But when they are studied under the point of view of software science, the sorts of symbols of the language are astonishingly few – only 2, those are the operators and the operands. More specifically speaking, tokens with a value are operands. Let's denote η_1 the number of unique operators and η_2 the number of unique operands in a language. Therefore we get the total vocabulary of language L as follows:

$$\eta = \eta_1 + \eta_2 \qquad (1)$$

Based on these 3 very basic counts, Halstead implemented his method of measuring and predicting of software quality.

2.1 Program length

Halstead has defined the following direct measures:

$f_{1,j}$ = number of occurrences of the jth most frequently used operator, while $j = 1,2, \ldots , \eta_1$,
$f_{2,j}$ = number of occurrences of the jth most frequently used operand, while $j = 1,2, \ldots , \eta_2$.
N_1 = total usage of all of operators appearing in that implementation.
N_2 = total usage of all of operands appearing in that implementation.

Then the following relations are derived:

$$N_1 = \sum_{j=1}^{\eta_1} f_{ij} \qquad (2)$$

$$N_2 = \sum_{j=1}^{\eta_2} j_{2j} \qquad (3)$$

And the length N of the program is

$$N = N_1 + N_2 = \sum_{i=1}^{2} \sum_{j=1}^{\eta_i} f_{ij} \qquad (4)$$

The program length as defined in the software science is clearly conceptually very close to the other well known length measure – count of executable statements, which is considered to be a theoretically valid and commonly used in practice today. More over, classical software science length measure, despite its simplicity, achieves more than just counting number of executable statements, it reflects more accurately difference between complexities of short and long statements. However, this measure was many times criticized and is completely out of fashion.

2.2 Program volume

An important characteristic of an algorithm is its size. However, whenever a given algorithm is translated from one language to another, its size changes and, thus, the size metric could not be independently used to characterize an algorithm. In order to enable comparison between different implementations of the same algorithm and comparison between different programming languages software science introduces the notion of value, denoted by V, which represents the number of bits required to encode the program in an alphabet with one character per operand or operator. More specifically, $\log_2 \eta$ is the minimum length in bits of all individual elements in a program and the volume of a program is defined as:

$$V = N \log_2 \eta \qquad (5)$$

But the volume of a program still changes when being translated from a language to another. Thus, the notion of potential volume, also called minimal volume, is induced to solve the problem.

$$V^* = (N_1^* + N_2^*)\log_2(\eta_1^* + \eta_2^*) \qquad (6)$$

where '*' means minimal.

Since it is in the minimal form, neither operator nor operand could require repetition, thus

$$N_1^* = \eta_1^* \qquad (7)$$

and

$$N_2^* = \eta_2^* \qquad (8)$$

giving

$$V^* = (\eta_1^* + \eta_2^*)\log_2(\eta_1^* + \eta_2^*) \qquad (9)$$

Furthermore, the minimum possible number of operators η_1^* for any algorithm must consist of one distinct name operator for the name of the function or procedure and another to serve as an assignment or grouping symbol, therefore,

$$\eta_1^* = 2 \qquad (10)$$

Then equation (6) becomes

$$V^* = (2 + \eta_2^*)\log_2(2 + \eta_2^*) \qquad (11)$$

$$V^*(\mathrm{L1}) \approx V^*(\mathrm{L2}) \qquad (11.1)$$

where η_2^* for small algorithms should represent the number of different I/O parameters.

Halstead claimed that the potential volume V^* would not change no matter how it is translated from one to another language.

Furthermore, a third volume – boundary volume, V^{**} is introduced as

$$V^{**} = (2 + \eta_2^* \log_2 \eta_2^*)\log_2(2 + \eta_2^*) \qquad (12)$$

But an interesting phenomenon we have discovered is that the potential volume V^* changes while an algorithm being translated from a low level language to an OOL. The discussion follows in the Section 3.

2.3 Language level and programming effort

Halstead had proposed a way to estimate the programming effort E

$$E = \frac{V}{L} \qquad (13)$$

where L is the level of language with the definition

$$L = \frac{V^*}{V} \qquad (14)$$

Hence

$$\mathrm{E} = \frac{V^2}{V^*} \qquad (15)$$

From Halstead' point of view, 'a potential language (L=1) would be the easiest to use'. This is clearly valid assumption as everyone would agree that it is easier to call a method than it is to implement the method itself. And consequently, to program with a higher-level language will require less effort.

It is true that for higher-level languages such as OOLs, the volume of program will always be smaller than that of their low level counterparts due to the high reuse ratio of words. In the next section we present a formal proof that the level L of OOL programs is higher than that of programs written in POLs. Hence the effort required for programming in OOL will be smaller.

3 *Halstead's Software Science for OOL*

What we present in this section is not a thorough study but rather a preliminary examination of the applicability of the software science to the object oriented software development. uorally Given the lack of time and previous experience, we provide just basic counting rules that could be used with OOL.

3.1 Syntax-Oriented counting rules and software size

Even though in the original work of Halstead not too much attention was paid to the problem of counting, it seems to be an important issue since all of the software science measures are build upon results of counting the operators and operands. Another reason to further investigate this issue is the fact that as we switch from POL to OOL we encounter many new constructs that were, naturally, unaccounted for in the Halstead's original work.

For our purpose of the initial investigation we have created a list of simple syntax-oriented counting rules. Our (limited) experiments have shown that various modifications to this list did not affect the final results. If counting rules used consistently the application of these rules in the counting of vocabulary, η_1 and η_2 of a piece of OO program code varies with the change of rules yet, all of the other measures remain proportionally consistent. More investigation including empirical validation is needed for the rules to be truly usable in practice.

Syntax-oriented counting rules used include:

- If two symbols always occur together, count them as one operator.
- If there are two different structures that are semantically the same, still count them as two different operators.
- Count declarations, I/O, etc.
- Count everything that is necessary for expressing the program.
- Operators are basically keywords.
- User-defined items are basically operands.

3.2 Relation between operators, operands, and volume

First of all, it has to be noted that implementation of an algorithm using an OOL will most likely give out a counting of a bigger η_1 and a smaller η_2 than that of the same algorithm implemented with POL. If we do not take any advantage of OOL, which means we code with the same range of operators as a POL, we can definitely hammer out the algorithm that we could expect if using POL. Thus,

$$\eta_1 \text{ (OOL)} \geq \eta_1 \text{ (POL)} \qquad (16)$$

holds. And since a greater operator set is created to enable a more efficient way of variable denotation and to offer a more brief and effective way of syntax, η_2 tends to decrease in OOL.

$$\eta_2 \text{ (OOL)} \leq \eta_2 \text{ (POL)} \qquad (17)$$

Hence,

$$\eta_2^* \text{ (OOL)} \leq \eta_2^* \text{ (POL)} \qquad (17.1)$$

because $\eta_2^* = \log_2 \eta_2$.

And consequently we get

$$\eta \text{ (OOL)} \leq \eta \text{ (POL)} \qquad (18)$$

Based on these observations, we can now examine Halstead's assertions about volume.

Formula (5) $V = N \log_2 \eta$ holds, which means that more powerful languages produce smaller program volumes. However, Halstead's claim that potential value V^* is language invariant (remains unchanged with the change of the programming language) does not hold any more when we move from POL to OOL. It is easy to show that

$$V^*(\mathrm{OOL}) \le V^*(\mathrm{POL}) \qquad (19)$$

and not $V^*(\mathrm{L1}) \approx V^*(\mathrm{L2})$ as we learned in (11.1).

We can prove inequality (19) as follows:

$$\frac{\mathrm{V}^*(\mathrm{OOL})}{\mathrm{V}^*(\mathrm{POL})} = \frac{(2+\eta_2^*(OOL))\log_2(2+\eta_2^*(OOL))}{(2+\eta_2^*(POL))\log_2(2+\eta_2^*(POL))} \le 1$$

As most of Halstead's later work is related to *V**, such as the estimation of level L and effort E, most of Halstead's measurements should be adjusted in the OO world.
For example compare the two implementation of the same simple program in C++ (left) and C(right):

`//class.h`
`class Object`
`{`
` public:`
`  int attr0;`
`  bool attr1;`
`  float attr2;`
`  char attr3;`
 	
`  Object()`
`  {`
`  attr0 = 1;`
`  attr1 = True;`
`  attr2 = 1.5;`
`  attr3 = 'A';`
`  }`
`}`
 	
`...`
 	
`//mainproc.cpp`
`#include "class.h"`
`int main()`
`{`
`  Object A,B,C,D;`
`}`
	`//mainproc.c`
	`main()`
	`{`
	`  int i;`
	`  int attr0[4];`
	`  bool attr1[4];`
	`  float attr2[4];`
	`  char attr3[4];`
	
	`  for(i = 0; i < 4; i++)`
	`  {`
	`  attr0[i] = 1;`
	`  attr1[i] = True;`
	`  attr2[i] = 1.5;`
	`  attr3[i] = 'A';`
	`  }`
	`}`

$\eta_2^* = 4$ $V^* = (2 + \eta_2^*) \log_2(2 + \eta_2^*)$ $= 6\log_2 6 \approx 16.5$ bits	$\eta_2^* = 11$ $V^* = (2 + \eta_2^*) \log_2(2 + \eta_2^*)$ $= 13\log_2 13 \approx 48.1$ bits

This phenomenon is due to the fact that in the OOL code the default constructor method is called 4 times, while in the POP the code has to be written explicitly to achieve the same action hence making the vocabulary of operands bigger. Even though this example is rather oversimplified, it illustrates the main point well.

3.3 Language level vs. effort

Without considering the deviation of potential volume V^* in OOL, the relationship between program, program language level, and programming effort remain constant. When we use more powerful language to program, usually we pay less effort and get shorter programs. In fact, the main question here is about the exact value of L and E due to V^* which has to be reconsidered taking into consideration the above discussion.

We intend to further investigate the behavior of value and other reliant measures in the context of OOL but it can be seen from the above discussion that it is not impossible to adjust the basic concepts of the software science to be still useful in the OO paradigm.

4 Conclusions

In this paper we have briefly introduced a part of the Halstead's theory on software science. Halstead's measurement is directly quantified on the source code. It only measures internal attributes and then deduces the estimated size, workload, effort, and time on the given program. From the OO view, we have examined Halstead's statements and assertions and have found out that while some of his methods became outdated due to the differences between POLs and OOLs, there are ways to make many of his measures to be useful today. We have discussed rules of counting that are applicable to programs written on an OOL, size, volume, language level and effort software science measures.

This is paper is just a brief overview intended to show that Halstead's software science could potentially be adapted to today's object oriented software development. In order to make it possible first we have to carry out a systematic study supported by strong experimental results and second the proposed measurements have to be validated based on the measurement theory. One of the suitable methods for such validation is the method proposed by Alain Abran [1] for validation of software measurements. The method is based on the following criteria:(1) representation condition; (2) attribute validity; (3) numerical assignment rules; and (4) meta-model, that is precise representation of the objects being measured.

Numerous critiques of Halstead's work refer to the estimation techniques that he provides. Since Halstead only measures internal attributes and then deduces the estimated size, work load, effort, and time on the given program, it would be almost "too good to be true" if his prediction system was proven to be reliable. Unfortunately, this is truly the weakest part of software science and that calls for the most effort to investigate.

References

[1] Abran, A.: *Metrics Validation Proposals: A Structured Analysis*, in 8th International Workshop on Software Measurement, Magdeburg, Germany, 1998, pp. 26

[2] Cartwright M., and Shepperd, M.: *An empirical investigation of an object oriented Software System*, IEEE Transactions on software engineering, volume. 26, No. 8, August 2000

[3] Chidamber, S.R., and Kimerer, C.F.: *A metrics suite for object oriented design*, IEEE Transactions on Software Engineering, 20(6), pp. 476-98, 1994.

[4] Fenton, N., and Pfleeger, S. L.: *Software Metrics: A Rigorous & Practical Approach*, PWS Publishing, 2nd edition, revised printing, 1998, ISBN 0-534-95425-1.

[5] Fenton, N., and R. W. Whitty: *Axiomatic Approach to Software Metrication Through Program Decomposition*. The Computer Journal 29 (4): 330-339 (1986)

[6] Halstead, M. H.: *Elements of Software Science*, Elsevier North-Holland, 1977, ISBN 0-444-00205-7

[7] Peter G. Hamer and Gillian D. Frewin: *M.H. Halstead's Software Science – a Critical Examination*, IEEE Transactions on Software Engineering, 20(6), pp. 197-206, 1982.

