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

Editors:

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

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

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

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

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

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

9th International Workshop on Software Measurement

of the German Interest Group on Software Metrics and the
Canadian Interest Group on Metrics (C.I.M.)
In cooperation with
COSMIC – Common Software Measurement International Consortium
September 8-10, 1999
in Montreal - Mont-Tremblant (Québec) CANADA

SCOPE

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.

Software suppliers face the challenge of translating customer requirements into the size of software to be produced as a key step in their project cost estimating. Customers want also to know the size delivered as an important component of measuring supplier performance. Given the explosive growth and diversity of software contracting and outsourcing, suppliers and customers need more accurate measurement methods which must work equally reliably across all types of software.

Current methods for measuring the size of software are not always of sufficient strength to meet market needs, or work only for restricted types of software. Industry urgently needs software size measures which extensive coverage across diversified software portfolios and within a rapidly evolving market.

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

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

The outcome of the workshop is the joint authorship of white papers to be tabled as inputs to the COSMIC (Common Software Measurement International Consortium) for the design of the next generation of software size measurement methods.

We are looking for position papers in the area of software measurement design and software measurement size applications from (but not limited to) on the following issues and topics:

❖ **Objects and size 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 or emerging technologies (OO, Multi-media, Web-based applications, etc.)
- Size attributes categories (Functional, Technical, Quality, etc.)

❖ **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.

❖ **How to address practical deficiencies that slows down penetration rate with practitioners:**

- requirements prior to measurement process (documentation gathering, reformatting of documentation, etc).
- Requirements during the measurement process (automation of data collection, integration with historical data, validation of results, etc.)
- Quality of measurement methods (repeatability accuracy, correctness, traceability, uncertainty, precision, etc).

❖ **Uses of measurements results in relationships with other measures**

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

PROGRAM COMMITTEE

Alain Abran, University du Québec, Montreal, Canada
Reiner Dumke, University of Magdeburg, Germany
Horst Zuse, TU Berlin, Germany
François Coallier, Bell Canada, Canada
Charles Symons, COSMIC - UK

SUBMISSIONS

4 Position Papers

Authors should send abstracts (1-2 pages) by mail, fax or e-mail by **April 1st , 1999** to

Alain Abran
University of Quebec
Dept. of Computer Science
C.P. 8888, Succ. Centre-Ville
Montreal (Quebec), Canada H3C 3P8
Tel.: +1-514-987-3000
Fax: +1-514-987-8477
email: abran.alain@uqam.ca

or to

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

WORKSHOP TIMETABLE

Submission deadline of abstract: April 1st, 1999
Notification of acceptance: April 15, 1999
Position paper deadline: August 1, 1999
Workshop date: September 8-10, 1999

FEES: none

NEWS

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

http://www.lrgl.uqam.ca/iwsm99/

The next Workshop on the *German Interest Group on Software Metrics* will take place at the

University of Regensburg,

September 30 – October 1, 1999

Arbeitskreis Softwaremetriken der GI-Fachgruppe 2.1.9

Key topics will be (but are not limited to)

Experiences in Metrics Applications,
New Areas of Metrics Applications,
Foundations on Software Measurement,
Metrics for new Software Technologies.

The Call for Papers will be published in the next "*Metrics News*".

Reiner Dumke
Chair of the GI-Metrics Group

Discussions and Presentations of the

8th International Workshop on Software Measurement

of the German Interest Group on Software Metrics

and the Canadian Interest Group on Metrics

*in **September 17** and **18, 1998** at the*

University of Magdeburg

The following description gives the contents of the abstracts of the published presentations at this workshop.

The full papers are available in the book

Dumke/Abran (Eds.):
Software Measurement – Current Trends in Research and Practice.
Deutscher Universitätsverlag, Wiesbaden, 1999
ISBN 3-8244-6876-X

THIRTY YEARS SOFTWARE MEASUREMENT

Horst Zuse, Technische Universität Berlin (Germany)

From our view, the year 1998 is the 30. anniversary of software measurement. It may be, that the earliest paper about software complexity was published by Rubey et al. in 1968. There is no reference to an earlier publication. Boehm et al., point out about this paper: Many software attributes and their metrics are defined and discussed. There is a total of 57 attributes distributed among seven categories. They are related to mathematical calculations, program logic, computation time and memory usage, program modifiability, etc. The metrics are realized as formulas and are explained in detail. Also in 1971 Knuth published a paper of an empirical investigation of FORTRAN programs.
The magnitude of costs involved in software development and maintenance magnifies the need for a scientific foundation to support programming standards and management decisions by measurement. Already in 1980 Curtis pointed out: Rigorous scientific procedures must be applied to studying the development of software systems if we are to transform programming into an engineering discipline. At the core of these procedures is the development of measurement techniques and the determination of cause effect relationships. As highlighted above, the establishment of well accepted structures of measurement in physics took a very long time. It is our impression that the same holds for software measurement. In the next sections we present an overview of software measurement from a historical perspective. A more detailed description of the history of software measurement can be found in Zuse: *A Framework of Software Measurement*. de Gruyter Publisher, Berlin, New York, 1998.

FUNCTION POINT EVOLUTION

Charles R. Symons, Software Measurement Service Ltd. Kent (England)

Charles Symons discussed the strengths and weaknesses of various ways of measuring "Software Size", which is a key component of performance measures and of estimating methods, namely

- Source Lines of Code,
- The "Function Point" method originally developed by Allan Albrecht of IBM, now standardised as the IFPUG 4.0 method,
- The MkII FP method, designed to overcome certain weakness of the Albrecht method, developed by the author,
- The "Full Function Point" method, developed by Alain Abran et al from Quebec, which aims to extend the IFPUG method to handle real-time software,
- The ISO standard 14143 Part 1 which establishes some principles of "Functional Size Measuremen".

Current Function Point methods for software sizing can and should be updated to meet modern software engineering needs

- re-base on modern software engineering concepts,
- generalise to be applicable in a wider range of software domains (e.g. real-time, process control, etc.),
- above all to improve and prove the accuracy of performance measurement and early life-cycle estimating,
- improved integration of early life-cycle estimating methods based on functional requirements with methods used later in the life-cycle based on components and activities.

The economic value of finding a solution to these needs would be enormous. The way forward in his view is to be found in ideas of Industrial Engineering published by Frederick Taylor in 1911. Symons proposes an "International Software Metrics Initiative" to meet these objectives.

METRICS VALIDATION PROPOSALS: A STRUCTURED ANALYSIS

Jean-Philippe Jacquet and Alain Abran, Université du Québec à Montréal (Canada)

In the literature, the expression metrics validation is used in many ways with different meanings. This paper analyzes and compares some of the validation approaches currently proposed. The basis for this analysis is a process model for software measurement methods which identifies the distinct steps involved from the design of a measurement method to the exploitation of the measurement results. This process model for software measurement methods is used to position various authors' validation criteria according to the measurement process to which they apply. This positioning enables the establishment of relationships among the various validation approaches. It also makes it possible to show that, because none of these validation approaches proposed to date in the literature covers the full spectrum of the process of measurement methods, a complete and practical validation framework does not yet exist.

ON THE USE OF A SEGMENTALLY ADDITIVE PROXIMITY STRUC-TURE TO MEASURE OBJECT CLASS LIFE CYCLE COMPLEXITY

Geert Poels, Catholic University of Leuven (Belgium)

According to Whitmire's about object-oriented software metrics software is characterised by three dimensions. The data dimension refers to what the software system remembers. The function dimension relates to what a system does. And finally, the control or dynamic behaviour dimension considers the different behavioural states of software. Most object-oriented software measures focus on aspects of the data and function dimensions. For instance, Chidamber's et al. MOOSE metric suite is used to measure the static structure of an object-oriented design (depth of inheritance, number of children), the structural complexity of the design (weighted methods per class, coupling between objects, response for a class), and the interaction between functions and data (lack of cohesion in methods).
Poels et al. (Poels97) propose a measure for a specific aspect of the dynamic behaviour of objects, i.e., life cycle complexity. After creation, the methods of an object can be invoked. The effect of this method triggering is that the state of the object changes. Although the triggering of methods is decided at run-time, the invocation order is subject to constraints imposed by the problem domain. These constraints are specified in the abstract data type definition of the object. The attribute 'life cycle complexity' refers to the complexity of this specification.
In (Poels97) a life cycle complexity measure was proposed using the so-called distance-based approach. Basically, the attribute 'life cycle complexity' is defined as the distance from the life cycle specification to some reference point, chosen by the person that performs measurement. The subjective reference point models the life cycle as if it were not complex at all. The greater the distance between the actual life cycle specification and the artificial zero complexity specification, the greater the life cycle complexity. The distance-based approach contains a constructive procedure to define a metric space that is used to represent these distances.
The main advantages of the distance-based approach are intuitiveness and flexibility. However, it was not clear how the approach fits into the representational theory of measurement. In (Poels97) only the numerical conditions for a metric space were considered. To define a metric as a measure in the sense of measurement theory, we also need to consider empirical conditions. This is the main focus of the current paper. We show that the function defined using the distance-based approach is not merely a metric, but an additive metric. Moreover, the approach constructs a segmentally additive proximity structure, which is exactly the empirical relational structure assumed by an additive metric. Using a theorem of Suppes et al. we further show that the scale type of the life cycle complexity measure is ratio.

ATTRIBUTE-BASED MODEL OF SOFTWARE SIZE

Lem. O. Ejiogu, Softmetrix, Inc. Chicago (USA)

Although many practitioners continue to expound KLOCs as a metric of software size, the popular consensus is that it does not address the well-known problems of software management: analysis, design, testing, maintenance, documentaion; and software certification. A principled approach built on sound theoretical foundation is critical for measuring software size. Every science has an underlying body of principles. The measurement of software size can greatly benefit from a formal component theory. One such methodology is briefly introduced below. Because of space, the reader is referred to the workshop book for its rich feedback effects and sample worked examples.

MULTIDIMENSIONAL SOFTWARE PERFORMANCE MEASUREMENT MODELS: A TETRAHEDRON-BASED DESIGN

Luigi Buglione, Università di Roma (Italy) and
Alain Abran, Université du Québec à Montréal (Canada)

This work addresses the growing importance for management to have tools at their disposal for performance measurement of company resources, in particular of software.
This work presents an improved version of an open model of performance, called QEST (Quality factor + Economic, Social and Technical dimensions), which suitably integrates into a single representation both a quantitative and a qualitative measurement from three distinctive but connected dimensions, each of them representing a distinct viewpoint of performance:

- economic dimension, the perspective is the managers' viewpoint, with particular attention paid to cost and schedule drivers;
- social dimension, the perspective is the users' viewpoint, with particular attention paid to the quality in use drivers;
- technical dimension, the perspective is the developers' viewpoint, with particular attention paid to technical quality.

The qualitative measurement is based on the ISO/IEC 9126 standard, expressed by a quality factor. The implementation of the model follows a 10-step procedure. This model allows extensive use of ISO and industry de facto standards-based measures.

A PASTRY COOK'S VIEW ON SOFTWARE MEASUREMENT

Frank Niessink and Hans van Vliet, Vrije Universiteit of Amsterdam (The Netherlands)

Many frameworks for implementing software measurement exist, ranging from collections of success factors to maturity growth models. One may ask to what extent these guidelines increase the chance of a successful measurement program. To aid in answering this question, we introduce a generic process model for measurement-based improvement. We use this model as a reference model to compare a number of existing software measurement implementation frameworks. From these assessments we conclude that the guidelines given by these frameworks provide a considerable amount of support for the basic activities needed to implement measurement programs. However, we also observe that the guidelines hardly provide any guidance to guarantee successful usage of the measurement program.

METRIC FOR EFFECTIVE TEST COVERAGE

Vedha Kichenamourty, University of Marne la Vallée, France

This paper highlights the concept of a software testing metric designed to improve the software testing effectiveness. The proposed metric Metric for Effective Test Coverage is a number of tests metric which can be defined as a ratio between the number of test cases developed (x) and the number of test cases to be developed (y). The quantity y represents the

actual test domain which gives full confidence in achieving the objectives of software testing. The metric focuses on a method to obtain a comparatively attractive value for y, a test pattern which will insure that all the pair wise combinations of a given set of selection have been exercised. The method builds on the concept of Orthogonal Arrays(OA), a mathematical tool developed by Dr. Genichi Taguichi, which helps to wisely determine the test cases from the large test suite.

MEASURING LEGACY DATABASE STRUCTURE

Harry M. Sneed and Oliver Foshag, Software-Engineering Service, Ottobrunn/Munich (Germany)

Metrics for databases have been neglected in the metric community. On the other hand, there is a great need to measure the size, complexity and quality of legacy data bases, in particular in regard to conversion and reuse. This paper presents a set of metrics for doing this. These metrics have been build into a tool for measuring database structures and have been applied to assess user applications in accordance with the ISO Standard 9126 for product evaluation.