[8] Jones, C.: *Computer*, Volume: 27, Issue: 9, pp. 98 – 100, Sept. 1994

[9] Li, W., and Henry, S.: *Object oriented metrics that predict maintainability*, Journal of Systems and Software, 23, pp.111-22, 1993

[10] McCabe, T.: *A software complexity measure*, IEEE Transactions on Software Engineering SE-2(4), pp. 308-2-, 1976

[11] Pfleeger, S.L., and Palmer, J.D.: *Software estimation for object-oriented systems*, Journal of Systems and Software, 12(3), pp. 255-61, 1990

Metriken für zustandsbasierte Software-Entwicklung

Martin Mutz

TU Braunschweig, Institut für Programmierung und Reaktive Systeme
Mühlenpfordtstr. 23, 38106 Braunschweig
mmutz@tu-bs.de

Abstract. *Im Folgenden wird ein prototypisches Werkzeug zur Qualitätssicherung im modellbasierten Entwurf vorgestellt. Mit diesem Programm kann eine automatische Überprüfung von Modellierungsregeln an zustandsbasierten Modellen erfolgen. Des Weiteren ist eine Bewertung des Gesamtsystems durch Software-Metriken möglich. Der Prototyp namens Regel Checker wurde im Rahmen des Automotive-Projekts STEP-X[1] entwickelt und eingesetzt. Mit Hilfe des Regel Checkers können*

[1] Vgl. www.step-x.de

Modellierungsregeln und Software-Metriken in OCL bzw. Java definiert werden, um Inkonsistenzen, Inkompatibilitäten sowie Designfehler zu analysieren und zu bewerten. In diesem Beitrag wird insbesondere auf die Definition und Ausführung der Software-Metriken eingegangen.
In the following, a prototypic tool is introduced for quality assurance in model-based design. Automatic checking of modelling rules in state-based models can be done with this application. Furthermore, an evaluation of the total system is possible by means of software metrics. The prototype called Rule Checker has been developed and used in the automotive project STEP-X. With the Rule Checker, modelling rules and software metrics can be defined in OCL and/or Java in order to analyse and evaluate inconsistencies, incompatibilities as well as design errors. This paper deals in particular with the definition and execution of software metrics.

Keywords*: automotive, software metrics, rule checker, statecharts, UML*

1. Einleitung

Die Modellierung von Software-Funktionen für *Reaktive Systeme* im Embedded-Bereich wird derzeit zum größten Teil durch diskrete, zustandsbasierte Beschreibungssprachen wie Statecharts oder UML-Zustandsdiagramme realisiert. Diese Beschreibungssprachen werden durch zahlreiche Werkzeuge unterstützt. Mit diesen Entwicklungswerkzeugen kann ein SW-System modelliert und simuliert werden. Anschließend lässt sich Code aus solchen ausführbaren Modellen für unterschiedliche Zielplattformen automatisch generieren. Schwächen der kommerziellen Modellierungswerkzeuge liegen z.B. in der unvollständigen Konsistenzüberprüfung, der Inkompatibilität zu anderen Werkzeugen und der fehlenden Bewertung der SW-Qualität durch Metriken. Dadurch ist bspw. die Durchgängigkeit eines Entwicklungsprozesses nicht überprüfbar (fehlende Kompatibilität-Checks) und die Einhaltung von Modellierungsrichtlinien (zumeist in externen Dokumenten vorhanden) kann nicht automatisch überprüft werden. Eine quantitative Bewertung des (Teil-)Systems ist mangels proprietärer Schnittstellen an den Werkzeugen nur schwer möglich.
Vor diesem Hintergrund wird in diesem Papier das Werkzeug *Regel Checker* [Mut03], welches im Rahmen des STEP-X Projektes [MHG+03] entwickelt wurde, vorgestellt. Mit dem Programm können zustandsbasierte Modelle (z.B. Statecharts, Statemachines) aus unterschiedlichen Werkzeugen eingelesen und anhand von definierten Modellierungs-regeln und Software-Metriken automatisch analysiert sowie qualitativ und quantitativ bewertet werden.

Das Papier ist folgendermaßen gegliedert. Nach der Einleitung wird im zweiten Kapitel ein kurzer Überblick zu Statecharts gegeben. Im dritten Kapitel werden einige Software-Metriken vorgestellt und in Bereiche klassifiziert. Kapitel 4 gibt einen Überblick über den Überprüfungsmechanismus des Regel Checkers. Das letzte Kapitel schließt mit einer Zusammenfassung ab und gibt einen Einblick in die zukünftigen Arbeiten auf diesem Gebiet.

2. Überblick über Statecharts

Statecharts wurden Mitte der 80er Jahre von Harel [Hare87] als ein visueller Formalismus zur Beschreibung reaktiver Systeme vorgeschlagen und stellen eine

Weiterentwicklung der allgemeinbekannten endlichen Automaten (Zustandsdiagramme) dar. Statecharts ermöglichen unter anderem das Schachteln von Zuständen, Zusammenfassen von komplexen Transitionen und die Darstellung von nebenläufigen Abläufen mit Hilfe der Broadcast-Kommunikation. Der strukturelle Aufbau eines Statecharts besteht aus hierarchisch aufgebauten Zuständen und Transitionen.

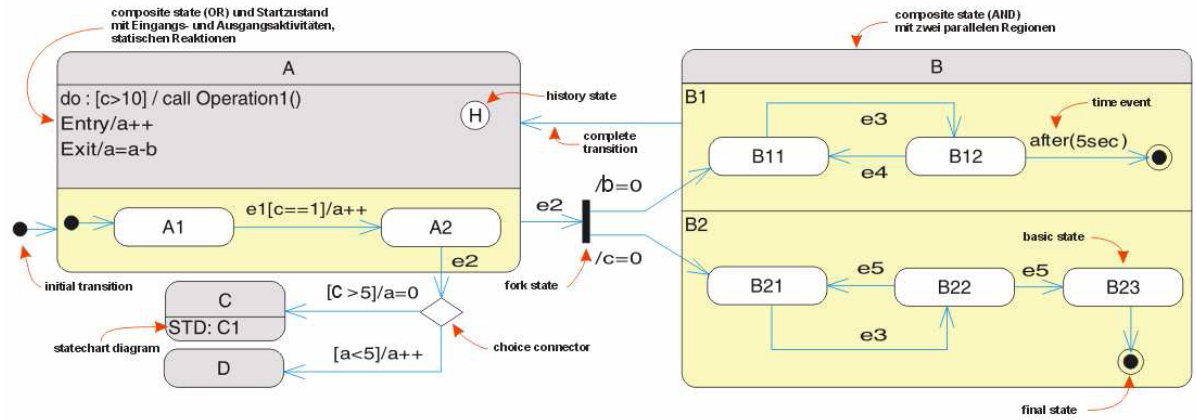


Abbildung 1: Ein Zustandsdiagramm der UML

Zustände

Jedes Zustandsdiagramm enthält eine endliche Menge von Zuständen und Pseudozuständen, einen Startzustand (initial state) und eine Menge von Endzuständen (final states). Das Eintreten in den Startzustand beschreibt den Zeitpunkt der Instanziierung eines Objekts, wohingegen das Betreten aller Endzustände in einem Statechart die Zerstörung des Objekts bedeutet. Zustände können dahingehend verfeinert werden, dass sie Unterzustände enthalten. Dies ist bis zu einer beliebigen Verschachtelungstiefe möglich. Zustände, die Unterzustände enthalten, werden zusammengesetzte Zustände (composite states) genannt. Basiszustände (basic states) dahingegen besitzen keine Unterzustände. Ein Zustand kann entweder in nebenläufige Regionen (AND-Zustände) oder in sich gegenseitig ausschließende Unterzustände (OR-Zustände) aufgegliedert werden.

Im Falle eines OR-Zustands kann sich das System zu einem Zeitpunkt nur in einem Zustand pro Ebene befinden. Wird in der Abbildung 1 der Zustand *A* betreten, so ist nur einer der beiden Unterzustände *A1* oder *A2* aktiv. Zur Kennzeichnung eines Startzustands wird dieser mit einer Initial-Transition gekennzeichnet. Beim Verlassen des Oberzustands werden alle seine Unterzustände mitverlassen.

Zustände können komplette Zustandsautomaten mit einem Startzustand und einem oder mehreren Endzuständen enthalten. Dies wird im Zustand *C* durch das Akronym *STD* (statechart diagram) gekennzeichnet. Als grafisches Hilfsmittel wurden Pseudozustände eingeführt, mit denen komplexere Modellkonstrukte einfacher dargestellt werden können. Ein solcher Pseudozustand ist bspw. der History-Konnektor (history state). Mit ihm werden beim Wiedereintritt in einen zusammengesetzten Zustand alle diejenigen Unterzustände betreten, die vor dem Verlassen aktiviert waren.

Transitionen

Eine Transition verbindet genau zwei Zustände bzw. Pseudozustände miteinander und ist für einen Zustandswechsel verantwortlich. Sie kann mit einem Ereignis *e*, einer Bedingung c und/oder einer Liste von Aktionen a in der Form $e[c]/a$ beschriftet werden. Ein Zustandsübergang kann nur dann stattfinden, wenn das Ereignis triggert und die Bedingung an der Transition wahr ist. Mit dem Schalten werden alle Aktionen der Transition ausgeführt, wodurch beispielsweise Variablen geändert und neue Ereignisse ausgelöst werden können. Anschließend wird der Zielzustand betreten. Im Allgemeinen kann der Quell-und Zielzustand auf unterschiedlichen Ebenen eines Statecharts liegen (Interlevel-Transition).

Die meisten Statechart-Varianten unterstützen drei Arten von Transitionen. Einfache Transitionen (simple transitions) führen direkt von einem Zustand in den anderen. Zusammengesetzte Transitionen (compound transitions) dagegen beinhalten zwei oder mehrere einfache Transitionen, die durch Pseudozustande verbunden sind. Beispielsweise führt eine zusammengesetzte Transition aus dem Zustand *A2 ü*ber den choice-Konnektor in die beiden Zustande *C* und *D*. Im Gegensatz zu „normalen“ Zustanden, darf in Pseudozuständen nicht verweilt werden. Ein Schalten ist daher nur möglich, wenn alle einfachen Transitionen einer zusammengesetzten Transition aktiviert sind. Bei der dritten Art der Transition handelt es sich um die so genannten abschließenden Transitionen (completion transitions), die mit keinem expliziten Trigger-Ereignis beschriftet sind. Sie werden erst dann getriggert, wenn ein oder mehrere Endzustande (final states) betreten werden.

Des Weiteren können zur besseren Übersicht Aktionen an einen Zustand gebunden werden, wie bereits von Moore-Automaten bekannt. Die Zustände können so über drei interne Aktionsarten verfügen. Die Eingangsaktion (entry action) wird erst dann ausgeführt, wenn der Zustand betreten wird. Beim Verlassen eines Zustands wird die Ausgangsaktion (exit action) ausgeführt. Mit Hilfe der statischen Reaktionen (static reactions) wird eine Aktivität solange ausgeführt, wie in diesem Zustand verweilt wird.

Transitionsverläufe können durch Verzweigungspunkte beeinflusst werden. Diese Verzweigungspunkte werden durch Auswahlkonnektoren (choice connector) dargestellt, in den eine Transition hineinführt und mehrere, mit booleschen Bedingungen versehene, Transitionen herausführen. Beispielsweise kann der Zustandsübergang vom Zustand *A2* je nach Bedingung entweder in den Zustand *C* oder *D* erfolgen.

Wenn nach dem Auslösen eines Ereignisses aus derselben Konfiguration unterschiedliche Folgekonfigurationen möglich sind, so spricht man hier von einem Konflikt.

Eine Konfliktsituation liegt dann vor, wenn mehrere aktive Transitionen aus einem zusammengesetzten Zustand auf unterschiedliche Ebenen führen. In Abbildung 1 wird dies durch den Zustand *A* und das Ereignis *e2* dargestellt. Für diesen Fall wird in der UML eine Prioritätsreihenfolge definiert, die dafür sorgt, dass Konflikte, die zwischen gleichzeitig aktivierten Transitionen auftreten, behoben werden können. Nach [OMG02] hat diejenige Transition eine höhere Priorität, die von einem tiefer liegenden Zustand ausgeht.

Eine Konfliktsituation entsteht auch, wenn mehrere aktive Transitionen aus einem Zustand führen. Der Zustand *B22* enthält zwei Transitionen mit demselben Ereignis *e5*. In diesem Fall haben beide Aktionen dieselbe Priorität. Die UML gibt für diesen Fall keine Lösung an. Jedes Werkzeug löst einen solchen Konfliktfall auf eigene Art und Weise. Entweder durch interne Prioritätenvergabe oder durch Warnmeldungen beim Simulationslauf.

Nebenläufigkeit

Das Konzept der Nebenläufigkeit wird durch orthogonale Regionen (AND-Zustande) repräsentiert. In der Abbildung 1 besitzt der AND-Zustand *B* die beiden Regionen *B1* und *B2*. Beim Aktivieren eines AND-Zustands wird keine Aussage bezüglich der Eintrittsreihenfolge in den Regionen gemacht. Wird ein solcher AND-Zustand betreten, so werden alle seine nebenläufigen Startzustände aktiviert. Soll dahingegen von einem anderen Zustand in einer Region gestartet werden, so ist ein Synchronisationsbalken zu verwenden. Je nachdem, ob ein Synchronisationsbalken mehrere Nachzustände oder mehrere Vorzustände besitzt, wird er als Fork-Konnektor oder als Join-Konnektor bezeichnet. Für Fork-Konnektoren gilt, dass ihre Nachzustände unterschiedlichen Subzuständen eines AND-Zustands angehören müssen und ihr Vorzustand außerhalb dieses AND-Zustands liegen muss. Bei Join-Konnektoren müssen die Vorzustande unterschiedlichen Subzuständen eines AND-Zustands angehören, während sich ihr Nachzustand außerhalb des AND-Zustands befinden muss.

Beispielsweise wird der OR-Zustand *A* mittels des Ereignisses *e2* verlassen und mit Hilfe des Fork-Konnektors in die Basiszuständen *B11* und *B21* des AND-Zustands *B* gewechselt, wobei die beiden Variablen *b* und *c* jeweils auf *Null* gesetzt werden.

Transitionen in Regionen schalten grundsätzlich unabhängig voneinander. Wird bspw. aus der momentanen Basiskonfiguration das Ereignis *e3* ausgelöst, so wechselt das System in die Folgezustände *B12* und *B22*.

3. Klassifikation von Metriken

Mit Hilfe des Regel Checkers können unterschiedliche Arten von Regeln und Metriken in Form von Java und OCL (Object Constraint Language) [OMG02] definiert werden. Im Gegensatz zu Modellierungsregeln, mit denen bestimmte Modellierungskriterien (Konsistenzsicherung, Werkzeugkompatibilität, UML-Konformität, Einhaltung von Designregeln usw.) definiert und überprüft werden können, werden Metriken zur Modellanalyse und Modellbewertung herangezogen. Da Software-Metriken je nach Verwendung unterschiedlich starke Ausprägungen haben, werden sie mit einer entsprechenden Gewichtung von 1 (unrelevant für die Gesamt-bewertung) bis 10 (sehr wichtig für die Gesamtbewertung) versehen. Die Gewichtungsangaben stehen in Klammern hinter der Metrik bzw. der Metriken-kategorie. Folgende Kategorisierung wurde für unsere Metriken vorgenommen:

- **Zustandsmetriken (10)**: Bewerten das Modell nach strukturellen Gesichtspunkten. Dazu gehören unter anderem die Anzahl und Verteilung der Zustände pro Statechart, die Lange der Zustandsbeschriftungen und die Hierarchietiefe. Das gleiche gilt auch für die Bestimmung von internen Aktionen (entry, exit) in einem Zustand.

- **Pseudozustandsmetriken (9)**: Die Bewertung des Modells durch Pseudozustandsmetriken gleicht der Bewertung durch Zustandsmetriken. Das Bewertungskriterium ist jedoch auf die Anzahl der Pseudozustände und deren Typ reduziert.
- **Transitionsmetriken (9)**: Bewerten die Komplexität eines Statecharts, indem die Transitionsübergänge zwischen den Zuständen analysiert werden. Dabei wird vorwiegend zwischen ein- und ausgehenden Transitionen unterschieden. Zusätzlich werden Transitionsverläufe (z.B. interlevel-oder self-loop Transitionen) bestimmt. Des Weiteren findet eine Unterteilung der Transitionen statt, wobei die Transitionsbeschriftung ein wesentliches Bewertungskriterium darstellt.
- **Layoutmetriken (8)**: Bewerten die Übersichtlichkeit und das Verständnis des zustandsbasierten Modells, indem Informationen zu Positionierungen, Skalierungen und der symmetrischen Anordnung von Modellelementen analysiert werden.
- **Klassenmetriken (4)**: Bei dieser Bewertung stehen Verhaltensklassen im Vordergrund. Dabei wird die Anzahl der Attribute und Methoden jeder Klassen analysiert und das Minimum, Maximum sowie der Durchschnitt ermittelt. Außerdem werden die Länge, der Typ sowie die Parameter der Attribute bzw. Methoden analysiert.

Bislang wurden über 100 Regeln und 40 Metriken zusammengestellt, die zum Teil einen standardisierten Charakter in der Automobilindustrie haben [Math01]. Die meisten Regeln und Metriken wurden in den Regel Checker in Java und OCL implementiert. Zurzeit werden die implementierten Metriken mit Parametern versehen, um einen flexibleren Bewertungsmechanismus zu erhalten.

4. Der Analyse-und Bewertungsprozess

Das Auffinden von Inkonsistenzen und weiteren Regelverletzungen ist der erste Schritt bei der Überprufung eines Statecharts. Um die identifizierten Beanstandungen zu einem späteren Zeitpunkt aussagekräftig darstellen zu können, wurde ein Nachrichtensystem entwickelt, das die spätere Aufbereitung des Überprüfungsergebnisses ermöglicht. Dieses ist in der Abbildung 2 zu sehen. Auf der linken Seite sind die aufgetretenen Beanstandungen nach Schweregraden (Fehler, Warnung, Vorschlage) aufgelistet. Die rechte Seite zeigt das Statechart als Baumstruktur, wobei die momentan selektierte Fehlernachricht in Form von Pfeilen dargestellt wird. Im unteren Fensterausschnitt werden die Programm- und Benutzeraktivitäten protokolliert. Wird das Statechart nach der Regelüberprüfung so modifiziert, dass keine Beanstandungen vom Regel Checker erzeugt werden, ist das Modell als korrekt einzustufen.

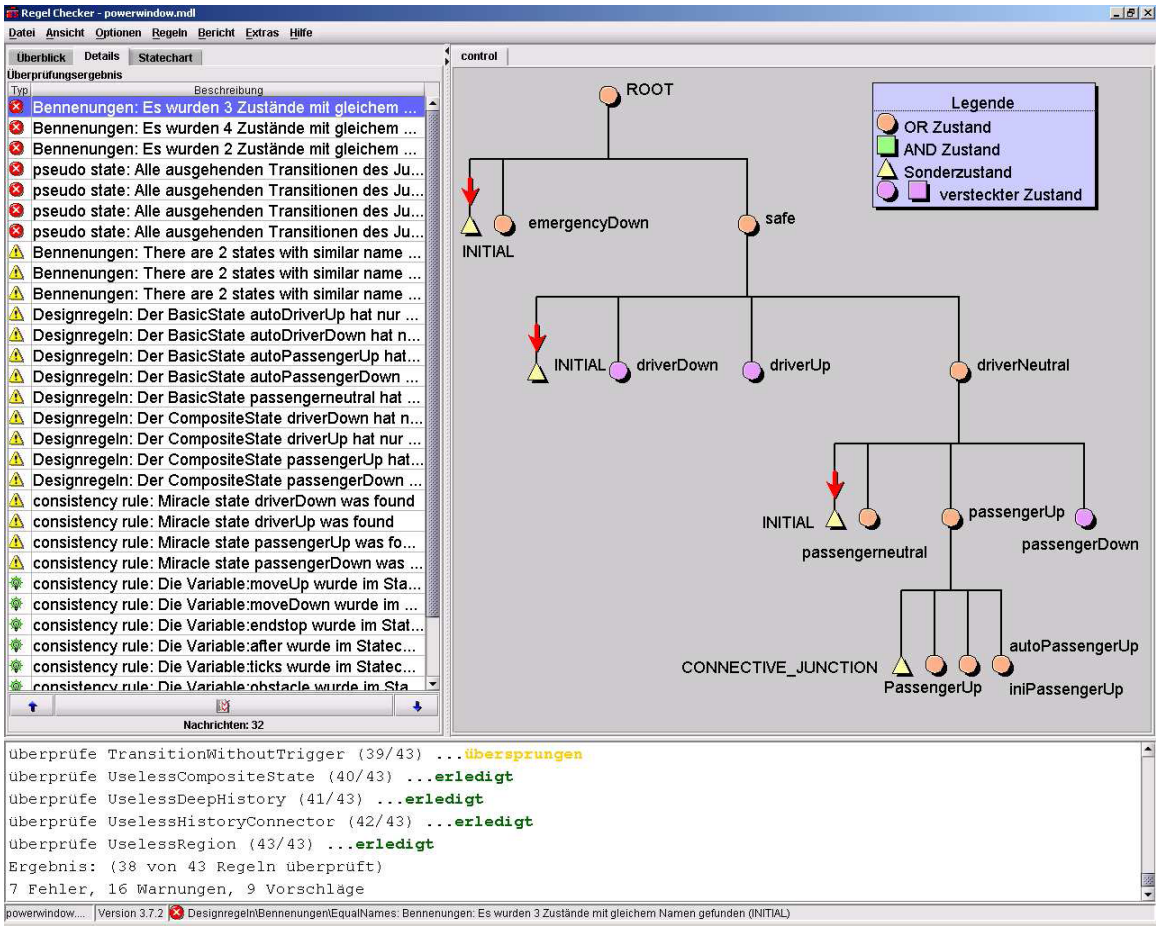


Abbildung 2: Ausgabe der Beanstandungen

Im zweiten Schritt erfolgt die Analyse des Modells durch Metriken. Bei der Metrikenwahl kann der Detaillierungsgrad je nach Abstraktionstiefe gewählt werden. Nach der Metrikenüberprüfung wird das Ergebnis entweder in textueller Form oder als Balkendiagramm (vgl. Abbildung 3) dargestellt.

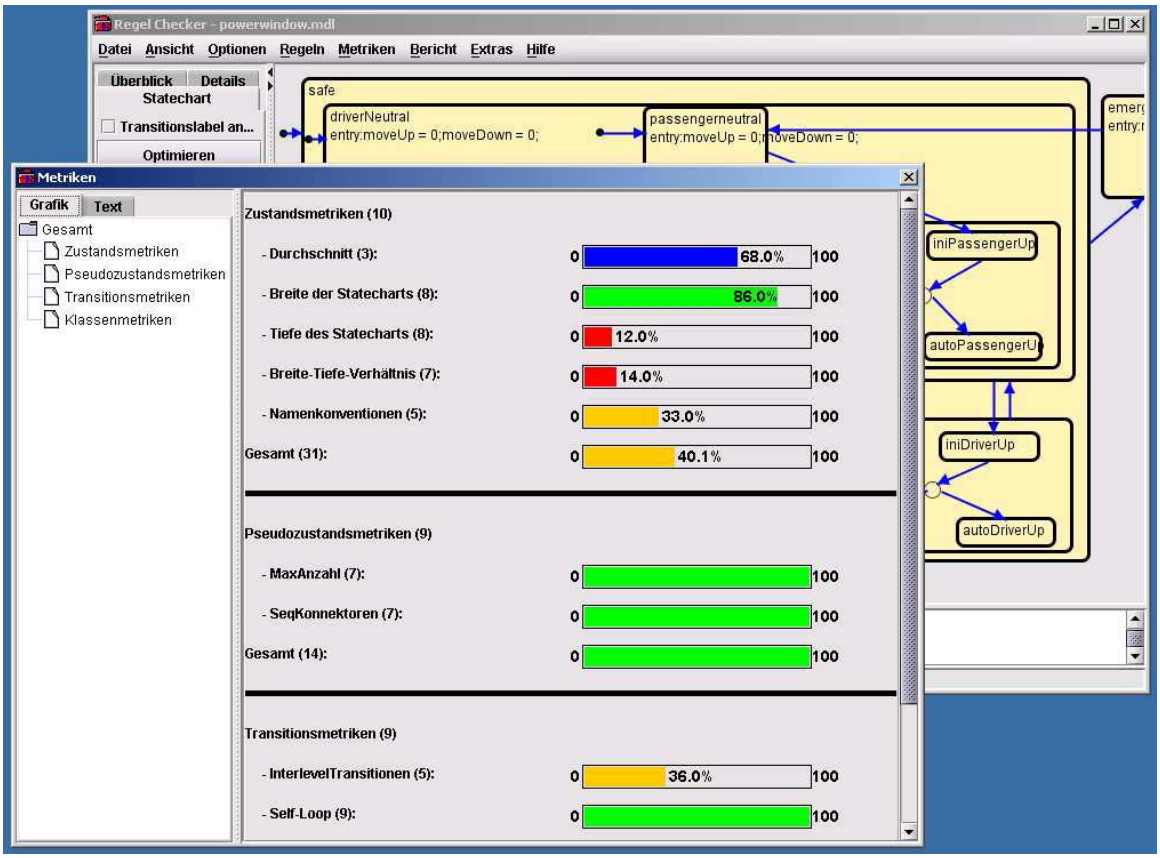


Abbildung 3: Ausgabe der Ergebnisse durch Metriken

Am Beispiel der Kategorie *Zustandsmetriken* aus Kapitel 3 soll der letzte Schritt, die Berechnung des Gesamtmodells (B_{Modell} in Prozent), erläutert werden. Eine Kategorie besteht grundsätzlich aus mehreren Metriken (M = Anzahl der Metriken pro Kategorie). Dies wird grafisch in der Abbildung 3 dargestellt. Für dieses Beispiel wird zunächst die Anzahl aller Zustände ermittelt, um den Umfang des Projekts einzuschätzen. Dabei werden State-chart-Modelle bis 500 Zustände als klein, bis 2000 als umfangreich und darüber hinaus als groß definiert. Des Weiteren wird das Minimum, Maximum sowie der Mittelwert der Zustände pro Statechart berechnet, um die ungefähre Verteilung der Funktionskomplexität zu ermitteln. Eine weitere Metrik analysiert die hierarchische Struktur der Statecharts. Dabei wird die Tiefe und Breite aller Statecharts festgestellt und ins Verhältnis gesetzt. Dadurch können unterschiedliche Aussagen getroffen werden. Beispielsweise sind tiefe (viele Ebenen) und schmale (wenig Zustände pro Ebene) Statecharts genauso unübersichtlich, wie flache aber sehr breite Statecharts. Als letztes wird die Korrektheit der Zustandsbeschriftungen überprüft. Zur Berechnung der Gesamt-bewertung wird die nachfolgende Formel verwendet.