REST - A TOOL TO MEASURE THE RIPPLE EFFECT OF C AND C++ PROGRAMS

Sue Black, South Bank University London (UK)

This paper describes the reformulation and subsequent implementation of the ripple effect measure first proposed by Yau and Collofello. Ripple effect traces the paths of variables through a program, providing a compound measurement of the effect that one module has upon the other modules. It may be used during software development to compare the stability of subsequent versions of a program, or during software maintenance to decide which modules within a program may need reengineering. The implementation of this reformulated measure REST (Ripple Effect and Stability Tool) gives ripple effect measurements for each individual module within a program and an overall stability measure: the reciprocal of the summed ripple effect for the program. During the reformulation of this measure using matrix algebra it was noticed that the computation algorithm for a certain matrix, D, which describes definition / use pairings, could be approximated, eliminating the need to use control-flow information. The simplified version of matrix D requires less memory and less time to compute. A previous implementation of the ripple effect measure suffered from slow computation times. REST has been tested on two versions of a mutation testing software tool and the approximate results compared with the ripple effect measure produced using the original matrix D. Initial results show that the simplified matrix D is a valid alternative to the original matrix D and as such the reformulation gives an acceptable approximation to the original measure.

Y2K FROM A METRICS POINT OF VIEW

Reiner R. Dumke and Achim S. Winkler, University of Magdeburg, Government Financial Computer Center Magdeburg (Germany)

The year 2000 problem (Y2K) is “not just a technical problem, it is a worldwide business problem affecting people and organizations everywhere”. The complexity of this problem and its solution can be resolved succesfully only on a measurement-based software development or maintenance environment.
In this paper we analyse the “measurement situation” on the Y2K area and consider the possibilities of the application of the existing results on the field of software measurement. In order to present the measurement aspects in a systematic manner, we use our measurement framework that leeds to a persistent metrics program in a given software development or application environment.

Software Metrics for Multimedia Languages

Stephen H. Edwards, Sallie M. Henry, and Roger P. Bodnar, Jr.,
Virginia Tech, Blacksburg (USA)

Software engineering researchers have attempted to improve the software development process for over two decades. A primary thrust in this process lies in the arena of measurement. “You can’t control what you can’t measure”. This research applies software metric techniques to the development of multimedia products. Problem areas such as education, instruction, training, and information systems can all benefit from more controlled approaches to development with multimedia tools. As an example, we focus on one multimedia language for creating multimedia products: Macromedia’s Authorware. This paper describes the measurement of various distinguishing properties of this language, together with an evaluation of the measurement process. The evaluation gives insight into the next step in establishing the goal of control, through measurement, of the multimedia software development process.

Improving Reliability of Large Software Systems

Christof Ebert, Thomas Liedtke, Ekkehard Baisch,
Alcatel Telecom, Antwerp (Belgium) / Stuttgart (Germany)

Improving field performance of telecommunication systems is one of the most relevant targets of both telecom suppliers and operators, as an increasing amount of business critical systems worldwide are relying on dependable telecommunication. Finding defects earlier of course should improve field performance in terms of reduced field failure rates and reduced intrinsic downtime. This paper describes an integrated approach to improve early defect detection and thus field reliability of telecommunication switching systems. The assumptions at the start of the projects discussed in this paper are: Wide application of code inspections and thorough module testing must lead to a lower fault detection density in following test phases. At the same time criteria for selecting the right modules for code reviews, code inspections and module test have to be improved in order to optimize efficiency. Experiences from projects of Alcatel Telecom's Switching System Division are included to show practical impacts.

Prototype of a Software Metrics Database for industrial use

Andreas Schmietendorf, Deutsche Telekom Development Center Berlin (Germany)

The suitability of using metrics in software projects is now generally acknowledged. There is still, however, a widespread lack of confidence in the interpretation of object-oriented metrics, in concentration on use of only a few metrics and in the processing of large sets of measured values. While in the scientific field a large number of different metrics are generally used, the development of industrial software requires just a few meaningful metrics that can be applied with a minimum of effort. This is exactly the background to the "metricDB" project, whose objective is to collect a few significant, mainly object-oriented metrics in a database, largely automatically, in order to enable qualitative statements to be made and effort to be estimated. There must be an extremely easy-to-use interface (Web client) for evaluation purposes, as well as the possibility of complex, highly flexible evaluations, such as is the case, for example, under SPSS or Excel.
The "metricDB" project was launched at the Development Center in Berlin this May. The findings that are described below are principally based on the predefined requirements, on our experiences with the metrics tools used and with a first prototype that is now complete. They also include the results of workshops held jointly with the software measurement laboratory at the Otto-von-Guericke university and a group at France Telekom engaged in research in this field.

COMPARISON BETWEEN FPA AND FFP: A FIELD EXPERIENCE

Jean-Marc Desharnais, SELAM Québec (Canada) and Pam Morris, Total Metrics (Australia)

A requirement for software productivity analysis and estimation is the ability to measure the size of a software product from a functional perspective rather than from a technical perspective. One example of such a functional size measurement (FSM) technique is Function Point Analysis (FPA). FPA is now widely used in the MIS domain, where it has become the 'de facto' standard in the industry. However, FPA has not had the same acceptance in other domains, such as real-time software. A new technique was proposed by the University of Quebec in Montreal (UQAM) and the Software Engineering Laboratory in Applied Metrics (SELAM) to extend the functional measure to real-time software. The new technique is called Full Function Points.
This article reports on a practical research undertaken in a telecommunications company which compared 5 major applications in the real-time and MIS domains when measured using both FPA and an proposed extension, called Full Function Points (FFP). This article presents the main FPA and FFP concepts, the description of the types of software, the methodology used in the field tests and the results of field tests. All of the applications are multidimensional and three of them demonstrate the characteristics of real-time software.
The results showed that, when applying the FFP technique for applications at the business user level, both techniques measured an equivalent functional size in term of number of points for MIS applications. However this was not the case, for some of the layered software used within MIS applications.
For applications within the MIS domain, the FFP technique was found to be as effective as the FPA technique, suggesting that it could be of benefit if used within this domain. For real-time applications, the difference in the size measured between the two techniques was significant. This was expected for the real-time applications since the FFP technique was designed to ensure that the functional characteristics of real-time are taken into account when measuring this type of software.

FUNCTION POINT COUNTS DERIVED FROM SAP BUSINESS SCENARIO REQUIREMENTS

Maya Daneva, Clearnet Communications Inc., Ontario (Canada)

Recent years have witnessed a very high level of interest in the SAP technology and the Function Points Analysis (FPA). The motivation for implementing the SAP R/3 business application suite is twofold. On one side, SAP provides the information technology infrastructure for business process re-engineering and a component-based concept for R/3 implementation which benefits the most from software reusability. On the other side, SAP has protected its customers from the Year 2000 problem and, in addition, has brought internet capabilities to the R/3 application suite. Simultaneously, Function Points (FP) have been becoming widely accepted as the standard metric for software controlling and planning. Given the heightened attention on both SAP technology and FPA, it is surprising that little research has been conducted to examine the linkages between these two concepts in depth and to integrate them properly.
This paper presents how FPA fits with the SAP technology. Specifically, we explore the use of Function Points to help with SAP business scenario requirements. Our work proposes an approach to deriving Function Points from SAP scenario process models and business object models. The paper discusses some size and complexity attributes pertinent to the SAP business requirements and suggests a set of components to be counted together with a set of rules which specify how to obtain the SAP size and complexity numbers.

INFORMATION REGARDING WOSP 98

"FIRST INTERNATIONAL WORKSHOP ON SOFTWARE AND PERFORMANCE 1998"

Andreas Schmietendorf, Senior Consultant for System and Software Development
Deutsche Telekom AG, Development Center Berlin (Germany)

The topics treated here encapsulate the work carried out by the workshop. They do not necessarily represent a consensus arrived at by all participants. Official information is available at the following workshop web site:
URL: http://www.sce.carleton.ca/wosp98

Abstract

The first international workshop on "Software Performance Engineering" – known hereafter as SPE – was held between October 12 and 16, 1998 in Santa Fe/New Mexico (USA). **DURING THE WORKSHOP, [11] EXPLAINED IN HER CONTRIBUTION THE STATISTICAL DATA RELATING TO THE CAUSES OF FAILURE OF PROJECTS. AFTER MANAGEMENT AND SPECIFICATION PROBLEMS, FACTORS RELATED TO THE UNSATISFACTORY PERFORMANCE OF AN**

INFORMATION SYSTEM IN TERMS OF RESPONSE TIMES AND THROUGHPUT WERE THIRD ON THE LIST OF FAILURE STATISTICS. THERE ARE EXAMPLES OF PROJECTS IN WHICH PERFORMANCE PROBLEMS DUE TO INADEQUATE RESPONSE TIME AND THROUGHPUT BEHAVIOR HAVE CAUSED DAMAGE TO THE TUNE OF MILLIONS OF DOLLARS.

The most common practise is, however, that performance characteristics of information systems are usually not considered until the latter stages of development. SPE intends to change this situation by proposing methods that allow performance aspects to be considered as early as during the design of a new applications as part of the software engineering process. SPE aims to assist in upholding defined requirements pertaining to performance in the development of an appropriate system concept (consisting of hardware, software and networks) for subsequent normal operation of the application using specified load profiles under consideration of cost factors.

Kazman et al. postulate in this respect, for example, that the quality attributes of large information systems depend in principle on the system architecture of the software. In systems of this kind, quality criteria such as performance, availability and modifiability are governed more by the overall software architecture than by special aspects of coding – e.g., the choice of programming language, special algorithms or data structures.

A number of parallel tutorials were held on the first day of the workshop – e.g., by [4] on the topic of "Unified Modelling Language and Performance Engineering" and [9] on "Performance Engineering Evaluation of CORBA-based Distributed System Architectures". The presentations and lectures held on the following days dealt with case studies on performance-related topics, presentation and comparison of methods of performance analysis, the industry's practical experience, performance models of software systems, the dependencies between architectural variants and performance, and the implementation of performance and work load measurements. Along with these lectures, the following key topics were also defined and discussed in depth in various forums and workgroups formed as part of the workshop's agenda.

- How can software engineering and SPE be integrated?
- How can the practical application of SPE methods be improved?
- Specification of main research aspects regarding software and performance.

AN EXTENSIVE, PRIORITIZED LIST OF TOPICS AND FUNCTIONS REGARDING THE ABOVE-MENTIONED POINTS WAS COMPILED DURING THE WORKSHOP. THE GENERAL CONSENSUS AMONG THE WORKSHOP PARTICIPANTS WAS THAT SUCCESSIVE PROCESSING OF THE POINTS IS CRUCIAL FOR THE SUCCESSFUL APPLICATION AND INTEGRATION OF SPE IN SOFTWARE ENGINEERING.

Introduction

SPE was established by Connie Smith; the publication of her "Performance Engineering of Software Systems" in 1990 represented a milestone in software engineering. By considering the life cycle of an information system, the phases of specification, requirements analysis, design for architectural specification, implementation (programming), testing and acceptance, launching and normal operation can be roughly distinguished. However, the performance of an information system in terms of response time and throughput is often not regarded until the testing and acceptance phase – i.e., at the end of the actual engineering process. Another typical occurrence is the measurement of performance metrics during normal operation of an application for appropriate tuning actions or specification of system extensions.

Since the performance of an information system depends on the complex interrelationships between all hardware and software layers that contribute to the performance, the consideration of performance aspects at the end of an implementation alone cannot be deemed to be reasonable. On the contrary, experiments have shown that performance is especially influenced by architecture – i.e., dependent on the design of the application. SPE attempts to realize systematic development and planning of requisite systems during – and, in particular, integrated in – the software engineering process.

A change from quantitative to qualitative software development can be recognized in the development of software for essential company applications. The objective is therefore to incorporate quality requirements – e.g., performance – in the development of software systems from the very beginning. A methodic, reproducible procedure based on practical experience is recommended. The procedure should be capable of integration into the existing procedural models for software engineering and into concrete tools for modeling and implementing software systems. Moreover, the use of a frequently selected prototype procedure within the object-oriented software development environment and the reuse of classes, or even components, enable performance criteria to be verified on the basis of the decomposition principle at early phases of application production.

In addition, large software projects require cost estimates for the requisite system environment at an early stage. The question posed at this time is what basis can be used for estimating a system concept (resources planning). The objective should be to use standardized methods that permit the necessary hardware resources to be estimated, similar to the estimate for software developments – e.g., in function points.