$$B_{Modell} = \frac{1}{G_{K_{Sum}}} \sum_{j=1}^{K} \left(\frac{1}{G_{M_{Sum}}} \sum_{i=1}^{M} B_{M_i} G_{M_i} \right) \cdot G_{K_j}$$

Dabei wird jede Bewertung einer Metrik (BMi in Prozent) mit deren Gewichtung (G_{Mi}/G_{MSum}) multipliziert und mit den anderen Metrikenbewertungen aufsummiert. Anschließend erhält man die Bewertung für diese Kategorie. Zur Gesamtbewertung

werden alle Kategorien auf dieselbe Weise berechnet und jeweils mit der dazugehörigen Kategoriengewichtung G_{Kj}/G_{KSum} versehen. Da die Gewichtungs-summe aller Kategorien G_{KSum} und die Gewichtungssumme G_{MSum} aller Metriken als konstant betrachtet werden kann, werden diese aus den jeweiligen Summen herausgezogen.

5. Zusammenfassung

Die hier verwendeten Regeln und Metriken leisten einen großen Beitrag zur Verbesserung des zustandsbasierten Entwicklungsprozesses, indem Werkzeugkompatibilitäten, Konsistenzsicherungen und weitere Qualitätssicherungsmaßnahmen eingehalten werden. Des Weiteren können Qualitätsmerkmale des Modells durch Software-Metriken analysiert und bewertet werden. Die Ergebnisrepräsentation der beiden Verfahren ist sowohl in textueller als auch in grafischer Form möglich. Im STEP-X Projekt wurde das Verfahren anhand einer Fallstudie aus dem Kfz-Komfortbereich erfolgreich erprobt. Die Analyseergebnisse trugen zur generellen Verbesserung des Modellierungsstils bei. Dadurch konnte die Lesbarkeit der Modelle erhöht werden, was zu besserem Verständnis und schnellerer Einarbeitung in das umfangreiche Modell führte. Außerdem konnte die manuelle Umsetzung der UML-Statecharts (Grobentwurf) in Stateflow bzw. Ascet-SD (Feinentwurf) erheblich vereinfacht werden.

Literaturverzeichnis

[Hare87] D. Harel. Statecharts: A *Visual Formalism for Complex Systems*, Science of Computer Programming, 1987

[MHG+03] M. Mutz, M. Huhn, U. Goltz, and C. Kromke. *Model based system development in automotive*. In: SAE World Congress 2003. Society of Automotive Engineers, 2003

[Math01] MathWorks. *Controller Style Guidelines For Production Intent Using MATLAB*, *Simulink and Stateflow*, MathWorks Automotive Advisory Board Version 1.0, 2001

[Mut03] M. Mutz. *Ein Regel Checker zur Verbesserung des modellbasierten Softwareentwurfs*, Toolpräsentation, EKA 2003, Entwurf komplexer Automatisierungssysteme, Braunschweig, Juni 2003, S. 411-423

[OMG02] Object Management Group. *UML 2.0 Superstructure Specification*, Version 2.0, 2002

Development of Quality Proved Object Based Distributed Applications in Control Area

Matthias Riedl
Institut für Automation und Kommunikation e.V. Magdeburg
Steinfeldstr. 3
39179 Barleben
matthias.riedl@ifak-md.de
http://www.ifak-md.de/

Abstract. *Traditional design methods in the control area are dominated by the Function Block approach. Such a block is defined by the International Electrotechnical Commission as the basic construct for the development of reusable, distributed and interoperable applications. A block represents a kind of object with an internal behaviour. The blocks interact via input and output variables with other blocks or with their environment. In order to improve the engineering, the quality of the system and the system design, other approaches like method invocations or statements about the component quality, coming from computer science, have to be considered. One promising approach among others is the usage of ports for the interaction of objects and the usage of meta information about quality representative attributes. This article describes an approach for a distributed, object-oriented automation system, based on objects interacting via the port concept which can be designed by a high level language DOME-L, offering the additional features for defining the automation objects. Quality declaring values can be determined by the compiler and can be put into the meta information of the object. Inside of DOME-L, the target object-oriented programming language for the controller (e.g. C++, Java) will be used.*

Keywords: *Component Based Control Systems, Distributed Control Systems, Distributed Object Model Environment (DOME), Embedded Systems, Event Driven Architecture, Object Based Distributed Applications, Software Quality Prediction*

Introduction

In the world of automation systems the Function Block approach is quite common. International standards like IEC 61131 [1], [2], IEC 61499 [3] or IEC 61804 [4] support this concept. In this concept, a Function Block (FB) instance is a special kind of object (class instance) with internal variables (attributes), describing e.g. the recent state of the concerning instance. The interaction between the FBs is always done by data connections. The data connections are represented by links between input and output variables (see
Fig. 1).

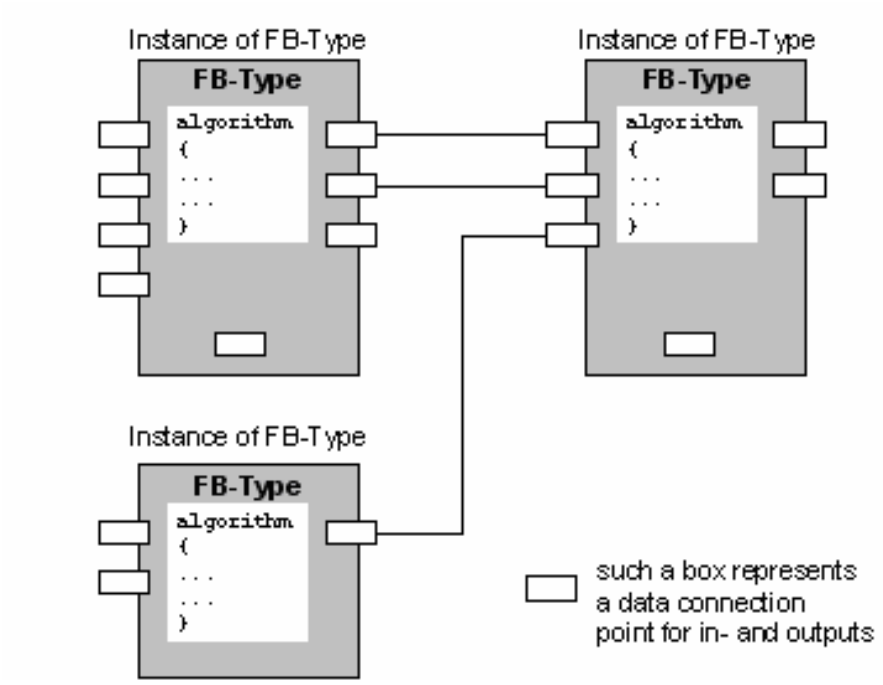


Fig. 1: **Data Flow in IEC 61131 FB applications**

Complete control applications can be designed as a network of such FBs. In IEC 61131, the FBs have only one (public) method. This method will be called by the environment implicitly periodically. The order of the FB invocations is defined during the design of the application by the user.

In addition to IEC 61131, the IEC 61499 offers the possibility to define special event connections in order to provide the event driven approach (explicit design of FB invocation order by event flow, see Fig. 2). Furthermore, the consistency between incoming events and the update of related data inputs can be defined. Depending on the event and the internal state, different internal methods of the FB can be executed. After the method execution, several output data can be set and output events can be sent out. The open issue is, that the standard does not define something about the FB scheduler. The FB scheduler is responsible for the activation of the FBs in an application. This is completely vendor or system specific. Out of scope is the interoperability of such systems, this has to be standardised in future too. Up to now, the Function Block Development Kit (FBDK) by Rockwell Automation [5] and CORFU-FBDK [6] are the only known tools supporting this approach.

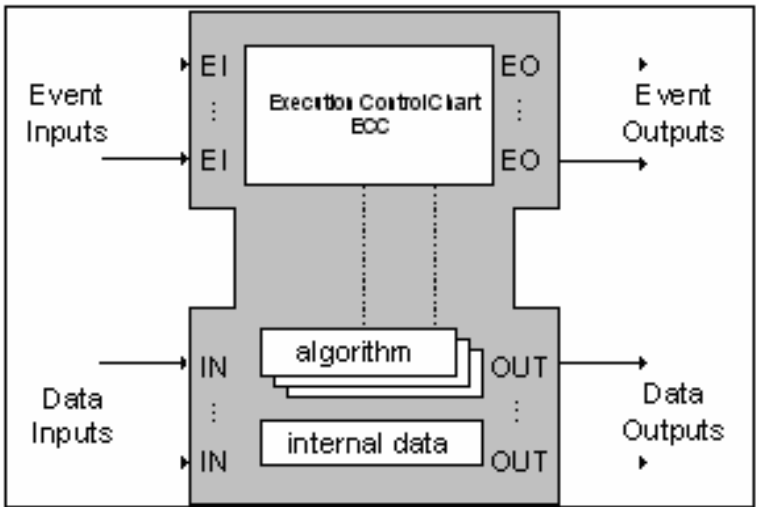


Fig. 2: **Function Block according to IEC 61499**

Additionally, the implicit method invocation by means of a combination of an event with several data inputs is very complicated. The differentiation in event and data connections has historical reasons. It comes from the world of cyclic running FBs. There it was the only possibility for a data exchange. But from the viewpoint of intended functionality, such an event driven approach is very close to a real function invocation. An explicit service method would be very helpful, that means a possibility of command driven architecture. Therefore, other approaches coming from computer science like object oriented design, middleware concepts or port based interactions have to be introduced in the control area.

Furthermore the software quality is a very important characteristic besides the area specific requirements of predictability and the usage of restricted hardware resources. From the viewpoint of the software engineering the software quality management is an integral part, including the complete software development and the maintenance. Therefore, additional standards describing the software life cycle [7] or the determination of product quality [8] should be considered.

The remainder of this paper is organised as follows: In the next section, I briefly refer to the known object oriented models and for quality management. In section 0 I will define the requirements to an object based, distributed automation system. Section 0 presents the DOME runtime environment as a reference implementation for a distributed object oriented model. Section 0 discusses measured quality aspects of the DOME reference implementation. In section 0 I will discuss an approach for the design of an application, running in DOME. Finally I conclude the paper in section 0.

2.1 Approaches from Computer Science for Distributed Systems

The data exchange between objects can be designed in different ways. The traditional object oriented design defines messages between objects. These messages are generally implemented as method calls. Several programming languages support object-oriented features, for instance Smalltalk, C++, Java or C#. Very common in embedded applications are C++ or Java.

Different approaches in other areas as automation were developed for the interaction between predefined objects. For instance CORBA ([9], [10], [11]) or (D)COM ([12], [13], [14], [15]) specify component models, based on abstract and public interface definitions and the possibility of usage of such components as black boxes from the viewpoint of their development and independent of the programming language. The disadvantages of both of these models is the lack of automation specific facilities like real-time aspects or small footprint. Both approaches follow the paradigm of a middleware concept. In this concept, the applications, which are designed in an object-oriented way, use a special middleware for the interaction between the (remote) objects. Normally, the middleware offers several services such as name services, transactions, security persistence or event notifications. The access to an other object is designed by an abstract interface, which is implemented by the object itself. The interaction between the objects can be done after the establishment of the link either directly or via the middleware.

Especially for Java the Jini approach is very popular for a distributed architecture. Jini offers the possibility for a self organized connection between objects of different nodes. In the same manner as the approaches above, the interface based technology and special protocols for the service location and usage are used here also[16].

2.2 Approaches for Real Time Aspects

Another important approach for embedded object interaction comes from the special design patterns. Such a pattern is the ROOM (Real-Time Object-Oriented Modelling) [17] concept. ROOM applies the UML (Unified Modelling Language, [18], [19] and following versions) to describe a set of constructs, suitable for modelling a system. In this pattern, the data exchange between objects will be done by means of ports and connectors. The interoperability between different operating systems is not the main focus in this approach. The classes, which are describing the object, get the stereotype <<capsule>> in ROOM. They interact with the environment by the ports. “Ports are objects whose purpose is to act as boundary objects for a capsule instance. They are ‘owned’ by the capsule instance in the sense that they are created along with their capsule and destroyed when the capsule is destroyed.” [17]. Ports are associated with a protocol that defines the data flow between the ports. So the internal implementation of objects is de-coupled completely from any knowledge about the environment. A connector is an abstract view about the communication between the interconnected ports. Each port plays a special role in this

interconnection, for example the relations of master-slave or proxy-stub are very common.

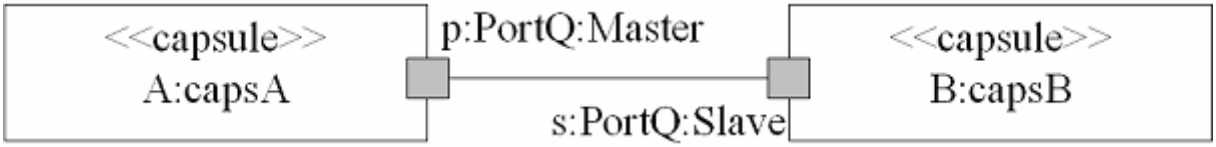


Fig. 3: **System Structure of object, port and connector [17]**

2.3 Approaches for Quality management

The software quality is an essential aspect in automation area. The technical procedures for the implementation of software components is thoroughly covered by the different specifications of the different approaches. But inside this specifications there is not such a lot of knowledge available about to measure or determine the quality of the components. This relates to both, to the software environment (firmware, operating system) and to the components of the control application. One approach to solve the problem of the software quality is the usage of design patterns. But such design patterns cover only a specific design problem or describing the interaction between the different component types. Up to now, patterns present functional solutions and do not provide a quantification of the component or the development process itself. Therefore general software measurement methods as a process for the quantification of attributes of the software objects or components are necessary. Two very common approaches are the static and the dynamic analysis of the software.

The static analysis quantifies syntactical and semantic aspects of the software. The results of such an analysis must be set in relation to a quality model. Well known are quality models, which are built according to the commonly used principles of the Goal/Question/Metric-Method[20].

Additionally a dynamic analysis is helpful for the investigation of the test coverage, for the generation of test cases or for getting detailed runtime information.

3 Requirements for a Distributed Control Systems

Some important criteria among other things have to be considered for the design of a distributed control system:

- New objects can be composed from existing ones (e.g. feature of Function Block approaches too).
- The interoperability must be guaranteed between heterogeneous devices.
- There must be possibilities for different object activation strategies (method calls) and it must be possible to define the behaviour of these activations.
- The usage of different communication services with predictable timing constraints, which are defined in engineering phase of the system.

- High efficiency, both for engineering and runtime, should be offered. In the current situation, the most effective phase for reducing costs is the engineering phase, because most controllers are very efficient and predominantly inexpensive.
- The possibility of textual restoration (documentation) of the whole system.
- Quantitative attributes about the software quality and quantitative attributes about the runtime features of the objects/components

A typical cycle for the creation of an automation system can be divided into three phases [21]:

1. Modelling and programming of the objects
2. Engineering of the complete system based on the available objects
3. Runtime / production

Furthermore there are interlocks between these phases, so the adaptation of the controller programme at runtime is necessary (e.g. downloads on the fly).

These automation area oriented requirements, the techniques of the middleware and the port approach shall be combined for a distributed, object-oriented control system.

4 The DOME Approach

4.1 Overview

DOME is the abbreviation for Distributed Object Model Environment. It is designed as a middleware, offering a range of capabilities for an automation specific object-oriented application[22].

In DOME, an application consists of objects, which can be predefined or developed by the user. The objects itself are defined in form of classes, in which features like abstract classes, inheritance or polymorphism are offered. For the clear reference of an object, each of them gets a unique name in the automation system. Therefore it is possible for the engineering of an application, to define all objects instances and a network of links between these objects. The links itself are also objects, called Link Objects, so all features of the mentioned class concept can be used for the definition of different link types. Therefore all objects, both Automation objects (with algorithms) or Link Objects, can be treated (creating, deleting etc.) in the same way (see Fig. 4).

The relation between the Automation Objects is typically Client-Server oriented. The offered services of such an object are described statically by the public methods of a class. We call these methods Service Interfaces. The interfaces, which need services from other objects, are called Required Interfaces. The ports represent explicit modelled connection points for the methods. So the Link Objects can be bound to the ports. The Link Objects itself offer services for local or remote port connections. Special implementations of Link Objects can offer services e.g. priorities or timing constraints.

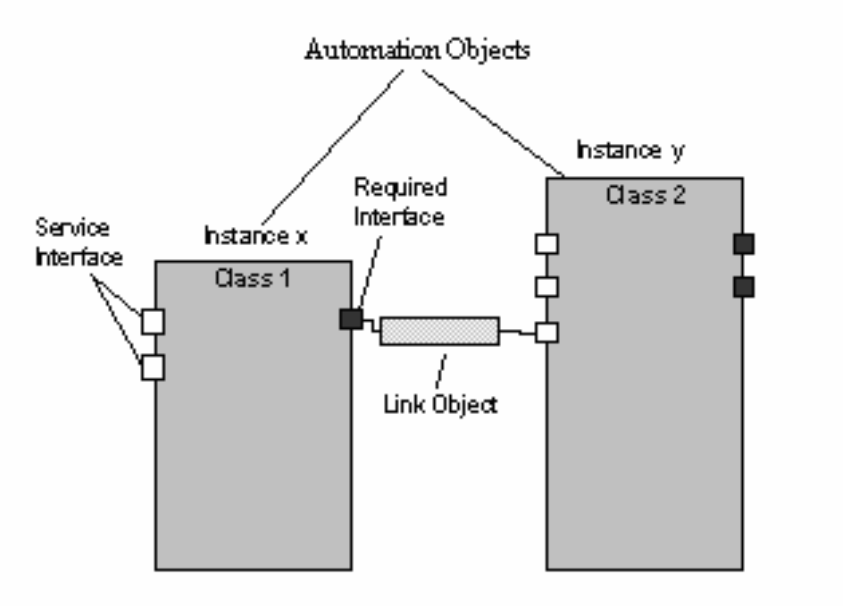


Fig. 4: **Connectors as Link Objects**

In order to manage all these objects, an efficient object management is necessary. The requirements are e.g. memory allocation, life time control or persistence. This object management is out of the scope of this paper, but by using the middleware concept, each device of the automation system have to have such an object management instance.

An other interesting feature of our DOME approach is the possibility for introspection. Therefore each object has its own reflexive interface in form of meta-information classes. Further, the approach is not limited to a special programming language. All common used object oriented languages like C++, Java or .Net-languages can be used, in order to implement DOME. So it is possible to use the best implementation platform for each device. The interoperability of such different implementations is guaranteed by the definition of the semantic of the bus protocols.

DOME represents a kind of middleware, so each device needs a kind of operating system, offering features like concurrent thread handling inside a process. DOME makes no assumptions about the underlying context switching of the threads. This has to be adapted to each specific (real-time) operating system. At least one process must be carried out by one device.

4 Application Model

DOME defines the design of a distributed system in top-down manner. The definition of the application model represents both, the Automation Objects and objects, relating to the DOME middleware (the runtime of the application objects). In DOME each device of the whole control system is called node. The node represents the basis for the device specific part of the distributed application. A node is always assigned to a platform, consisting of the operating system, the I/O system as interface to the automation process or as interfaces to the communication network.

The DOME middleware is represented by DOME_RT and is specific for the Platform. Based on DOME_RT, the local running part of the distributed application is defined in Fig. 5. Each local application is activated by the DOME_RT and assigned to one process. Each application consists of one object manager (ObjectManager). All other objects inherit some functionality from Object. The implementations of GroupBase represent the different scheduling strategies for the set of Automation Objects, which inherit from ClassObject.

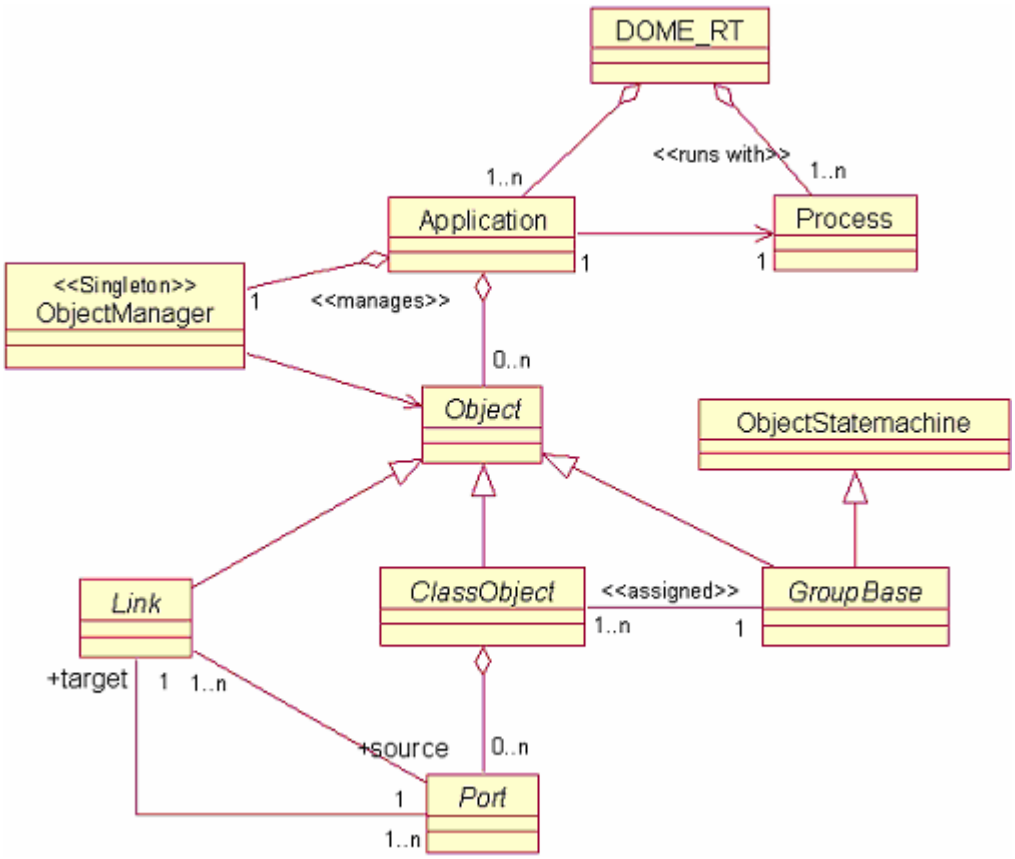


Fig. 5: **DOME Application Architecture**

The flexible usage of the objects is guaranteed by the late binding of the information/data flow between the objects. Therefore the ports are aggregated in each object. The ports decouple the connection of objects at runtime from the design of the algorithm inside the objects completely. So it is possible to handle the very complex interaction unchained from the object design. The heterogeneous access to the object Service Interfaces in different concrete implementations of DOME is handled by the DOME middleware, here the network/communication objects. This decoupling is necessary for the developer of the object algorithms, because this person does not know anything about the concrete runtime environment of the object. So it makes no difference in the design of the algorithms inside the objects.