SPE undergoing radical changes

SINCE THE PUBLICATION OF THE BOOK BY [8] ON THE TOPIC OF SPE IN 1990, NO CRITICAL BREAKTHROUGH WITH REGARD TO THE INTEGRATION IN INDUSTRIAL SOFTWARE ENGINEERING HAS YET BEEN ACHIEVED. THE AUTHOR INITIATED A SURVEY ON THE CAUSES OF THE LOW ACCEPTANCE OF SPE AMONG 69 SOFTWARE PERFORMANCE ANALYSTS. THE RESULTS ARE AS SUMMARIZED BELOW:

- 41%: little or no familiarity of the SW developers with SPE methods
- 32%: time pressure given as reason for the low use of SPE
- 16%: lack of performance analyses for and experience with available systems
- 4%: only a small number of tools available for SPE support
- 3%: SPE costs during development
- 4%: other problems.

The results indicate that the most common causes are to be found in the lack of familiarity with the methods for performance analysis. In my view, the causes are due – along with the limited distribution of SPE – to the underdevelopment of the available methods and tools. Usually, performance analyses during the early software engineering phases are associated with complex mathematical or simulative methods that can only be applied to powerful industrial software systems with great difficulty.

Another significant factor is the time pressure that often exists on industrial software applications, as well as a lack of information regarding the performance of information systems in operation in the software engineering process. Of great surprise is the fact that the cost aspect is regarded as relatively unimportant. According to analysis by Capers Johnes, the costs for SPE as part of a development project account for about 3% of the overall budget. In contrast, the costs resulting from the unsatisfactory performance of an application can often reach millions.

Nonetheless – in the unanimous opinion of all the workshop participants – the performance of systems in the area of distributed applications (for example, Java/RMI, CORBA) is influenced for the most part by the architecture – i.e., by the application design. While it was conceivable in the past to resolve design faults during implementation or operation by means of suitable tuning or extension actions, this will hardly still be possible for complex distributed applications based on CORBA, for example, and will generally result in costly redesigns.

Methods of performance assessment at a glance

It is useful to outline the methods of assessing performance for the purpose of presenting the remaining topics. In this regard, I would like to refer to a presentation held during the CMG Annual Conference in Berlin in 1998 on the same topic. [6]

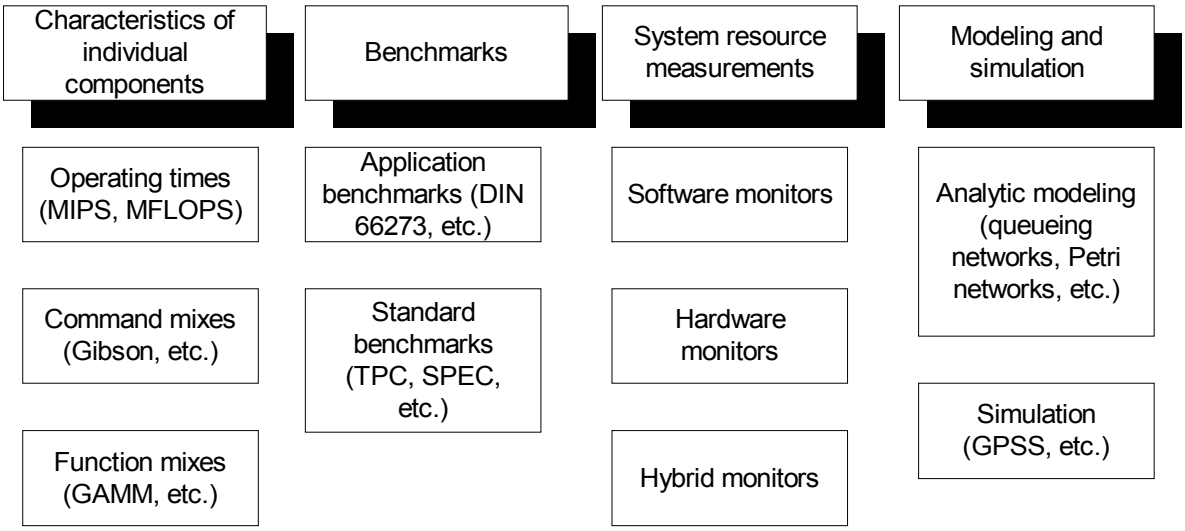


Fig. 1: Methods of performance assessment in overview

Fig. 1 illustrates the conventional methods of assessing performance. Far less detailed methods are often selected with the consequence that a distinction is made only between measurement methods and modeling. Whereas the measurement methods indicate the requirements of a real system, the modeling and simulation procedures do not. This means that – during the life cycle of an information system – only modeling and simulation methods can be used in the early phases of software development because no real system is yet available.

In my opinion, practical-minded users can estimate – in addition to the methods outlined –the hardware resource requirements for applications by means of rules of thumb in combination with values derived from standard benchmarks (e.g., TPC-C). "Rules of thumb" are based on analogous conclusions derived from similar information systems already in operation and on performance values obtained in the course of practical experience with the system. Typical rules of thumb exist for estimating database server resources, but less so for new technologies such as Internet- and CORBA[1]-based applications. In this way, general estimates for computer systems are possible. However, the response time behavior of actual application functions cannot be precisely determined. [5]

The conventional methods of performance assessment can only be used in the early phases of software engineering with difficulty. The reason for this is the non-utilization of performance models (e.g., queueing models, Petri networks) within the context of conventional software engineering. The workshop clearly found out, on the one hand, that developers of software applications must learn to use SPE methods but, on the other hand, that tools are also required for rendering the complex performance models calculable and for automatic derivation of the models from the software models (e.g., dynamic UML model).

Integration Software Engineering and SPE

[1] Common Object Request Broker Architecture

One of the workshop's focal points of interest were object-oriented systems. For this reason, these systems are regarded here only in regard to SPE. UML notation for modeling subsequent performance in the design phase is being used in ever greater measure in object-oriented software engineering. The suitability of UML for imaging components critical to performance was heatedly discussed during the workshop.

The manual derivation of related queueing models from UML-compliant diagrams for modeling the dynamic behavior of information systems is treated in [4]. The performance of the subsequent information systems is obtained by then subjecting the queueing models to simulation (for example, discrete simulation with GPSS[2], CSIM[3] or PDQ[4]), operational analysis or analytical examination using theoretical model procedures.

Yet another procedure – conditional to the iterative development particularly common to object-oriented software systems – refers to the use of early prototypes for analyzing performance aspects. Model variables are derived from the implemented prototypes on the basis of performance measurements. These variables are then in turn themselves modeled – e.g., using queueing models. This refers in particular to the representation of the prototype for subsequent application, a detailed load description (technical usage profile of the application), the definition of the resource requirements and the specification of requirements regarding the performance that is decisive in determining a suitable use of models created in this manner.

In addition to these procedures, which are aligned according to the conventional software engineering process, performance analyses of various HW and SW methods (for example, embedded realtime systems, intelligent networks) were also presented. Also of interest were treatments of methods of performance analysis using colored Petri networks [12] – for example, for the development of software architecture for mobile telephones, and the application of SDL/MSC: "Specification and Description Language and Message Sequence Charts" [3].

In summary, it can be stated that a major point of interest of continued scientific topics will be to automate the derivation of performance models from software engineering models as much as possible so as to achieve appropriate acceptance within industrial software engineering. The procedural models currently used for software engineering must also be supplemented by SPE aspects.

Tools for SPE support

The relationships treated in the preceding section demonstrate that it is quite possible to convert software engineering models to corresponding performance engineering models. Nonetheless, industrial applications require the support of tools because, on the one hand, there is insufficient time for manual resolution of complex queueing networks using, for example, the queueing theory and, on the other hand, the variety of models used in software engineering should be limited for reasons of consistency. What follows is a list of the most

[2] GPSS General Purpose Simulation System (programming language for discrete simulation)

[3] CSIM C/C++ simulation system (process-oriented, discrete event simulation models)

[4] PDQ (Pretty Damn Quick) analyzer (C-programming system for resolution of queueing systems)

contemporary and widely available tools for performance modeling whose suitability for SPE was also discussed or presented during the workshop.

SPE-ED: A performance model (execution graphs) is developed on the basis of the SW specification and dynamic UML models (use cases, message sequence charts: correspond in some cases to the UML sequence charts, etc.), estimates of the load conditions of new applications, experiences with prototypes or similar systems, benchmarks, etc. Using SPE-ED, the graphs can be further analyzed with regard to their performance-related attributes (best and worst case analyses, average values, hot spot analysis, etc.). [10]

SES workbench: A generic simulation environment for the performance analysis of hardware, software and networks based on discrete event-oriented simulation. The simulation environment permits interactive creation of reusable queueing models. The SES strategizer, based on the SES workbench, is provided. It contains actual hardware and software systems, thereby allowing special client/server environments to be modeled relatively easily as well as permitting simulation using the effective application load to be defined. [7]

ObjectTime: An engineering environment was presented by [2] with ObjectTime. It is based on the LQN (Layered Queueing Network) method and enables performance analyses to be carried out during the design of software. Performance models are created from previous design decisions or estimates, a derived execution cycle (sequence of actual scenarios for the application) and performance measurements/standard benchmarks for the target environment (e.g., servers and networks). It should be particularly noted that software developers only have a perspective of the conventional model elements of software engineering – i.e., they are not confronted with special performance models, but can still check the performance of an application's functions.

BEST/1: Enables monitoring of resource utilization of running applications at process level. Along with long-term analyses (e.g., recording of load profiles), performance estimates for altered load conditions and for modified configurations of hardware systems can be made using a 'predict' component. The performance of the hardware systems is described in terms of internal system characteristics or relevant SPEC[5] benchmarks. The Deutsche Telekom is currently examining the 'predict' component of BEST/1 as to its suitability for use in software engineering. [1]

Workshop findings

The Software Performance Engineering workshop had the three objectives described below. Separate workgroups were established to pursue the individual objectives. The major topics of interest, as I see them, are briefly outlined. All aspects defined in the course of brainstorming sessions and work groups cannot be completely represented here.

1. How can software engineering and SPE methods be integrated?

[5] SPEC Standard Performance Evaluation Cooperation

- Software engineering process model that includes software engineering and SPE
- Transparency of costs for SPE and losses due to performance problems
- Standard specification language for software engineering and SPE
- Consistency between software engineering and SPE models
- Tools for supporting SPE and exchanging information with software engineering tools
- Presentation of a schedule for application of SPE methods
- Performance information for software engineering obtained from operational software systems
- The granularity of SPE methods is required but costs must nevertheless be minimized.

2. How can SPE methods be made more practical to use?

- Performance specifications for finished software components
- Provision of simple methods which are consequently suitable for industrial application
- Opportunities for specialization as or training to become performance engineers
- Creation of databases for performance metrics based on practical experience
- Standard format for exchanging performance metrics between different tools.

3. Definition of research focal points regarding software and performance.

- Automatic generation of performance models from software models
- Determination of relationships between conventional SW metrics and performance metrics
- Development of a performance API for application in performance management
- Increased modeling of I/O subsystems (network connection, bulk storage, etc.).

Summary and outlook

The findings of the workshop indicate that a solitary consideration of the performance in terms of hardware or software alone will no longer be sufficient to constitute a solution in view of the increasingly complex nature of – in particular – distributed software systems. Emphasis should be placed on the early control and verification of design decisions for software systems undergoing development regarding their performance by using analytic or simulative methods. The approaches to SPE presented by the workshop are still in need of further refinement for industrial application. In my view, different yet accurate SPE methods related to the phases of software engineering and performance requirements (risk of a non-performatory system) of an information system are particularly necessary. The objective in this respect must be to define steps as part of a procedural model for software engineering that

allow a low-cost SPE method to be selected in accordance with the actual functionality required.

Benchmarks were not treated in any great detail in this report. The reason for this is that benchmarking during software production can only be performed using prototypes and that the use of performance analysis models during the design phase of a software system would appear to be more suitable. Nonetheless, benchmarks as part of performance tests for the subsequent information system are an important instrument for determining load limits (e.g., maximum throughput in compliance with specified response times) of an application. Standard benchmarks for generating model variables are likewise necessary, particularly for determining the performance characteristics of hardware and software systems. It should be noted, however, that standard benchmarks provided by independent organizations – e.g., universities or respected benchmark organizations such as SPEC, TPC[6] and AIM[7], are implemented and certified. The principle objective is to use standard benchmarks whose performance indicators have real relevance to one's own application, and are not just useful for marketing purposes.

Despite the workshop there will be no radical alteration in the approach toward industrial software engineering. However, a greater level of awareness and responsiveness concerning the subject should exist as a consequence of the widespread publication of the findings and objectives of the workshop. For example, there are plans to formulate a requirement for OMGs with regard to the UML in order to include performance-related problem aspects in future versions with the support of the UML.

It is my view that performance is only a qualitative objective as defined under "Efficiency" in the ISO 9126. Other aspects, such as availability, maintainability, etc. must likewise be considered in software engineering. If the initial step is the integration of software performance engineering and software engineering, subsequent steps will be an enhancement of an extensive systems engineering process that can also take other quality aspects into account, thereby permitting software production controlled by measurement values (software metrics).

References

[1] BMC Software, Inc. Waltham, MA.: Informationen zum BEST/1 System im Internet (Information regarding the BEST/1 System in the Internet), URL: http://www.bgs.com/ 1998

[2] Curtis Hrischuk et al.: A Wideband Approach to Integrating Performance Prediction into a Software Design Enviroment; Proceeding WOSP 1998, Carleton University Ottawa and ObjecTime Limited Ottawa

[3] Andreas Mitschele-Thiel, Bruno Müller-Clostermann: Performance Engineering of SDL/MSC Systems, Tutorial WOSP 1998

[6] TPC Transaction Processing Council

[7] AIM Independent performance certification using proprietary Benchmark suites

[4] Rob Pooley: Unified Modeling Language and Software Performance Engineering, Tutorial WOSP 1998, University of Edinburgh

[5] Schmietendorf, A.: Application of Empirical Values for System Dimensioning. MBB Notifications 33 of GI Technical Group 3.2.1, Spring 1998

[6] Schmietendorf, A.: System Concept Development in the Object-oriented Software Engineering Phases for Client/Server Systems. Presentation at the 11the CECMG Annual Convention, Berlin, April 22, 1998

[7] Scientific and Engineering Software, Inc.: URL: http.//www.ses.com, Austin Texas USA 1998

[8] Connie U. Smith: Performance Engineering of Software Systems, Addison-Wesley Publishing Co., New York 1990

[9] Connie U. Smith, Lloyd G. Williams: Performance Engineering Evaluation of CORBA-based Distributed System Architectures; Tutorial WOSP 1998

[10] L&S Computer Technology, Inc.: URL: http.//www.perfeng.com/~cusmith, Austin Texas USA 1998

[11] Elaine J. Weyuker, Filippos I. Vokolos: Performance Testing of Software Systems; Proceeding WOSP 1998, AT&T Labs

[12] Jianli Xu, Juha Kuusela: Modeling Execution Architecture of Software System Using Colored Petri Nets; Proceeding WOSP 1998, Nokia Research Center

FUNCTION POINT PROGNOSIS

Manfred Bundschuh (CNV AG), CFPS, Vice President of DASMA (Germany)

1 Environment

Within 1996 and 1997 about 20 Function Point counts with totally 26,575 Function Points (FP's) have been accomplished in CNV AG. FP's mean througout this paper IFPUG 4.0 unadjusted FP's. Function Point counts are obligatory at least at the end of the Requirements Analysis, end of the Design Phase and at project post mortem. Small projects are only counted twice. (not at the end of the Design Phase). Function Point Prognosis (instead of count) is obligatory during the Feasibility study and at Project start.

CNV AG is the outsourced non - assurance administration of the AXA - COLONIA Insurance. The IT - Department belongs to it. It includes about 500 IT - Professionals with approximately 50 project leaders. The insurance branches deliver about 160 IT - coordinators supporting the IT - projects. IT - development is mostly host - based with COBOL / Generator - programming. There are about 500 IMS - databases and 2000 DB2 - tables, 1.6 Million IMS

- and 1.2 Million CICS - transactions per day. PC - projects are programming with Power++, a C++ Shell. Projects develop mostly interactive database administration systems for e.g. car or life insurance or claims management, within a very complex environment with centralized data bases for e.g. insurance partners, cash systems, document management etc.

2 Underlying Data

Fig. 1 (*see end of the report*) shows detailed informations for each FP count as being kept in our project register, giving the quantity of EI's, EO's, EQ's, ILF's and EIF's. If a project was counted repeatedly only the most actual count is shown in that table. Older counts are kept in a history table. There are sums of the quantity of EI's and EO's, of ILF's and EIF's which were needed for our research. IO means througout this paper the sum of EI and EO.

The idea for the research was to find out if there were hidden informations in that data collection. The results proved to be very productive.

Fig. 2 *(see end of the report*) shows the according FP's as well as sums for the FP's of the EI's and EO's; of the EI's, EO's and EQ's; of the ILF's and EIF's. The last two lines in this Table are for comparison with [8] and the Quick Estimate mode of Checkpoint, an Estimation tool from SPR in Burlington, MA.

3 First Results

It can easily be seen, that EO's dominate in CNV AG (39%) compared with the other two sources (22%-24%) in literature, whereas ILF's are of minor importance (18% vs. 24%, 43%, resp.). Because of this peculiarity one conclusion was not to use Checkpoint (EIF + ILF = 46% vs. 23 % in CNV) in Quick Estimate mode to estimate FP's. The reason for the minor importance of ILF's may be that CNV AG has many centralized databases which are wideley refereced by lots of applications.

The domination of the EO's is the reson for the strong correlation between IO's and the unadjusted FP's- the main result. Charley Tichenor [9] from IRS Information Systems found for IRS's the tax processing (batch) Software a similar strong correlation as we did with R^2 = 0.9483 for the prognosis of unadjusted Function Points counts from ILF's, instead of IO's. **Fig. 3** shows the differences between IRS (typical batch environment) and CNV (typical online environment).

---

[8] P. M. Morris (Total Metrics), J. M. Desharnais (SELAM): „Validation of the Function Point Counts", in: Metricviews, Summer 1996, p. 30

[9] Charly Tichenor: „The IRS Development and Application of the Internal LOGICAL FILE Model to estimate Function Point counts", in his presentation at he IFPUG Fall conference „Target 2000" in Scottsdale, Arizona, USA, Sep. 15 - 19, 1997, pp. 299 - 321,

ratio	Tichenor	mostly batch	CNV AG	mostly online
EI:ILF	0,3331	often an EI references numerous ILF's	2,664	an ILF is often referenced by more than 2 EI
EO:ILF	0,3903	normally, most EO's reference multiple ILF's	3,142	more than 3 EO's usually reference 1 ILF
EQ:ILF	0,011	in most cases, IRS tax processing software is batch	1,2	an ILF is sometimes referenced by more than 1EQ
EIF:ILF	0,097	normally our tax processing sofware does not use EIF's	0,404	most EIF's reference multiple ILF's

Fig. 3: Comparison with Tichenors ILF model

Another one of the first results of this data collection was, that the VAF in CNV AG is typically between 0.8 and 1.2 (Host and/or PC, with an average of 0.99) and 0.7 for migrations.

Fig. 1 and 2 were analysed with the EXCEL problem solver,

a) to find out how many FP's a „typical" EI, EO, EQ, ILF or EIF has in our enviroment. The results are shown in **Fig. 4** and can be used as a rule of thumb, if the EI's, EO's, EQ's, ILF's and EIF's of a count are not yet qualified as low, average or high, but are known in quantity, thus leading to a „*FP - Prognosis*". The values are shown for PC - Projects, Host - Projects and both.

	PC - Projects									Host - Projects									
	Quant.				Quant FP					Quant.				Quant FP					
	LOW	AVE.	HIGH		LOW	AVE.	HIGH	Σ2	Ave. FP's Σ2/Σ1	LOW	AVE.	HIGH		LOW	AVE.	HIGH	Σ2	Ave. FP's Σ2/Σ1	Ave. FP's PC&HOST
EI	277	83	157	**517**	831	332	942	**2105**	**4,1**	130	358	513	**1001**	390	1432	3078	**4900**	**4,9**	**4,6**
EO	81	150	179	**410**	324	750	1253	**2327**	**5,7**	308	611	503	**1422**	1232	3055	3521	**7808**	**5,5**	**5,5**
EQ	170	70	74	**314**	510	280	444	**1234**	**3,9**	93	142	161	**396**	279	568	966	**1813**	**4,6**	**4,3**
ILF	182	6	1	**189**	1274	60	15	**1349**	**7,1**	211	137	17	**365**	1477	1370	255	**3102**	**8,5**	**8,0**
EIF	32	1	2	**35**	160	7	20	**187**	**5,3**	90	13	21	**124**	450	91	210	**751**	**6,1**	**5,9**

Fig. 4: Typical CNV Function Points

SPR Function Points use the average classification for Function Point estimation. Instead of SPR Function Points we advise our project leaders to use our „Typical FP's classification".

Fig. 5 can be used to compare both SPR and CNV typical FP's (extracted from Fig. 4).

	Classification			
	Average	CNV Typical FP's		
	IFPUG FP	PC	HOST	PC&Host
EI	4	4,1	4,9	4,6
EO	5	5,7	5,5	5,5
EQ	4	3,9	4,6	4,3
ILF	10	7,1	8,5	8,0
EIF	7	5,3	6,1	5,9

Fig. 5: Typical FP's comparison

We tested this typical FP's by multiplying it with the quantity of EI's, EO's, EQ's, ILF's, EIF's and comparing the result with the unadjusted FP's of the counts. The error proved to be less than 26%. **Fig. 6** (see end of the report) shows the according error - table (PC, HOST, both, resp).

In Fig.6 we also compared the errors of our typical CNV FP'S with the application of the average IFPUG FP's (SPR Function Points) and found that our result was some better.

b) to find correlations between the quantities of the EI's, EO's, EQ's, ILF's or EIF's and the total Number of FP's for each count. For this reason regression analysis was performed. The correlation coefficients R^2 and their square root R are shown in **Fig. 7**.

	R^2									R								
	Quantitiy			FP´s unadjusted			SPR FP´s			Quantitiy			FP´s unadjusted			SPR FP´s		
	< 1200	> 1800	all	< 1200	> 1800	all	< 1200	> 1800	all	< 1200	> 1800	all	< 1200	> 1800	all	< 1200	> 1800	all
EI´s	0,28	0,11	0,77	0,30	0,51	0,86				0,53	0,33	0,88	0,55	0,71	0,93			
EO´s			0,84	0,90	0,44	0,85						0,92	0,95	0,66	0,92			
EQ´s			0,33	0,01	0,31	0,41						0,57	0,10	0,56	0,64			
EI´s + EO´s	0,95	0,71	0,95	0,95	0,90	0,97	0,91	0,70	0,93	0,97	0,84	0,97	0,97	0,95	0,98	0,95	0,84	0,96
EI´s + EO´s + EQ´s			0,94			0,96						0,97			0,98			
ILF´s	0,86	0,19	0,75	0,14	0,75	0,83				0,93	0,44	0,87	0,37	0,87	0,91			
EIF´s			0,05	0,34	0,84	0,05						0,22	0,58	0,92	0,22			
ILF´s + EIF´s			0,47	0,92	0,47	0,62						0,69	0,96	0,69	0,79			

R^2: Determination coefficient

$R^2 \geq 0{,}90$ bold surrounded

R: probability for correlation

Fig. 7: Determination coefficients

It can easily be seen that the usage of FP's instead of the quantity for the Prognosis gives a stronger correlation, but the higher effort for counting is not adequate to the higher precision. One should always keep in mind that estimation has to do with uncertainty per se.

Correlations were analysed with the lower and upper half of the data (< 1,200 ; >1,800 FP's ; resp.) and all counts (exept migrations, because they are built for obsolesence and are batch - systems). The quantity, the FP's and SPR FP's were examined.

4 Main Result

The result of the research was, that the sum of the quantities of EI's and EO's (IO's in our diction) is correlated with $R^2 \geq 0.95$ ($R \geq 0.97$) to the total amount of FP's of a count and can thus be used as a „rule of thumb" for the „Prognosis" of FP's when the FP's of EQ's , ILF's and EIF's are not known.

Fig. 8 shows the regression analysis for this outstanding result. We found a comparable verbal information in [10] later on.

The formula for our CNV FP- Prognosis (extracted from Fig. 8) is **FP = 7.3 IO's + 56**

[10] John E. Gaffney, Jr. from Corp. Software Productivity Consortium in Herudon, Virginia 22070 in his Presentation „A Simplified Function Point Measure" at the IFPUG 1994 Fall Conference, Oct. 19-21, 1994 in Salt Lake City, Utah, pp. 207 - 209

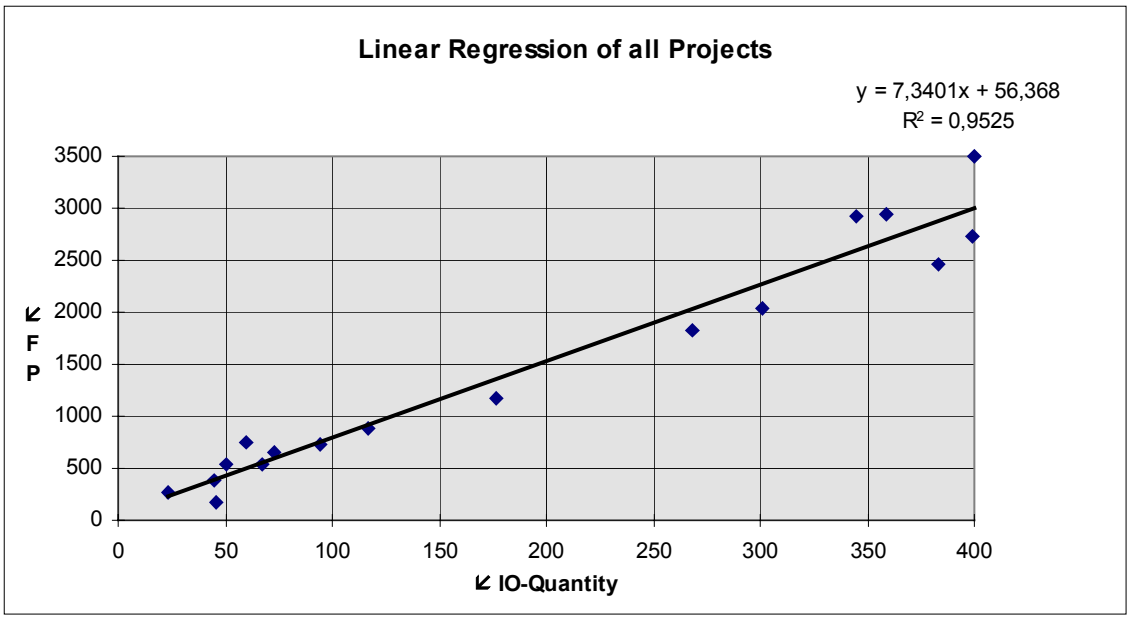


Fig. 8: Regression analysis of main result

5 Enhancement

Adding the last 3 counts to the Fig. 1 and 2 didn't change the coefficient R^2 . Hence the correlation proofs to be stable. Different parts of the dataset delivered often worse results. One can conclude from this, that at least about 15 counts are necessary to deliver a first set of profitable data for such research.