The connection between the port can be handled as a permanent message channel, established at application design time, see Fig. 5. The information and data flow is always typed (the associated protocol in ROOM). This typed connection is more effective as a generic, non typed one. The non typed approach is very easy to implement, but has an additional footprint for the ineffective type checking of the

events/data (additional runtime errors or type mismatches are possible). In DOME, the type conform connection is guaranteed by checking the method signature of the connected ports at establishing the connection. A type conform connection between ports will be called compatible or pluggable [23]. The message channel represents in the world of Function Blocks the connections between these blocks.

Furthermore the information flow is directed from Required-Interface to Service-Interface, the dataflow is bi-directional. Therefore the roles of the port types are also defined: Required-Interface has a client role and the Service-Interface the server role.

Additionally, no compiler has to establish the connection between the ports. So the ports themselves have to check the type conform connection at runtime. The type conformity is given by a correspondence of the joint port methods. However on the other hand, special engineering tools will check the connections at design time. The link object between the ports decides about the protocol of the data exchange. This will be explained by the illustration of the different kinds of considered use cases of a local call, a call between active objects and between remote objects, see Fig. 6.

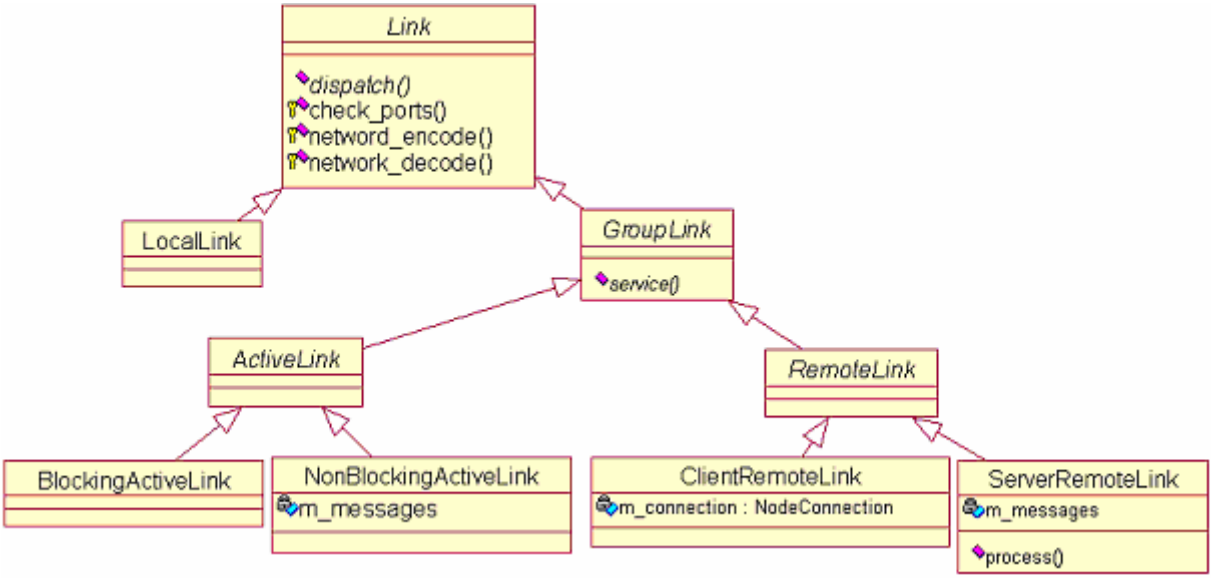


Fig. 6: **Definition of Link-Objects**

4.3 Kinds of Coupling

The advantage of using ports for the connection between Automation Objects will be visible, when we consider the different kinds of coupling in a distributed automation system.

A local link can be used between objects of the same thread context. The information flow is not interrupted. A local call between ports can be handled directly by the offered interface methods of the ports. The wrapped method of the server object is executed immediately.

A call between active objects [24] is more difficult than the local call. This pattern is very interesting for the concurrent programming of applications. The concurrent,

distributed applications will be more and more interesting in the automation area too. We have adapted the Active Object pattern to the usage of ports as the interaction object of an Automation Object. Therefore, the original object does not have to be changed and is totally independent of its environment. There are differences in the runtime. At least the called object has to be assigned to its own scheduler, e.g. a specialisation of GroupBase, see Fig. 5. The Required-Interface can not directly call the offered interface method of the Service-Interface, because they both run in different thread contexts. Therefore the message has to be marshalled and buffered. In DOME, the Required-Interface is responsible for the marshalling of the parameters and the Service-Interface queues the messages in the environment. It depends on the scheduling strategy to which message the environment will handle the next time. A simple FIFO strategy was implemented in which the adding, removing and call-backs to the Required-Interface are guarded by semaphores, in order to handle the race conditions.

Depending on their coupling, not so closed coupled objects can run in different processes. These processes can run on the same device or on different devices. Therefore the local loads can be balanced between the devices. In order to connect objects in different processes, a remote link object will be used. Such a connection is only possible by two special sub-link objects, called ClientRemoteLink and ServerRemoteLink. Furthermore a communication object is involved, see Fig. 7. This object acts as an Active Object, so the connections to these objects are always active link types, discussed above. Each process has such a communication object. Between the communication objects datagrams are transmitted, e.g. IP for Ethernet. This can be done in an asynchronous way. At normal client-server interactions, the sending communication object has to wait (e.g. sleeping or do other things meanwhile) for the response.

5 Quality Measurement of the Realized Framework

In order to determine aspects of the quality of DOME, the C++ source code (very efficient, very portable) of DOME has to be analysed. This investigation was done by means of the Telelogic Tau® Logiscope™ tool, which realises the static and dynamic source code analysis [25]. The C++ reference quality model of Logiscope has been used therefore.

Particularly the model is devided into three parts, the application level, the class level and the functional level, according to the Goal/Question/Metric approach. These levels are dedicated to several typical object-oriented features of C++. The application level evaluates the whole programme, the class level collects metrics about classes and the functional level is focused on the properties of the functions (methods of the classes). Sub devisions inside the levels are the analyzability, the changeability, the stability and the testability, each of them are based on specific metrics, e.g. Metrics for Object Oriented Design, Cyclomatic Complexity of FAN-IN, FAN-OUT. The results of the investigations of these three levels are summarized in qualitative statements: EXCELLENT, GOOD, FAIR and POOR, whereas each sub devision gets a different weighted factor.

At application level, DOME gets the qualitity statements between FAIR and GOOD. These relatively inadequate evaluation depends on the fact, that DOME is not an application but a framework.

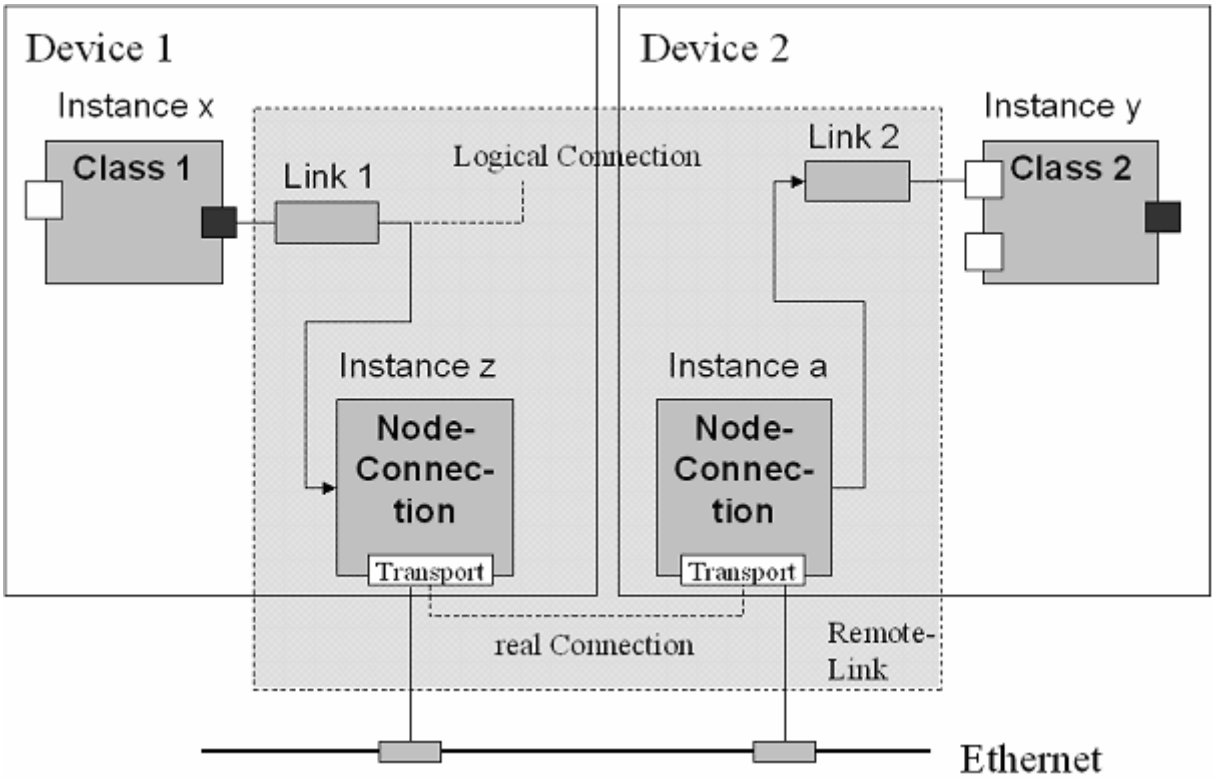


Fig. 7: **Remote access to a port**

At class level, most criteria are evaluated excellently, see Table 1. The results represent the ratio of the classes, which are evaluated with the special quality statement.

Criteria	EXCELLENT [%]	GOOD [%]
Analyzability	23,81	76,19
Changeability	95,24	4,76
Stability	100,00	0,00
Testability	95,24	4,76

Table 1: **Evaluations of criteria at class level**

At function level, most methods are evaluated as GOOD or EXCELLENT, see Table 2. The results represent the ratio of the methods, which are evaluated with the special quality statement.

Criteria	EXCELLENT [%]	GOOD [%]
Analyzability	13,71	84,96
Changeability	96,19	3,62
Stability	99,43	0,57
Testability	91,59	8,03

Table 2: **Evaluations of criteria at function level**

6 Object Desgin in DOME-L

6.1 Language Features

The DOME-L is a specially introduced programming language to utilise the ideas and the potential of the DOME approach. There are two main objectives behind DOME-L. At first the ports, classes and properties should be described on a semantically high level. This reduces redundant work of trivial and error-prone things and lets the programmer concentrate on the real implementation work. The second aim is to be very close to the underlying programming language to achieve the full potential and efficiency of these language. This results also in a simple DOME-L that can be easily used by the programmer. He does not need to learn something totally new as the implementation is done in the native programming language. There is also no need for hidden executed functions, e.g. proxies and stubs, that may confuse or make it difficult to understand the inner working of DOME-L.

One important feature of the DOME approach is the dynamic instantiation of objects and communication links between these objects at runtime. The ports are well defined communications points between objects and are linked against ports of other objects as long as the ports are type compatible with each other. For the object itself is completely transparent if the link communicates directly to another object or if it communicate through a network connection to a remote object. To realise this behaviour metadata about the ports and classes are required at runtime. This meta data is derived from the DOME-L written specification of ports and classes. A developed DOME-L compiler generates classes and methods in the target programming language and enhances it with all required meta data. This compiler provides also a runtime library that contains all DOME interface definitions and base classes. The generated metadata implements these interfaces and can generically be accessed from any other object in this way. The network link object for example uses the metadata to encode/decode the communication data for the network transport.

Furthermore a documentation tool support is an interesting feature of DOME-L. The DOME-L compiler supports the Doxygen documentation tool [26]. It detects documentation comments and passes them through to the generated source code. The Doxygen tool can then be normally used to create documentation of the source code.

The first implementation uses C++ as target language. A DOME-L specification is translated into a module that consists of a header file and the implementation. The header file contains the definitions about the port meta classes for reuse by other modules. The implementation contains the DOME classes with all meta information.

6.2 Quality of the Generated Objects

The generated C++ code was analysed by the Logiscope™ tool, investigating quality aspects of the code. Here the same levels were evaluated as for the DOME framework. Especially for these objects, the application level is completely uninteresting. Therefore only the class and function levels will be discussed. The measurements were investigated by four different object implementations according to the approaches of IEC 61131, IEC 61499, improved IEC 61499 (combined event and data flow in one direction) and a client-server implementation (control flow and bi-directional data flow). The objects have to calculated a simple limit checker. The algorithm inside is very simple and is adapted from an example from IEC 1131-3.