6 Error Discussion

Next we tested the result achieved by using only the quantity of EI's an EO's (summarized, =IO's in out diction) for each count, computing it with the regression formula and determing the differnce between this computation and the counted FP's. **Fig. 9** (see end of the report) shows this data (FP-Prognosis1) and leads to an error range of 12.11% (Median 10.6%).

Of course we tried the same research for parts of the data: Upper and lower half, PC - and Host - Projects (FP-Prognosis2), but no improvement for correlations was found. Both calculations were also performed with (rounded) integer coefficients and deliverd worse results.

One Project in the 1997 data collection was outstanding compared with the others for two reasons: The error was 35,6% and we soon found that it had one third of EI's and 3 times as much EQ's compared to the other projects. We learned that it was the only major information System compared to the other Systems which were all administration Systems.

7 Conclusion

Thus we recommend our project leaders our rule of thumb for FP - Prognosis (main result) with an error range of 15 - 20 % - but only for early FP - Prognosis. Of course a complete FP count at the end of the requirements analysis is obligatory.

Our findings compared with those from other firms show that such data collections can be used to find heuristic solutions for FP - Prognosis, either using „typical FP's" or regression formulae. But there is evidence that different environments demand for according solutions. Hence each firm should develop its own know - body of heuristic solutions and should distingnish between different development platforms etc. when doing so.

1998 all applications in CNV AG will be counted. Until June all groups will have at least one trained FP Counter. Hence we hope to get more reliable data and will keep this rule of thumb up to date. We will not count but estimate the FP'S of SAP, using this prognosis.

Astonishingly only few counts thoroughly recorded lead to an outstanding help for FP Prognosis in early project phases.

References

[1] **M. Morris, J. M. Desharnais:**
Validation of the Function Point Counts. In: Metricviews, Summer 1996, p. 30.

[2] **Charley Tichenor**, from IRS in Proceedings of the IFPUG Fall conference *Target 2000* Scottsdale, Arizona, USA, Sep. 15 - 19, 1997, pp 299 – 321.

[3] **John E. Gaffney, Jr.** from Corp. Software Productivity Consortium in Herudon, Virginia 22070 in his Presentation *A Simplified Function Point Measure* at the IFPUG 1994 Fall Conference, Oct. 19-21, 1994 in Salt Lake City, Utah.

MEASUREMENT OF STATIC SOFTWARE QUALITY

Salvatore Valenti, Alessandro Cucchiarelli, Maurizio Panti, University of Ancona (Italy)

GROWTH MODELS FOR SOFTWARE RELIABILITY

Dirk Schmelz and Julia Schmelz, Friedrich-Schiller-University of Jena (Germany)

Abstract

THERE ARE A LOT OF CRITICAL REMARKS ON SOFTWARE GROWTH MODELS, [2]. WE KNOW DOUBT ABOUT BOTH THE PREDICTION ABILITY OF FAILURE EVENTS AND THE EASY HANDLING FOR THE USER. FINALLY, THE UTILITY SHOULD GIVE THE LAST DECISION ABOUT THE USABILITY OF SUCH MODELS. OUR PAPER DEALS WITH WHITE BOX MODELS IN CONNECTION WITH INPUT EXPERIMENTS TO HELP BUILDING A BRIDGE TO PRACTICAL EXPLOITATION.

1 Software Reliability

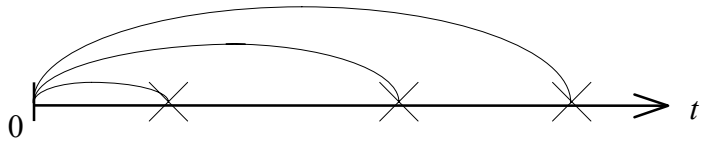
1.1 Concepts and Basic Context

A measure of system reliability is the probability that the software system does not fail. This probability describes the ability of the system to work correctly in a certain period. That is a realistic view, since even software developer exercise care on asserting existence of real flawless systems. Observations of system behavior are only meaningful in a time interval with sufficient size. The probability of correct working must be high enough for warranting a sufficient reliability of the program. The failure probability of the system is the difference of one and the mentioned probability. The length of a time interval without any failure is supposed as a real-valued random variable and it is often called life time of the program. The end of such an interval is a time point in which a failure event arrives. The choice of the suitable time basis used for measuring the life time of the program depends from the run conditions of the program. Measurement may be distorted by interruptions from the outside that carry off time. Therefore the measuring has to be independent from such influences by the reasonable choice of time basis. A counter of run number can be a suitable time basis in a few of applications. Then each run of a program or only a part of program is a time step. A measure of failure free time intervall is given by the number of all neighbouring failure free runs of program, so the geometrical distribution is the corresponding distribution of life time. However, a continuously distributed time interval length sometimes is necessary instead of a discrete time basis. Then the use of the exponential distribution is justified as life time distribution in the operational phase. The interpretation of life time as a real-valued random variable leads to quality statements that describe the statistical behaviour of the program in running time. Well-known examples are the expected number of failure events in a fixed time intervall and the expected length of a failure free time interval. A measure of increasing of program reliability is the increase of the probability for the failure free working of the program originating from decrease of the number of faults in the program. This increase of probability corresponds to a rise of the mathematical expectation of program life time. It corresponds to a reduce of the mathematical expectation of number of failure events in a fixed

time intervall. The process of repeated reducing of fault number in the programm is mentioned as growth process of reliability in the literature. The growth process is accompanied with an instationary behavior of the life time with jumps of the failure level in the failure points. That means that the mathematical expectation of the number of failure events in a fixed interval goes down steplike. The existence of faults in the program is the reason for the appearance of failures. The elimination of faults leads to progressing of the reliability growth process.

1.2 Reliability Growth

The black box modeling of failure behavior of a program results in the analysis of a sequence of random time points. The sequence of random time points with failures is obviously a sequence of increasing, not negative real-valued random variables. The picture of a realisation could have the following shape:



Then the failure-free time intervals and the number of failure events in a fixed interval are real-valued random variables too. If the failure free time intervals are independent and identically distributed, the process will be often called renewal process in the strict sense. The most prominent example is the Poisson process. The number of failures in a given interval is Poisson distributed with the parameter λ, $\lambda>0$. The parameter λ, $\lambda>0$, thereby corresponds to the mathematical expection of failure in the interval (0,1). The failure-free intervals are independent and exponential distributed with the same parameter λ. This distribution causes that the process is without memory, so the length of the failure-free time interval after a given time point t is independent from the behavior of the process until t. The density of the life time is given as $p(t) = \lambda \exp(-\lambda t)$, $t \geq 0$, $\lambda > 0$.

The probability that the life time T has at most the length s can be expressed as

$$F(s) = P(T \leq s) = 1 - \exp(-\lambda s), \quad s \geq 0, \lambda > 0.$$

The survival probability has the value

$$R(s) = P(T > s) = \exp(-\lambda s), \; s \geq 0, \lambda > 0.$$

The Poisson process corresponds to a run without fault reducing. For some practical purposes a supposition is meaningful that the intensity decreases by fault detecting and fault elemination after a failure event. The reliability remains constant, for no fault is removed.

A large number of the well-known models carrys out peacewice constant and decreasing intensities by progressing time. The i-th surviving time has the intensity λ_i. The program will contain N + 1 - i faults in the i-th life time, if in the first interval N faults are supposed.

Other ideas exploit a function $\lambda(t)$ instead of λ_i to generate failure events. Finally, it was also tried to model the information of the failure data by the aid of ARIMA processes. A survey and comparisons of existing growth models is given in [8] and in [1].

2 White Box Modeling

Now we change our interpretation of a program system. We replace the black box concept by the white box concept. For this the source code is broken down into parts. The parts are called components. They represent executable parts of the system. Components also can be modules or subsystems composed by modules. The structure of the language often provides a basis definition of the concept of module.

Models using informations from the internal structure of the program system are called white box models. This models require in general a higher expenditure than black box models. They are able to regard interactions between components. In addtion, such models allow the involving of complexity of each module to evaluate the failure behavior of the system and parts of it. A high complexity of a modul often causes the fault-proneness and the failure of the system. Here the concept of black box is applied to each component of the program decomposition.

2.1 Modeling of Structure

A few papers as given in [4] have been published on structural software reliability modeling. Our presentation of the white box technique leans on this paper of Ledoux and Rubino. For this approach is assumed that the interaction of components only works by execution control transfer from any system component to another. A sequence of active components is a realisation of a random process. The states of the process correspond to the components. A transition of activity between two fixed components is checked by the call graph of the program structure. Stochasticity is given by the inputs arriving at the system randomly. The call graph is built as Markovian graph. The sojourn time in a component is interpreted as a real-valued random variable. The sojourn time distribution can be shaped in dependence on the realisation of the last sojourn location of the graph. The modeling will be not too expensive if we only regard the suspect model components or links between components. The model of program structure is called execution model. The mathematical presentation of the program execution process as a homogeneous time-continuous Markovian process X is characterized by

- a state space, denoted by $M = \{1, 2, ..., m\}$,
- an infinitesimal generator, denoted by $Q = (q(i,j)), \quad i,j \in M$,
- and an initial distribution, denoted by $\alpha = (\alpha_1, ... , \alpha_m)$.

The process X is assumed to be irreducible and decribes runs of program as abstraction.

2.2 Modeling of Failures

Ledoux and Rubino add the failures process to the execution model and obtain the operational model. The result is a bi-dimensional time-continuous process (N_t, X_t^+), where N_t counts the failures in interval (0,t]. The process X_t^+ will be determined by X , from section 2.1, if the following assumption are met :

(i) an active component i fails with a constant rate

(ii) an activity change from component i to component j fails with a constant interface failure probability.

Additionally, the authors distinguish between primary and secondary failure in both of cases (i) and (ii). Primary failure is followed by an execution break. Secondary failure is not attended by impacts on execution time. Failures within (i) are assumed to be independent. The same is assumed for failures inside (ii) and for pairs of failures of (i) and (ii). The construction of the Markovian process requires the notation of conditional probabilities :

(1) $P(X_{t+dt}^+ = j \text{ and } N_{t+dt} - N_t = 0 \mid X_t^+ = i \text{ and } N_t = k)$

(2) $P(X_{t+dt}^+ = j \text{ and } N_{t+dt} - Nt = 1 \mid X_t^+ = i \text{ and } N_t = k)$, $\quad i, j \in M.$

The division of terms (1) and (2) by dt and the transition $dt \to 0$ lead to the transition rates without and with failures denotet by a(i,j) and d(i,j), respectively, in [4]. It holds

(3) $\Sigma_j\ (a(i,j) + d(i,j)) = \Sigma_j\ q(i,j) + \lambda_i + \mu_i, \quad i = 1, ..., M.$

Assuming (i) the notations λ_i and μ_i stand for failure rates in cases of primary and secondary failures. Assuming (ii) the probabilities for the primary and the secondary case, respectively, are termed $\lambda(i,j)$ and $\mu(i,j)$. The elements a(i,j) and d(i,j) are formed in dependence on λ_i , μ_i , $\lambda(i,j)$, $\mu(i,j)$ and q(i,j). If we have a primary interface failure, there will be a jump from state i to another state j with the probability $\alpha(i,j)$. The formula

(4) $d(i,i) = (\lambda_i + \Sigma_{k \neq i}\ q(i,k)\ \lambda(i,k)\ )\ \alpha(i,i) + \mu_i, \quad i \in M,$

shows possible transitions in failure case without any change of state. The last facts prove that X_t^+ can be seen as a Markovian process being irreducible on M. The start distribution is α and the matrix

(5) $Q^+ = A + D$ is the infinitesimal generator of the process.

3 Demonstration Example

3.1 The Code Example

Now we describe an example demonstrating concepts and techniques given above. We observe failures caused by a SRC-Modula-3 freeware release [5], because we found some failures in arithmetical operations while we were using the above mentioned Modula-3 version in a computer course. For instance the addition of positive integers provides a negative result without any hint from the system. Hand-calculated we can detect that the sum is out of integer range. For example, the sum of integer'last and 1 constitutes ostensible

integer'first. In a failure observation program we perform the arithmetical operations addition (+), subtraction (-), multiplication (×) and division (/) working some data producing by a uniform pseudo random number generator in the following kind:

- At first we randomly choose one operation.
- The uniform pseudo random generator gives two random integers lying between an upper and a lower bound of a fixed range.
- The chosen operation provides a result exploiting the input.
- The next operation is determined in dependence on a result.

The four operations are here the components defined above. The interaction of the components in an observation program was performed by control transfer from the addition to the subtraction, from the subtraction to the multiplication, from the multiplication to the division, from the division to the addition and so on in cycles. But, if a wrong result appears, the activity of components will be not transfered and a corresponding failure counter will be increased by one.

3.2 Choice of Time Basis

A failure model has to use a time basis for length measuring of a time interval. Our example allows to choose a rigorous discrete time basis. We count one component execution as one time step. Then we can count the number of operations between two neighbouring failures to measure a life time interval. The simplified model is a time discrete Markovian chain. Another time discrete or time continuous model may be used different. But, we do not consider this.

3.3 Modeling of Failures

The run of chains is managed by a quadratic matrix of transition. Its entries decribe the possibilities of the state changes at one time step while the program execution and are called transition probabilities. This matrix holds for each time step.

The transition matrix has the form :

$$(6)\qquad \begin{array}{c|cccc} & + & - & \times & / \\ + & * & * & 0 & 0 \\ - & 0 & * & * & 0 \\ \times & 0 & 0 & * & * \\ / & * & 0 & 0 & * \end{array}$$

for our program example.

For instance, the transition from / to × has the entry zero.

Symbol * means a not negative probability value to transfer the activity from one component to another one. If a one-step-connection of two different components does not exist in the call graph of the program, a zero entry will be given to the matrix element corresponding to these components. Summation of entry values of a row must be equal to one. Entries situated in the diagonal represent the failure behavior of the Modula-3-system with respect to the four arithmetical operations. A zero in the diagonal means that the corresponding component is without any failure. The failure structure (6) is different from structure (7) and also from (8) :

$$(7)\quad \begin{array}{c} \\ + \\ - \\ \times \\ / \end{array} \begin{array}{c} \begin{array}{cccc} + & - & \times & / \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & * & * & 0 \\ 0 & 0 & * & * \\ * & 0 & 0 & * \end{pmatrix} \end{array} \quad , \quad (8)\quad \begin{array}{c} \\ + \\ - \\ \times \\ / \end{array} \begin{array}{c} \begin{array}{cccc} + & - & \times & / \end{array} \\ \begin{pmatrix} 0 & * & * & 0 \\ 0 & * & * & 0 \\ 0 & 0 & 0 & 1 \\ * & 0 & 0 & * \end{pmatrix} \end{array} .$$

In (7) and (8) the operation + is without any failure. But, the scheme (8) allows a program structure with a connection from + to × . The operation × may be without any failure in (8), for it uses the failure free operation + . The sequence of schemes shows removing of faults in the addition + , (6) → (7), in the multiplication × , (7) → (8).

In our experiments runs without any failures correspond to schemes with only zero entries in the diagonal.

The art to relate failures on the one side to programm structure and on the other side to functionality that is determined by input qualities in the operational phase can be supported by white box experiments.

3.4 Input Variation

Now it is time for an explanation how to get values from the program for each * entry within the transition matrix. A long run of the observation program gives frequencies of transitions being exploited as estimates for transition probabilities. These values characterize the distribution of used input data.

To demonstrate the influence of different input behavior, we see over the frequencies of two transition matrices below from a program structure (6) to compare each entry value of one matrix with the corresponding value of the other matrix. For each of matrices (9) and (10) we generated random numbers in the same integer range.

Numbers are chosen from

$$[-2^{31}+1, \ldots, 2^{31}-1] \text{ for operations } +, -, /$$

and

$$[-2^{17}, \ldots, 2^{17}] \text{ for operation } \times .$$

Entry values in experiments are given by

$$(9)\quad \begin{array}{c} \\ + \\ - \\ \times \\ / \end{array} \begin{array}{c} \begin{array}{cccc} + & - & \times & / \end{array} \\ \begin{pmatrix} .28 & .72 & 0 & 0 \\ 0 & .24 & .76 & 0 \\ 0 & 0 & .64 & .36 \\ 1 & 0 & 0 & 0 \end{pmatrix} \end{array} \qquad (10)\quad \begin{array}{c} \\ + \\ - \\ \times \\ / \end{array} \begin{array}{c} \begin{array}{cccc} + & - & \times & / \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & .27 & .73 & 0 \\ 0 & 0 & .57 & .43 \\ 1 & 0 & 0 & 0 \end{pmatrix} \end{array} .$$

In the first experiment we generated all operands at random and independently. We contained matrix (9). In a second input simulation (10) we used all correct operation results of the process to be the first operands of the next operation in the following step. That means in the process providing (10) the operation + seems to be flawless. But, in reality the failure of the operation + does not appear in consequence of the changed functionality of the program in the operation. This shows the impact of input quality on the failure probability. We note that the number of use the four operations are the same in experiments of (9) and (10).

4 Reliability in the Operational Phase

In this section we develop a principal view of software reliability. This emphasis to the operational phase throw light on the interplay between the reliability and the test phase.

We suppose that all the functions given in the specification of application were treated satisfactory in the test phase. A final test report has to contain an assignment of certain input ranges to particular functions. Now we assume that failure events arrive in the operational phase. The input quality used in a failure case reveals a test deficiency. Program faults detected in the operational phase were not found or were tolerated during the test phase. For instance the test phase will tolerate a fault, if the failure does probably not arrive or, if we can go with the damage size. The reason of failuring is given in the release used in the operational phase, so the resulting product only has insufficient quality. In a lot of situations we suppose that the possible failures can be explained with the nature of particular components or with interface failures. For this we go on with a low complexity of components being reasonable. Furthermore we assume that the submitted structure masters the required functionality. In section 3.4 we have demonstrated by an example, how the different use of a program system can lead to different reliability evaluations. So we have emphasized a dynamic effect, which is due to the variable data quality. This effect has to be regarded at determining the operational profile of a program enviroment. The concept of the dynamic complexity from Khoshgoftaar et. al. , [3] does not regard this aspect.

The connection of static complexity and operational profiles determines the dynamic complexity of a program system. The importance of static complexity information about these system parts and about interactions increases with the growing size of a program system. Complexity metrics can be used for the identification of system parts prone to failures. Additionally, such models offer possibilities to detect faults by input experiments of critical parts of system. Large parts of a system and also the whole system can be characterized by domain attributes called factors of complexity, [6], [7]. Complexity assessments of the system give important hints for the system decomposition in parts operating interactive.

White box modeling is a proved remedy in the operational phase, since is allows the simulation of many parts of the program system as an abstract enviroment of the critical components while experimental runs. White box models are able to connect the concept of the operational process, [4], with the concepts of the important profiles of the dynamical complexity: functional, execution and system component profile, [6]. White box models are flexible for adapting to different operational enviroments on different levels of abstraction.

References

[1] **Belli, F.; Grochtmann, G.; Jack, O.:**
Erprobte Modelle zur Quantifizierung der Softwarezuverlässigkeit. In: Informatik-spektrum, Band 21, Heft 3, 1998.

[2] **Butler, R.W.; Finelli, G.B.:**
The Infeasibility of Experimental Quantification of Live-Critical Software Reliability. ACM, 1991, 0-89791-455-4/91/0011/0066.

[3] **Khofgoshgoftaar, T.M.; Munson, J.C.; Lanning, D.L.:**
Dynamic System Complexity. IEEE 1993, 0-8186-3740-4/93.

[4] **Ledoux, J.; Rubino, G.:**
A Counting Model for Software Reliability Analysis. Rapport de Recherche n^0 2060 juillet 1993, Rennes, unité de recherche INRIA.

[5] *Modula3-Implementation auf DEC 5000/260, MIPS R4400 ultrix4.4*. Rechenzentrum der Fak. für Mathematik und Informatik, FSU Jena , Aug. 98.

[6] **Munson, J.C.:**
Software Faults, Software Failures and Software Reliability Modeling. IEEE : 1-4, 1994.

[7] **Schmelz, D.; Schmelz, M.:**
The Use of Factor Analysis in the Area of Software Metric. In: Software Metrics (F. Lehner, R. Dumke, A. Abran Eds.) Gabler - Verlag, Wiesbaden 97.

[8] **VDI-Verlag Düsseldorf 93:**
Softwarezuverlässigkeit, Grundlagen, Konstruktive Maßnahmen, Nachweisverfahren. Herausg. : VDI - Gemeinschaftsausschuß Industrielle Systemtechnik.

Growth Models for Software Reliability

Friedrich-Schiller-Universität Jena
Fakultät für Mathematik und Informatik
Dirk Schmelz Julia Schmelz
Ernst-Abbe-Platz 1-4
07740 Jena
Tel. +49 03641 9 46343
email: nis@inf.uni-jena.de nsj@inf.uni-jena.de

Abstract :

There are a lot of critical remarks on software growth models, [BuFi91]. We know doubt about both the prediction ability of failure events and the easy handling for the user. Finally, the utility should give the last decision about the usability of such models. Our paper deals with white box models in connection with input experiments to help building a bridge to practical exploitation.

1. Software Reliability

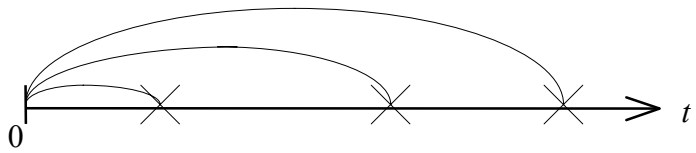
1.1 Concepts and Basic Context

A measure of system reliability is the probability that the software system does not fail. This probability describes the ability of the system to work correctly in a certain period. That is a realistic view, since even software developer exercise care on asserting existence of real flawless systems. Observations of system behavior are only meaningful in a time interval with sufficient size. The probability of correct working must be high enough for warranting a sufficient reliability of the program. The failure probability of the system is the difference of one and the mentioned probability. The length of a time interval without any failure is supposed as a real-valued random variable and it is often called life time of the program. The end of such an interval is a time point in which a failure event arrives. The choice of the suitable time basis used for measuring the life time of the program

depends from the run conditions of the program. Measurement may be distorted by interruptions from the outside that carry off time. Therefore the measuring has to be independent from such influences by the reasonable choice of time basis. A counter of run number can be a suitable time basis in a few of applications. Then each run of a program or only a part of program is a time step. A measure of failure free time intervall is given by the number of all neighbouring failure free runs of program, so the geometrical distribution is the corresponding distribution of life time. However, a continuously distributed time interval length sometimes is necessary instead of a discrete time basis. Then the use of the exponential distribution is justified as life time distribution in the operational phase. The interpretation of life time as a real-valued random variable leads to quality statements that describe the statistical behaviour of the program in running time. Well-known examples are the expected number of failure events in a fixed time intervall and the expected length of a failure free time interval. A measure of increasing of program reliability is the increase of the probability for the failure free working of the program originating from decrease of the number of faults in the program. This increase of probability corresponds to a rise of the mathematical expectation of program life time. It corresponds to a reduce of the mathematical expectation of number of failure events in a fixed time intervall. The process of repeated reducing of fault number in the programm is mentioned as growth process of reliability in the literature. The growth process is accompanied with an instationary behavior of the life time with jumps of the failure level in the failure points. That means that the mathematical expectation of the number of failure events in a fixed interval goes down steplike. The existence of faults in the program is the reason for the appearance of failures. The elimination of faults leads to progressing of the reliability growth process.

1.2. Reliability Growth

The black box modeling of failure behavior of a program results in the analysis of a sequence of random time points. The sequence of random time points with failures is obviously a sequence of increasing, not negative real-valued random variables. The picture of a realisation could have the following shape:



Then the failure-free time intervals and the number of failure events in a fixed interval are real-valued random variables too. If the failure free time intervals are independent and identically distributed, the process will be often called renewal process in the strict sense. The most prominent example is the Poisson process. The number of failures in a given interval is Poisson distributed with the parameter λ, $\lambda>0$. The parameter λ, $\lambda>0$, thereby corresponds to the mathematical expection of failure in the interval (0,1). The failure-free intervals are independent and exponential distributed with the same parameter λ. This distribution causes that the process is without memory, so the length of the failure-free time interval after a given time point t is independent from the behavior of the process until t.