Following results are measured for class level for the different implementations:

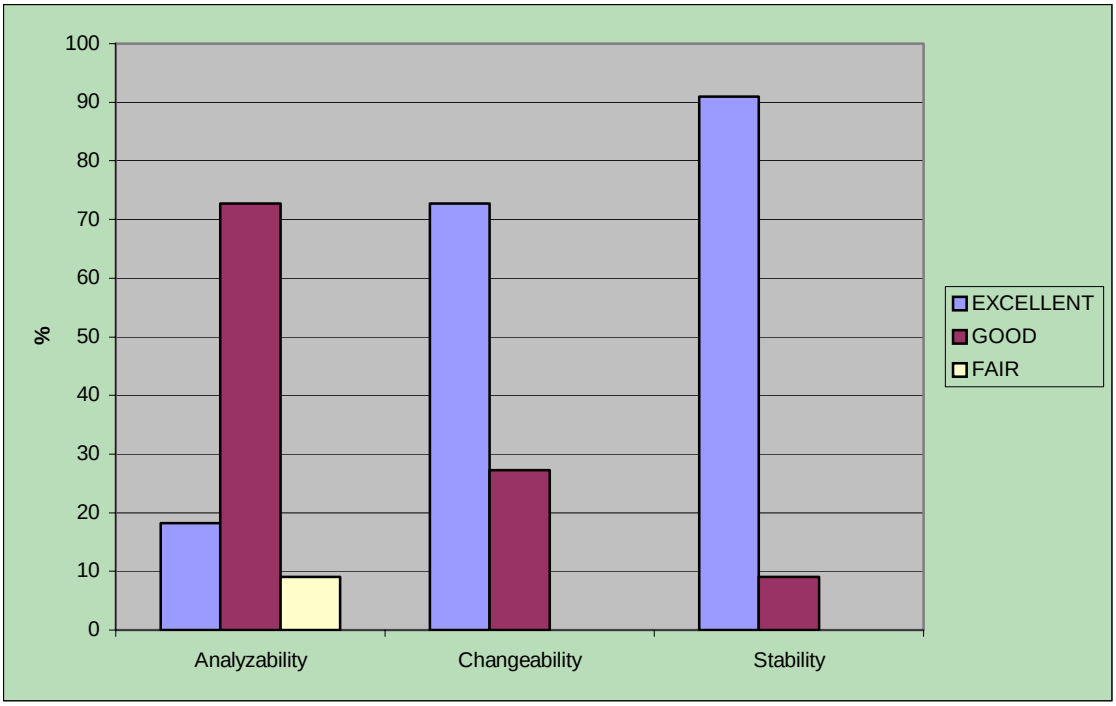


Fig. 8: **Function distribution of the class level of IEC 61131 objects**

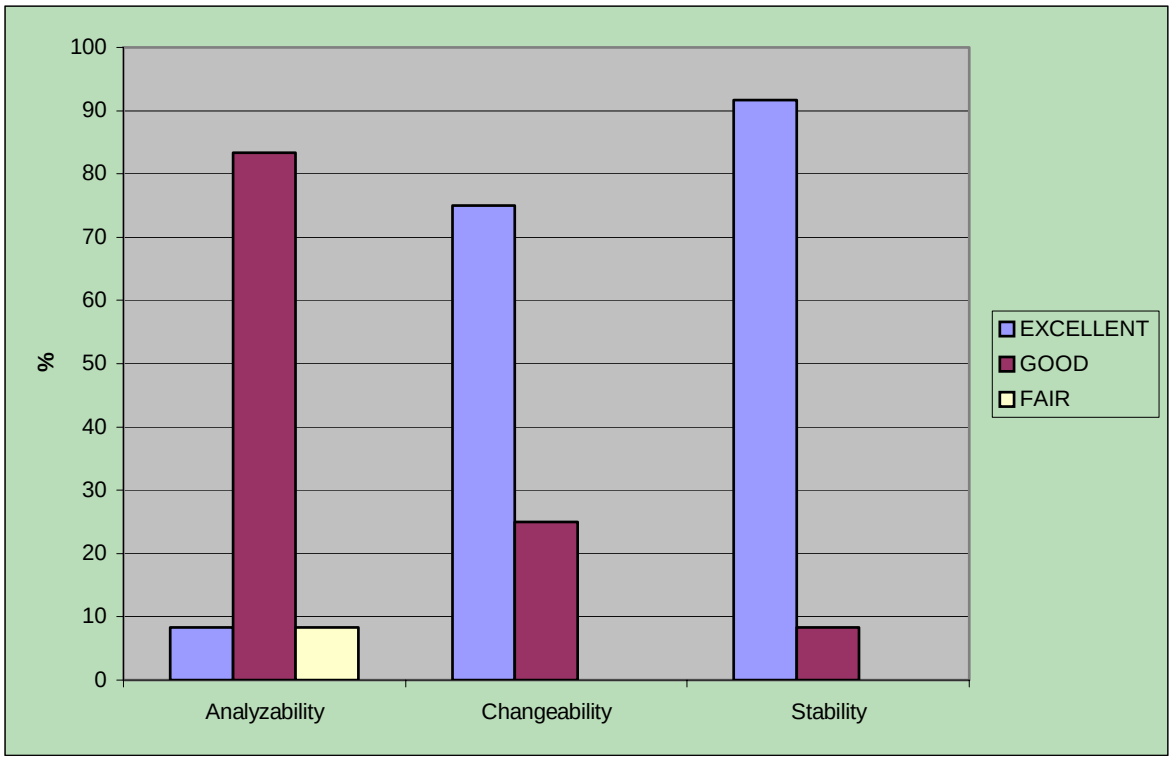


Fig. 9: Function distribution of the class level of IEC 61499 objects

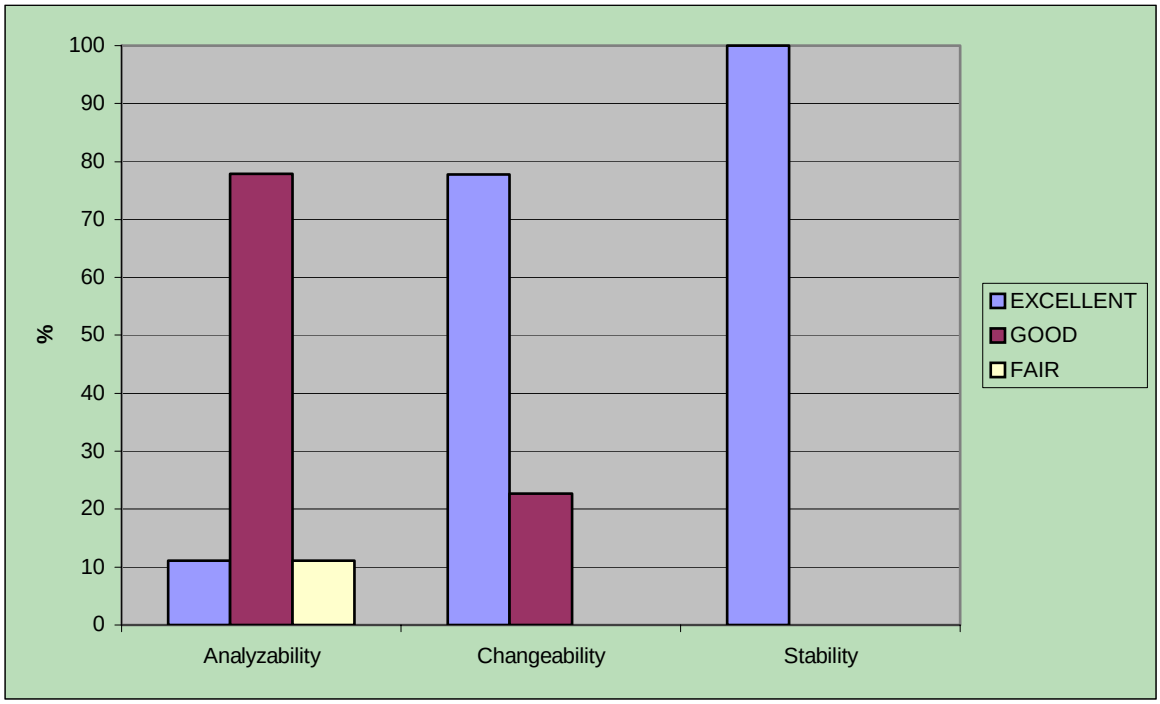


Fig. 10: **Function distribution of the class level of improved IEC 61499 objects**

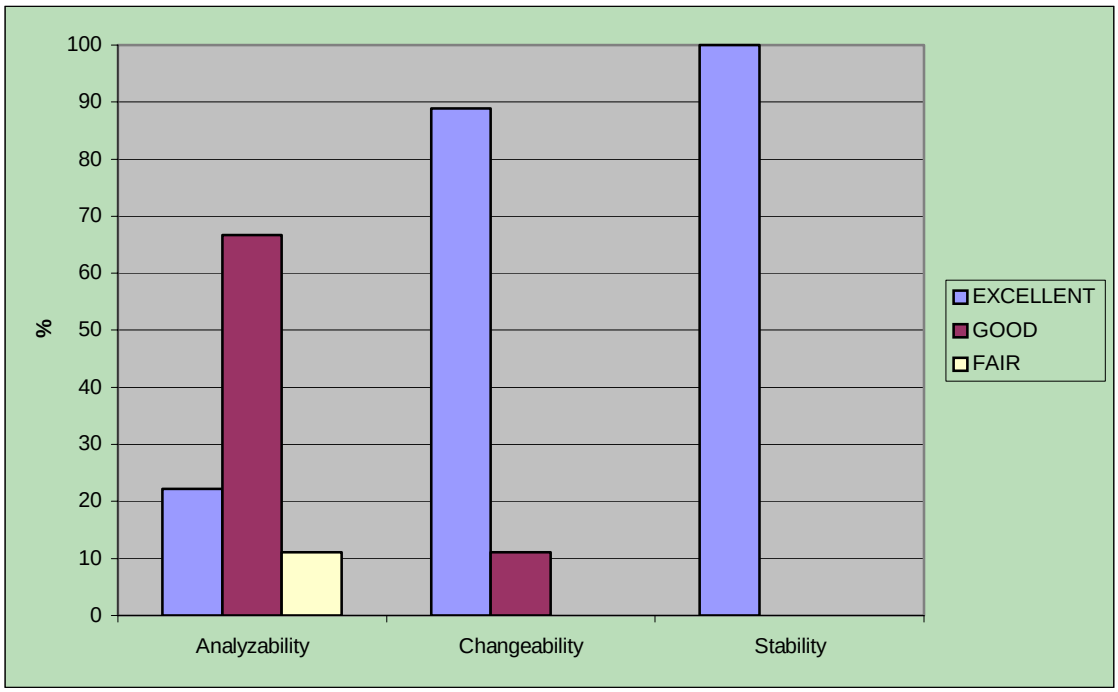


Fig. 11: **Function distribution of the class level of objects with client-server relation**

In all cases, the testability was about 100%, so it is not relevant for the comparison. There are nearly no differences between the approaches of IEC 61131 and IEC 61499. The most the classes can be analysed excellently or good. Significant differences can be shown by the usage of a combined event and data flow. The better stability statements depends on the reduces number of ports for the explicit event flow, therefore there are fewer interface methods of such objects. The next stage of improvements came with the introduction of a bi-directional data flow in combination with the client-server-relation between the objects. More classes are better analyzeable and the changeability is improved.

For the function level, nearly the same statements can be done. The criteria changeability and stability are always excellent. There are tiny differences for analysability and testability in this manner, that the effort for the explicit event connections between the objects results in additional (public) methods. The biggest effort can be recognized in the implementation according to IEC 61499, the qualitative best implementation offers the implementation according to the client-server paradigm, see Table 3.

Criterium	IEC 61131	IEC 61499	Improved IEC 61499	Client-Server-Relation
Analyzability				
EXCELLENT	8,70 %	8,87 %	9,76 %	11,11 %
GOOD	91,30 %	91,13 %	90,24 %	88,89 %
Testability				
EXCELLENT	98,26 %	98,39 %	100,00 %	100,00 %
GOOD	1,74 %	1,16 %	0,00 %	0,00 %

Table 3: **Function distribution of the function level criteria**

7 Conclusions and Future Work

The advantages of the presented middleware concept are the simplicity and flexibility of the late instantiating of connections between the Automation Objects. Furthermore these objects are totally independent of their real connections to other Automation Objects or to the environment. DOME based object designs are well suited for the adaption of the very popular Function Block concept. Additional to the data flow concept between Function Blocks, command flows are possible between Automation Objects. The command and data flow is always type safe.

An great advantage is the possibility to design the automation application in UML-RT. Such a model can be converted into DOME-L very easily. So all positive effects of the very closed coupling to the DOME runtime and to the native programming language can be used at commissioning the system. Furthermore, the generated native code is very efficient, e.g. the ports are integrated as virtual methods inside the DOME objects and no additional instantiations are necessary.