The density of the life time is given as $p(t) = \lambda \exp(-\lambda t)$, $t \geq 0$, $\lambda > 0$.

The probability that the life time T has at most the length s can be expressed as

$$F(s) = P(T \leq s) = 1 - \exp(-\lambda s), \quad s \geq 0, \lambda > 0.$$

The survival probability has the value

$$R(s) = P(T > s) = \exp(-\lambda s), \quad s \geq 0, \lambda > 0.$$

The Poisson process corresponds to a run without fault reducing. For some practical purposes a supposition is meaningful that the intensity decreases by fault detecting and

fault elemination after a failure event. The reliability remains constant, for no fault is removed.

A large number of the well-known models carrys out peacewice constant and decreasing intensities by progressing time. The i-th surviving time has the intensity λ_i. The program will contain N + 1 - i faults in the i-th life time, if in the first interval N faults are supposed. Other ideas exploit a function $\lambda(t)$ instead of λ_i to generate failure events. Finally, it was also tried to model the information of the failure data by the aid of ARIMA processes. A survey and comparisons of existing growth models is given in [VDI93] and in [BGJ98].

2. White Box Modeling

Now we change our interpretation of a program system. We replace the black box concept by the white box concept. For this the source code is broken down into parts. The parts are called components. They represent executable parts of the system. Components also can be modules or subsystems composed by modules. The structure of the language often provides a basis definition of the concept of module.

Models using informations from the internal structure of the program system are called white box models. This models require in general a higher expenditure than black box models. They are able to regard interactions between components. In addtion, such models allow the involving of complexity of each module to evaluate the failure behavior of the system and parts of it. A high complexity of a modul often causes the fault-proneness and the failure of the system. Here the concept of black box is applied to each component of the program decomposition.

2.1 Modeling of Structure

A few papers as given in [LeRu93] have been published on structural software reliability modeling. Our presentation of the white box technique leans on this paper of Ledoux and Rubino. For this approach is assumed that the interaction of components only works by execution control transfer from any system component to another. A sequence of active components is a realisation of a random process. The states of the process correspond to the components. A transition of activity between two fixed components is checked by the call graph of the program structure. Stochasticity is given by the inputs arriving at the system randomly. The call graph is built as Markovian graph. The sojourn time in a component is interpreted as a real-valued random variable. The sojourn time distribution can be shaped in dependence on the realisation of the last sojourn location of the graph. The modeling will be not too expensive if we only regard the suspect model components or links between components. The model of program structure is called execution model. The mathematical presentation of the program execution process as a homogeneous time-continuous Markovian process X is characterized by

a state space, denoted by $M = \{1, 2, ..., m\}$,

an infinitesimal generator, denoted by $Q = (q(i,j)), \quad i,j \in M$,

and an initial distribution, denoted by $\alpha = (\alpha_1, ... , \alpha_m)$.

The process X is assumed to be irreducible and decribes runs of program as abstraction.

2.2 Modeling of Failures

Ledoux and Rubino add the failures process to the execution model and obtain the operational model. The result is a bi-dimensional time-continuous process (N_t, X_t^+), where N_t counts the failures in interval (0,t]. The process X_t^+ will be determined by X , from section 2.1, if the following assumption are met :

(i) an active component i fails with a constant rate

(ii) an activity change from component i to component j fails with a constant interface failure probability.

Additionally, the authors distinguish between primary and secondary failure in both of cases (i) and (ii). Primary failure is followed by an execution break. Secondary failure is not attended by impacts on execution time. Failures within (i) are assumed to be independent. The same is assumed for failures inside (ii) and for pairs of failures of (i) and (ii). The construction of the Markovian process requires the notation of conditional probabilities :

(1) $P(X_{t+dt}^{+} = j \text{ and } N_{t+dt} - N_t = 0 \mid X_t^{+} = i \text{ and } N_t = k)$

(2) $P(X_{t+dt}^{+} = j \text{ and } N_{t+dt} - Nt = 1 \mid X_t^{+} = i \text{ and } N_t = k)$, $\quad i, j \in M.$

The division of terms (1) and (2) by dt and the transition $dt \to 0$ lead to the transition rates without and with failures denotet by a(i,j) and d(i,j), respectively, in [LeRu93]. It holds

(3) $\Sigma_j \ (a(i,j) + d(i,j)) = \Sigma_j \ q(i,j) + \lambda_i + \mu_i, \quad i = 1, ..., M.$

Assuming (i) the notations λ_i and μ_i stand for failure rates in cases of primary and secondary failures. Assuming (ii) the probabilities for the primary and the secondary case, respectively, are termed $\lambda(i,j)$ and $\mu(i,j)$. The elements a(i,j) and d(i,j) are formed in dependence on λ_i , μ_i , $\lambda(i,j)$, $\mu(i,j)$ and q(i,j). If we have a primary interface failure, there will be a jump from state i to another state j with the probability $\alpha(i,j)$. The formula

(4) $d(i,i) = (\lambda_i + \Sigma_{k \neq i} \ q(i,k) \ \lambda(i,k) \ ) \ \alpha(i,i) + \mu_i, \quad i \in M,$

shows possible transitions in failure case without any change of state. The last facts prove that X_t^{+} can be seen as a Markovian process being irreducible on M. The start distribution is α and the matrix

(5) $Q^{+} = A + D$ is the infinitesimal generator of the process.

3. Demonstration Example

3.1 The Code Example

Now we describe an example demonstrating concepts and techniques given above. We observe failures caused by a SRC-Modula-3 freeware release [Mod3], because we found some failures in arithmetical operations while we were using the above mentioned Modula-3 version in a computer course. For instance the addition of positive integers provides a negative result without any hint from the system. Hand-calculated we can detect that the sum is out of integer range. For example, the sum of integer'last and 1 constitutes ostensible integer'first. In a failure observation program we perform the arithmetical operations addition (+), subtraction (-), multiplication (×) and division (/) working some data producing by a uniform pseudo random number generator in the following kind:

– At first we randomly choose one operation.
– The uniform pseudo random generator gives two random integers lying between an upper and a lower bound of a fixed range.
– The chosen operation provides a result exploiting the input.
– The next operation is determined in dependence on a result.

The four operations are here the components defined above. The interaction of the components in an observation program was performed by control transfer from the addition to the subtraction, from the subtraction to the multiplication, from the multiplication to the division, from the division to the addition and so on in cycles. But, if a wrong result appears, the activity of components will be not transfered and a corresponding failure counter will be increased by one.

3.2 Choice of Time Basis

A failure model has to use a time basis for length measuring of a time interval. Our example allows to choose a rigorous discrete time basis. We count one component execution as one time step. Then we can count the number of operations between two neighbouring failures to measure a life time interval. The simplified model is a time discrete Markovian chain. Another time discrete or time continuous model may be used different. But, we do not consider this.

3.3 Modeling of Failures

The run of chains is managed by a quadratic matrix of transition. Its entries decribe the possibilities of the state changes at one time step while the program execution and are called transition probabilities. This matrix holds for each time step.

The transition matrix has the form :

(6)

$$\begin{array}{c} \\ + \\ - \\ \times \\ / \end{array} \begin{array}{c} \begin{array}{cccc} + & - & \times & / \end{array} \\ \begin{pmatrix} * & * & 0 & 0 \\ 0 & * & * & 0 \\ 0 & 0 & * & * \\ * & 0 & 0 & * \end{pmatrix} \end{array}$$

for our program example.

For instance, the transition from / to × has the entry zero.

Symbol * means a not negative probability value to transfer the activity from one component to another one. If a one-step-connection of two different components does not exist in the call graph of the program, a zero entry will be given to the matrix element corresponding to these components. Summation of entry values of a row must be equal to one. Entries situated in the diagonal represent the failure behavior of the Modula-3-system

with respect to the four arithmetical operations. A zero in the diagonal means that the corresponding component is without any failure. The failure structure (6) is different from structure (7) and also from (8) :

$$(7)\quad \begin{array}{c} \\ + \\ - \\ \times \\ / \end{array} \begin{array}{c} \begin{array}{cccc} + & - & \times & / \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & * & * & 0 \\ 0 & 0 & * & * \\ * & 0 & 0 & * \end{pmatrix} \end{array} \quad , \quad (8)\quad \begin{array}{c} \\ + \\ - \\ \times \\ / \end{array} \begin{array}{c} \begin{array}{cccc} + & - & \times & / \end{array} \\ \begin{pmatrix} 0 & * & * & 0 \\ 0 & * & * & 0 \\ 0 & 0 & 0 & 1 \\ * & 0 & 0 & * \end{pmatrix} \end{array} .$$

In (7) and (8) the operation + is without any failure. But, the scheme (8) allows a program structure with a connection from + to × . The operation × may be without any failure in (8), for it uses the failure free operation + . The sequence of schemes shows removing of faults in the addition + , (6) → (7), in the multiplication × , (7) → (8).

In our experiments runs without any failures correspond to schemes with only zero entries in the diagonal.

The art to relate failures on the one side to programm structure and on the other side to functionality that is determined by input qualities in the operational phase can be supported by white box experiments.

3.4 Input Variation

Now it is time for an explanation how to get values from the program for each * entry within the transition matrix. A long run of the observation program gives frequencies of transitions being exploited as estimates for transition probabilities. These values characterize the distribution of used input data.

To demonstrate the influence of different input behavior, we see over the frequencies of two transition matrices below from a program structure (6) to compare each entry value of one matrix with the corresponding value of the other matrix. For each of matrices (9) and (10) we generated random numbers in the same integer range.

Numbers are chosen from $[-2^{31}+1, \ldots, 2^{31}-1]$ for operations +, -, / and

$[-2^{17}, \ldots, 2^{17}]$ for operation $\times$.

Entry values in experiments are given by

$$(9)\quad \begin{array}{c} \\ + \\ - \\ \times \\ / \end{array} \begin{array}{c} \begin{array}{cccc} + & - & \times & / \end{array} \\ \begin{pmatrix} .28 & .72 & 0 & 0 \\ 0 & .24 & .76 & 0 \\ 0 & 0 & .64 & .36 \\ 1 & 0 & 0 & 0 \end{pmatrix} \end{array} \qquad (10)\quad \begin{array}{c} \\ + \\ - \\ \times \\ / \end{array} \begin{array}{c} \begin{array}{cccc} + & - & \times & / \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & .27 & .73 & 0 \\ 0 & 0 & .57 & .43 \\ 1 & 0 & 0 & 0 \end{pmatrix} \end{array} .$$

In the first experiment we generated all operands at random and independently. We contained matrix (9). In a second input simulation (10) we used all correct operation results of the process to be the first operands of the next operation in the following step. That means in the process providing (10) the operation + seems to be flawless. But, in reality the failure of the operation + does not appear in consequence of the changed functionality of the program in the operation. This shows the impact of input quality on the failure probability. We note that the number of use the four operations are the same in experiments of (9) and (10).

4. Reliability in the Operational Phase

In this section we develop a principal view of software reliability. This emphasis to the operational phase throw light on the interplay between the reliability and the test phase. We suppose that all the functions given in the specification of application were treated satisfactory in the test phase. A final test report has to contain an assignment of certain input ranges to particular functions. Now we assume that failure events arrive in the operational phase. The input quality used in a failure case reveals a test deficiency. Program faults detected in the operational phase were not found or were tolerated during the test phase. For instance the test phase will tolerate a fault, if the failure does probably not arrive or, if we can go with the damage size. The reason of failuring is given in the release used in the operational phase, so the resulting product only has insufficient quality. In a lot of situations we suppose that the possible failures can be explained with the nature of particular components or with interface failures. For this we go on with a low complexity of components being reasonable. Furthermore we assume that the submitted structure masters the required functionality. In section 3.4 we have demonstrated by an example, how the different use of a program system can lead to different reliability evaluations. So we have emphasized a dynamic effect, which is due to the variable data quality. This effect has to be regarded at determining the operational profile of a program enviroment. The concept of the dynamic complexity from Khoshgoftaar et. al. , [KML93] does not regard this aspect.

The connection of static complexity and operational profiles determines the dynamic complexity of a program system. The importance of static complexity information about

these system parts and about interactions increases with the growing size of a program system. Complexity metrics can be used for the identification of system parts prone to failures. Additionally, such models offer possibilities to detect faults by input experiments of critical parts of system. Large parts of a system and also the whole system can be characterized by domain attributes called factors of complexity, [Mun94], [ScSc97]. Complexity assessments of the system give important hints for the system decomposition in parts operating interactive.

White box modeling is a proved remedy in the operational phase, since is allows the simulation of many parts of the program system as an abstract enviroment of the critical components while experimental runs. White box models are able to connect the concept of the operational process, [LeRu93], with the concepts of the important profiles of the dynamical complexity: functional, execution and system component profile, [Mun94]. White box models are flexible for adapting to different operational enviroments on different levels of abstraction.