A small disadvantage is the additional indirection by calling an object service method, because a Link Object is always involved in the information flow.

Interesting in future examinations is the usage/generation of quality metrics inside the design process of an Automation Object. This is possible, when the metrics as mentioned above of the generated target code will be examined and the relevant metrics values will be put into the available meta information about the object. Such information is interesting as well as in the engineering phase of a control system or at runtime in order to prove the correct behaviour of the objects, e.g. timing constraints.

References

[1] International Electrotechnical Commission: IEC 61131 – Programmable Controllers, parts 1-5, Genf, 1992

[2] International Electrotechnical Commission: IEC 61131 – Programmable Controllers - Part 3, Programming Languages, 2nd Edition, Genf, 1999

[3] International Electrotechnical Commission: Draft IEC 61499 - Function Blocks for Industrial Process Measurement and Control Systems, Part 1 and 2, TC65/WG6, Genf, 1999

[4] International Electrotechnical Commission: Technical Committee No. 65C: Digital Comm., WG 7: Function blocks for Process Control, IEC 61804, General Requirements, IEC Draft 1999

[5] Rockwell Automation, FBDK, **http://www.holobloc.com/fbdk/README.htm**

[6] CORFU-FBDK, **http://seg.ee.upatras.gr/corfu/**

[7] International Electrotechnical Commission: *Information Technology – Software Life Cycle Processes*, IEC 12207, International Electrotechnical Commission, Genf, 1995

[8] International Electrotechnical Commission: *Product Quality*, parts 1-4, IEC 9126, International Electrotechnical Commission, Genf, 2001

[9] Object Management Group: *The CORBA IDS Specification*, Framingham, Mass: OMG, 1997

[10] Object Management Group: *A Discussion about the Object Management Architecture*, Framingham, Mass:OMG, 1997

[11] Object Management Group: *The Common Object Request Broker: Architecture and Specification*, Framingham, Mass:OMG, 1997

[12] Microsoft Corporation: *The Component Object Model Specification*, Draft 0.9, 1995

[13] Microsoft Corporation: DCOM Architecture White Paper, 1997

[14] Microsoft Corporation: DCOM Technical Overview, 1997

[15] Redmont, F. E.: *DCOM: Microsoft Distributed Component Model*, IDG Books Worldwide, 1997

[16] Sun Microsystems, Jini, **http://www.sun.com/jini**, 1998.

[17] Selic, B.; Gullekson, G., Ward, P. T.: *Real-Time Object-Oriented Modelling*, John Wiley & Sons, New York, 1994

[18] Object Management Group: *The UML Semantics*, Version 1.1, Framingham, Mass:OMG, 1997

[19] Selic, B.; Rumbaugh, J.: *Using UML for Modeling Complex real-Time Systems*, Rational Software, 1998.

[20] Solingen, R. v.; Berghout, E.: *The Goal/Question/Metric Method*, Mc-Graw-Hill, London, 1999

[21] Meyer, D.: *Objektverwaltungskonzept für operative Prozeßleittechnik*, PHD thesis on the RWTH Aachen, 2001

[22] Riedl, M.; Diedrich, Ch.; Naumann, F.; Simon, R.: *An Object Based Approach for Distributed Automation*, IEEE Africon'04; 2004

[23] Hoang, M.S.: DIO - *Ein komponentenbasiertes Softwaremodell für verteilte Automatisierungssysteme*, PHD Thesis Technische Universität Dresden, 1998

[24] Schmidt, D.; Lavender, G.: *Active Object: An Object Behavioural Pattern for Concurrent Programming*, Pattern Languages of Program Design 2, ed. John M. Vlissides et al., Addison Wesley 1996

[25] Telelogic Tau Logiscope 6.0: Diverse Manuals, recognized at **www.telelogic.com**, 2004

[26] **www.doxygen.org**

Springer Publ., Aachen, 2004 (602 pages)
ISBN 3-8322-3383-0

This proceedings of the joined conferences, the 14th International Workshop on Software Measurement (IWSM 2004) and the DASMA MetriKon 2004, try to reflect a bit of all the concepts developed and the experiences made when measuring software. They are of particular interest to software engineering researchers, as well as to practitioners, in the areas of project management and quality improvement programs, for both software maintenance and software development.

Ebert, C.; Dumke, R.; Bundschuh, M.; Schmietendorf, A.:

Best Practices in Software Measurement

Springer Publ., 2004 (320 pages)
ISBN 3-540-20867-4

The software business is challenging enough without having to contend with recurring errors. One way repeating errors can be avoided is through effective software measurement. In this book is offered a practical guidance built upon insight and experience. The authors detail knowledge and experiences about software measurement in an easily understood, hands-on presentation and explain many current ISO standards (see also **`http://metrics.cs.uni-magdeburg.de/`**).

Pandian, C. R.:

Software Metrics – A Guide to Planning, Analysis, and Application

CRC Press Company, Boca Raton, 2004 (286 pages)
ISBN 0-8493-1661-8

The book simplifies software measurement und explains its value as a pragmatic tool for management. Ideas and techniques presented in this book are derived from best practices. Some of the keywords are fundamentals of software measurement, metrics system architectures, regression models, exploring metrics for defect management, and strategic visions.

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

1.1.1.1.1 Investigations in Software Measurement

Shaker Publ., Aachen, 2003 (326 pages)
ISBN 3-8322-1880-7

ThisThe book includes the proceedings of the 13th International Workshop on Software Measurement (IWSM2003) held in Montreal in September 2003. It is a collection of theoretical studies in the field of software measurement as well as experience reports on the application of software metrics in companies and universities of Argentinia, Canada, Finland, Germany, India, Italy, Japan, and Netherlands.

Preprints/Technical Reports:

Dumke, R.; Coté, I.; Andruschak, O.: *Statistical Process Control (SPC) – A Metrics-Based Point of View of Software Processes Achieving the CMMI Level Four*. University of Magdeburg, 2004

April, A.A.; Dumke, R.R.; Abran, A.: *SM^{mm} Model to Evaluate and Improve the Quality of the Software Maintenance Process*. University of Magdeburg, 2004

Dumke, R.; Schmietendorf, A.; Zuse, H.: *Formal Description of Software Measurement and Evaluation*. University of Magdeburg, 2005

IASTED SE 2005:

IASTED International Conference on Software Engineering 2005
February 15-17, 2005, Innsbruck, Austria
see: http://www.iasted.org/conferences/2005/innsbruck/se.htm

SEPG 2005:
17th Software Engineering Process Group Conference
March 7-10, 2005, Seattle, Washington
see: http://www.sei.cmu.edu/sepg/main.htm

SMEF 2005:
Software Measurement European Forum
March 16-18, 2005, Rome, Italy
see: http://www.iir-italy.it/smef2005/

CSMR 2005:
9th European Conference on Software Maintenance and Reengineering
March 21-23, 2005, Manchester, UK
see: http://www.rcost.unisannio.it/csmr2005/index2.html

EASE 2005:
9th International Conference on Empirical Assessment in Software Engineering
April 11-13, 2005, Staffordshire, UK
see: http://ease.cs.keele.ac.uk/call_for_papers.html

ASQT 2005:
Arbeitskonferenz Softwarequalität und Test 2005
April 27-29, 2005, Klagenfurt, Austria
see: http://www.ifi.uni-klu.ac.at/Conferences/ASQT2005/

SPICE 2005:
5th International SPICE Conference on Process Assessment and Improvement
April 28-29, 2005, Klagenfurt, Austria
see: http://www.ifi.uni-klu.ac.at/Conferences/SPICE2005

PQST 2005:
International Conference on Practical Software Quality & Testing
May 2-6, 2005, Las Vegas
see: http://www.psqtconference.com/2005west/

WMM 2005:
1st Workshop on Web Measurement and Metrics
May 10, Chiba, Japan
see: https://www.cs.auckland.ac.nz/wmm05/

WWW 2005:

International World Wide Web Conference
May 10-14, 2005, Chiba, Japan
see: http://www2005.org/

IWPC 2005:

International Workshop on Program Comprehension
May 15-16, 2005, St. Louis
see: http://www.ieee-iwpc.org/iwpc2005/

PROMISE 2005:

International Workshop on Predictor Models in Software Engineering
May 16, 2005, St. Louis, Missouri
see: http://promise.site.uottawa.ca/

ICSE 2005:

3rd Workshop on Software Quality
May 17, 2005, St. Louis, Missouri
see: http://attend.it.uts.edu.au/icse2005/

REBSE 2005:

Workshop on Realising Evidence-Based Software Engineering
May 16, 2005, St. Louis, Missouri
see: http://evidence.cs.keele.ac.uk/rebse.html

SIGMetrics 2005:

ACM SIGMetrics - Performance 2005
June 6-10, 1005, Banff, Alberta, Canada
see: http://www.cse.cuhk.edu.hk/~sigm2005/

PE2005:

6. Workshop Software Performance Engineering
10. Juni 2005 in Berlin,
see: http://ivs.cs.uni-magdeburg.de/~gi-peak/

ESEPG 2005:

10th European Software Engineering Process Group Conference
June 13-16, London, UK
see: http://www.espi.org/sepg/

PROFES 2005:

6th International Conference on Product Focused Software Process Improvement
June 13-16, 2005, Oulu, Finland
see: http://profes2005.oulu.fi/

QAST 2005:

1st Workshop on Quality Assurance and Software Testing
June 27-30, 2005, Las Vegas
see: http://people.cs.und.edu/~reza/QAST05.htm

WOSP 2005:

5th International Workshop on Software & Performance
July 11-15, 2005, Las Palmas, Spain
see: http://wosp2005.uib.es/

QATWBA 2005

2nd International Workshop on Quality Assurance and Testing of Web-Based Applications
July 25-28, 2005, Edinburgh, UK
see: http://aquila.nvc.cs.vt.edu/compsac2005/

ICWE 2005:

5th International Conference on Web Engineering
July 25-29, 2005, Sydney, Australia
see: http://www.icwe2005.org/

SPPI 2005:

Software Process and Product Improvement - 31th Euromicro Conference
August 30 - September 3, 2005, Porto, Portugal
see: http://www.sea.uni-linz.ac.at/SPPI2005/

IWSM 2005:

15th International Workshop on Software Measurement
September 12-14 in Montreal, Canada
see: http://www.lrgl.uqam.ca/workshops/iwsm2005/

QFD 2005:

17th Symposium on Quality Function Deployment
September 15-23, 2005, Portland, Oregon
see: http://www.qfdi.org/call_for_papers.htm

METRICS 2005:

10th International Symposium on Software Metrics
September 19-22, 2005, Como, Italy
see: http://metrics2005.di.uniba.it/

QSIC 2005:

International Conference on Software Quality
September 19-21, Melbourne, Australia
see: http://www.ict.swin.edu.au/conferences/qsic2005/

ISMA 2005:

1th Annual International Software Measurement and Analysis Conference

September 18-23, 2005, New Orleans

see: **http://www.ifpug.org/press/2005ConferenceAnnouncement.htm**

SOQUA 2005:

International Conference on Software Quality

September 19-22, Erfurt, Germany

see: **http://www.mathematik.uni-ulm.de/sai/jmayer/soqua05/**

QEST 2005:

International Conference on Quantitative Evaluation of SysTems

September 19-22, Torino, Italy

see: **http://www.qest.org/**

3WCSQ 2005:

World Congress on Software Quality

September 26-30, 2005, Munich, Germany

see: **http://www.isqi.org/isqi/eng/conf/wcsq/3/**

UKSMA 2005:

16th Annual UKSMA Conference - Managing your Software (through Measurement)

October 13, 2005, London, UK

see: **http://www.uksma.co.uk/**

MetriKon 2005:

DASMA Workshop

November 15-16, 2005, Kaiserslautern

see: **http://www.metrikon.de/**

ISESE 2005:

ACM-IEEE 4th International Symposium on Empirical Software Engineering

Nov 17-18, 2005, Noosa Heads, Australia

see: **http://attend.it.uts.edu.au/isese2005/cfp.htm**

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://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://www.lrgl.uqam.ca/**
  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 of 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 have 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.dasma.org**
  DASMA Deutsche Anwendergruppe für SW Metrik und Aufwands-schätzung e.V.

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

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

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

- **http://www.mai-net.org/**
  Network (MAIN) Metrics Associations International

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

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

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

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

- **http://user.cs.tu-berlin.de/~fetcke/measurement/products.html**
  Tool overview by Thomas Fetcke

- **http://zing.ncsl.nist.gov/WebTools/tech.html**
  An Overview about Web Metrics Tools

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