Literatur:

[BGJ98] Erprobte Modelle zur Quantifizierung der Softwarezuverlässigkeit F. Belli,

		G. Grochtmann, O. Jack in Informatikspektr. Band 21 Heft 3 1998
[BuFi91]	The Infeasibility of Experimental Quantification of Live-Critical Software Reliability	R. W. Butler, G. B. Finelli ACM, 1991, 0-89791-455-4/91/0011/0066
[KML93]	Dynamic System Complexity	T. M. Khofgoshgoftaar, J. C. Munson, D. L. Lanning IEEE 1993, 0-8186-3740-4/93
[LeRu93]	A Counting Model for Software Reliability Analysis	J. Ledoux, G. Rubino Rapport de recherche n^0 2060 juillet 1993, Rennes, unité de recherche INRIA
[Mod3]	Modula3-Implementation auf DEC 5000/260, MIPS R4400 ultrix4.4	Rechenzentrum der Fak. für Mathematik und Informatik, FSU Jena , Aug. 98
[Mun94]	Software Faults, Software Failures and Software Reliability Modeling	J. C. Munson, IEEE : 1-4, 1994
[ScSc97]	The Use of Factor Analysis in the Area of Software Metric	D. Schmelz, M. Schmelz Software Metrics (F. Lehner, R. Dumke A. Abran Eds.)Gabler - Verlag, Wiesbaden 97
[VDI93]	Softwarezuverlässigkeit Grundlagen, Konstruktive Maßnahmen, Nachweisverfahren	VDI-Verlag Düsseldorf 93 Herausg. : VDI - Gemeinschaftsausschuß Industrielle Systemtechnik

COMMON SOFTWARE METRICS INTERNATIONAL CONSORTIUM

Preliminary Public Announcement

We are delighted to announce the start-up of 'COSMIC', the Common Software Metrics International Consortium. COSMIC aims to develop improved ways of measuring the size of software for use in software development project estimating and in measuring the performance achieved in software activities.

Background

Accurate estimating and measuring the size of software is a subject of huge economic importance for the software industry.

Software suppliers face the task of translating customer requirements into the size of software to be produced as a key step in their project cost estimating. Customers want to know the size delivered as an important component of measuring supplier performance.

Given the explosive growth and diversity of software contracting and outsourcing, suppliers and customers need more accurate ways of estimating and of measuring performance which must work equally reliably across all types of software. Current methods for measuring the size of software are not always of sufficient strength to meet market needs, or work only for restricted types of software. Industry urgently needs software size measures which are demonstrably more accurate and more widely usable.

As software contracting and outsourcing is a global activity, it is also essential that the new methods are tested and accepted internationally.

The COSMIC initiative aims to meet these needs.

Conditions are now ripe for success

The consensus of a large number of industry experts in estimating and performance measurement is that there is now sufficient experience with existing software sizing methods (specifically the 'IFPUG', 'MkII', 'FFP' methods and ideas embodied in the developing ISO standards) together with emerging ideas, that the new methods can be developed in the very short term.

A group of internationally recognised experts in the software measurement field have pooled their resources to start work on defining the new methods. Their target is to have a first release ready for testing within a very few months. Experts participating in the project are initially drawn from Australia, Canada, Finland, Netherlands, UK and the USA.

The group has formally defined a set of Aims (see below) and established itself as the Common Software Measurement International Consortium, or 'COSMIC'.

The need for field testing and hence industrial sponsorship

Experience shows that development and thorough field testing of new software sizing methods to the point where they are demonstrated to be robust can take several years. But industry urgently needs the new methods today.

The COSMIC founders are therefore currently seeking sponsorship from major software producers and users to support the detailed design effort and to participate in field testing which is planned to begin in the next few months. If enough sponsoring partners can be found, the time to market can be reduced from several years to one to two years.

If you are interested please contact one of the following:

- Americas alain.abran@cosmicon.com
- Australia, South East Asia and Japan pam.morris@cosmicon.com
- British Isles , Middle East, India charles.symons@cosmicon.com
- Scandinavia risto.nevalainen@cosmicon.com
- Western Europe jolijn.onvlee@cosmicon.com
- Other Regions peter.fagg@cosmicon.com

COSMIC AIMS

To develop test and bring to market a new generation of software size measure(s) which

Business purposes (highest priority first):

- are suitable as a basis for normalising and comparing performance measures of productivity (size/effort), speed of delivery (size/elapsed time) and quality (e.g. defects/size) for activities throughout the life cycle of the software
- are suitable for use as a component of estimating methods for development and maintenance effort and time
- can assist in the estimation of the operational life cycle costs

(Scope of applicability)

- in principle applicable to as wide as possible a range of software 'domains'; in practice priority will be given to business software and its supporting software as in operating systems, and to real-time software as in telephony, process-control, or embedded software; a second priority will be to investigate algorithmic-rich software

(Desirable Characteristics)

- should be derivable from user requirements; the size measures should separately cover functional user requirements and other user requirements (eg. technical and quality requirements). The priority will be to initially measure functional user requirements. These methods should also work for sizing information processing requirements before they are specifically allocated to software
- are based on some verifiable theory, i.e. the method has an academically sound basis and we should be able to explain very clearly what it is we are measuring. The sizing methods should be independent of specific software development methods and notations, but compatible with modern ways of stating software requirements such as structured methods,

relational data analysis, the object orientated paradigm, etc.; conformant with ISO Standard 14143/1 on Functional Size Measurement and conformant with measurement theory

- draw on the best ideas from the current ISO 14143, NESMA, IFPUG 4.1, MkII 1.3.1, and FFP methods and other relevant concepts if needed, but unconstrained by adherence to any particular method
- can be calibrated to demonstrable levels of accuracy and precision for defined purposes, that is producing software sizes with known confidence levels. This implies the weightings (or 'units' allocated to the various types of requirements) have some rationale behind them, for example they can be justified in relation to measurable or observable phenomena in real software.
- should be reasonably simple to explain to potential users of the methods and require a manageable effort to apply.
- should be precisely defined and designed to be repeatable, so to aim to eliminate subjectivity. The result will be that the methods will be more easily automated
- should be sufficiently widely accepted to be regarded as an Industry Standard; target maximum time to market should be 2 years.

The COSMIC Core Team
November 1998

Contact address:

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

Some of the Quality Initiatives in Germany:

- Arbeitskreis Software-Qualität Franken e.V.

see: http://www.asqf.de

- Gesellschaft für Softwarequalitätssicherung GmbH

see: http://www.sqs.de

- Fraunhofer Institut Experimentelles Software Engineering (IESE)

  see: http://www.iese.fhg.de/

- Deutsche Informatik Akademie Seminare:

  - Strategien zur Verbesserung des Softwareentwicklungsprozesses (Dr. C. Ebert)
  - Methodisches Testen und Analysieren von Software (Dr. P. Liggesmeyer)
  - Software-Projektsteuerung für die Praxis (Metriken für Projektmanagement, Qualitäts- und Prozeßverbesserung) (Dr. C. Ebert)

  see: http://www.gi-ev.de/dia/

- Deutschsprachige Anwendergruppe für Softwaremetriken und Aufwandsschätzung (DASMA)

  see: http://home.t-online.de/erwin.loch/dasma_index.htm

- European TeleCASE Center (ETC) in Braunschweig

  see: http://www.dlr.de/bs_d.html

- Das Software Meßlabor (SMLAB) der Universität Magdeburg

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

- etc.

LEHNER, F.; DUMKE, R.; ABRAN, A.: SOFTWARE METRICS - RESEARCH AND PRACITCE IN SOFTWARE MEASUREMENT

Gabler-Verlag, Wiesbaden, 1997 (232 p.) *ISBN: 3-8244-6518-3*

This book contains all presentations of the 1996 workshop of the GI-interest group on software metrics and of the Canadian Group (CIM) in September in Regensburg. It is a collection of theoretical studies in the field of software measurement as well as experience reports on the application of software metrics in Canadian, Austrian, Belgian and German companies and universities. Some of these papers and reports describe new software measurement applications and paradigms for knowledge-based techniques, maintenance service evaluation, factor analysis discussions and neural-fuzzy applications. Others address

the object-oriented paradigm and discuss the application of the Function Point approach to an object-oriented design method, the evaluation of the Java development environment, the analysis of quality and productivity improvements of object-oriented systems, as well as the definition of the metrics of class libraries. Other papers offer a different perspective, presenting a software measurement education system designed to help improve the lack of training in this field, for example, or they include experience reports about the implementation of measurement programs in industrial environment.

Poulin, J.S.: Measuring Software Reuse

Addison-Wesley, 1997 (195 p.)

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

Dumke, R.; Foltin, E.; Koeppe, R.; Winkler, A.: Softwarequalität durch Meßtools - Assessment, Messung und instrumentierte ISO 9000

Vieweg Publ., 1996 (223 p.) *ISBN 3-528-005527-8*

This book gives an overview about the software metrics tools for all phases of the software development process. The metrics tools are defined as CAME tools (Computer Assisted Software Measurement and Evaluation). The introduction describes the essential aspects of the software measurement. The description oft the CAME tools includes the cost estimation tools, Capability Maturity Model evaluation tools, metrics tool for software specification, design and code, and tools forthe software testing and maintenance (including network performance). Some tables helps to decision of choosing the useful tool integration for the software quality assurance process.

Zuse, H.: A Framework of Software Measurement

de Gruyter Publ., Berlin New York, 1997 (755 p.) *ISBN 3-11-015587-7*

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

Dumke, R.; Abran, A.: Software Measurement – Current Trends in Research and Practice

DUV Publisher, Wiesbaden, 1999 (269 p.) *ISBN 3-8244-6876-X*

This new book includes key papers presented at the 8th International Workshop on Software Metrics in Magdeburg (Germany), September 1998. 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 USA, Canadian, Netherlands, Belgian, France, England and German companies and universities. Some of these papers and reports describe new software measurement applications and paradigms for knowledge-based techniques, test service evaluation, factor analysis discussions and neural-fuzzy applications. Other address the multimedia systems and discuss the application of the Function Point approach for real-time systems, the evaluation of Y2K metrics, or they include experience reports about the implementation of measurement programs in industrial environments.

70 Position Papers

CSMR'99:

3nd Euromicro Working Conference on Software Maintenance and Reengineering,
March 3-5, 1999, Amsterdam, Netherlands
see: **http://www.wins.uva.nl/events/CSMR99/**

SEPG'99:

11th Software Engineering Process Group Conference,
March 8-11, 1999, Atlanta, USA
see: **http://www.sei.cmu.edu/products/sepg99/**

IFPUG'99, Spring:

International Function Point User Group Spring Conference,
April 26-27, 1999, New Orleans, USA
see: **http://www.ifpug.org/conferences/conf.html**

ESSDE'99:

Empirical Studies of Software Development and Evalution,
May 18th, 1999, Los Angeles, USA (during the ICSE99)
see: **http://www.staff.ecs.soton.ac.uk/~rh/ICSE99/cfp.html**

QW'99:

Quality Week 1999,
May 24-28, 1999, San Jose, USA
see: **http://www.soft.com/QualWeek/QW99**

IWSM'99:

9th International Workshop on Software Measurement,
September 8-10, 1999, Montreal – Mont-Tremblant, Canada
see: **http://www.lrgl.uqam.ca/iwsm99/**

CONQUEST'99:

Conference on Quality Engineering in Software Technology,
September 27-28, 1999, Nuremberg, Germany
see: **http://www.asqf.de/**

FESMA'99:

Second European Conference on Software Measurement,
October 4-7, 1999, Hamburg, Germany
see: **http://www.ti.kviv.be/conf/fesma.htm**

IASTED'99:

International Conference Software Engineering,
October 6-8, 1999, Scottdale, Arizona, USA
see: **http://www.iasted.com/**

IFPUG'99, Fall:

International Function Point User Group Fall Conference,
October 20-22, 1999, New Orleans, USA
see: **http://www.ifpug.org/conferences/conf.html**

METRICS'99:

Sixth International Symposium on Software Metrics,
November 5-6, 1999, Boca Raton, Florida, USA
see: **http://www.iese.fhg.de/METRICS99/metrics99.htm**

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

Other Information Sources and Related Topics

- **http://rbse.jsc.nasa.gov/virt-lib/soft-eng.html**
  Software Engineering Virtual Library in Houston

- **http://www.mccabe.com/**
  McCabe & Associates. Commercial site offering products and services for software developers (i. e. Y2K, Testing or Quality Assurance)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

- **http://freedom.larc.nasa.gov/spqr/spqr.html**
  Software Productivity, Quality, and Reliability N-Team

News Groups

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

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

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

METRICS NEWS

VOLUME 3 *1998* *NUMBER 2*

CONTENTS

ISSN 1431-8008