A major difference between a "well developed" science
such as physics and some of the less
"well-developed" sciences such as psychology or sociology
is the degree to which things are measured.
.
Fred S. Roberts /ROBE79/.

1 History of Software Measurement

Horst Zuse

Measurement in science has a long tradition as described by Lemmerich /LEMM88/. While the measurement of length - the meter - was defined in 1889 /LEMM88/, p.87, the measurement of temperature was more complicated. The graduation of the distance of the fixpoints was introduced by Fahrenheit in 1714 and by Celsius in 1742. Celsius devided the scale of 100 degrees and 0 degrees in hundred equal sectors. But the question was for a long time whether 50C was uniquely defined. Another interesting case is that the measurement of length is always a ratio scale, but temperature can be both an interval (Celsius / Fahrenheit) or a ratio scale (Kelvin).

Today, computers play a primary role in almost every area of our life. The increased importance of software also places more requirements on it. Thus, it is necessary to have precise, predictable, and repeatable control over the software development process and product. Software measures are a tool to measure the quality of software.

The area of software measurement or software engineering measurement, is one of the areas in software engineering where researchers are active since more than thirty years. The area of software measurement is also known as software metrics. There is a confusing situation using the terms software measures or software metrics. Using the terminology of measurement theory /ROBE79/ we use the term measures. In literature the terms metric and measure are used as synonyms. A metric is here not considered in the sense of a metric space, it is considered as: measurement is a mapping of empirical objects to numerical objects by a homomorphism. A homomorphism is a mapping which preserves all relations and structures. Put in other words: software quality should be linearly related to a software measure. This is a basic concept of measurement at all and of software measurement.

Software measure researchers are split into two camps: those who claim software can be measured, and those who say that software can not be analyzed by measurement. In any case, the majority of researchers are concerned about software quality and the need to quantify it. During the past more than one thousand software measures were proposed by researchers and practitioners, and till today more than 5000 papers about software measurement are published. For this reason it is not possible to give a complete overview of the history of software measures. We only discuss some *milestones* in the development of software measures. An overview of software measurement or of published papers can be found, among others, in the following books or papers: /CONT86/, /GRAD87/, /EJIO91/, /DUMK92/ /MILL88/, /PERL81/, /FENT91/, /FENT90/, /KITC89/, /MOEL93/, /SHEP93/, /SHEP93a/, /JONE91/, /GOOD92/, /SAMA87/, /GRAD92/, /DEMA82/, /AMI92/, /CARD90/, /HETZ93/, /SHOO83/, /MAYR90/, /SOMM92/, /PRES92/, /ZUSE91/, /ZUSE94a/, and /ZUSE95a/.

1.1 Why (Software) Measurement?

Measurement has a long tradition in natural sciences. At the end of the last century the physicist, Lord Kelvin (1824-1904), formulated the following about measurement /KELV91/: *When you can measure what you are speaking about, and express it into numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meager and unsatisfactory kind: It may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the stage of science.* This view of the application measurement in sciences is also supported by scientists who treat with measurement theory. Fred S. Roberts /ROBE79/, p.1 points out in his excellent book about measurement theory: *A major difference between a "well developed" science such as physics and some of the less "well-developed" sciences such as psychology or sociology is the degree to which things are measured.*

In the area of software engineering, measurement is discussed since ca. 25 years. 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 /CURT80/ 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.*

Here, the goals of software measurement are to answer, among others, the following questions:

- Is it possible to predict the error-proneness of a system using software measures from its design phase.
- Is it possible to extract quantitative features from the representation of a software design to enable us to predict the degree of maintainability of a software system.
- Are there any quantifiable, key features of the program code of a module that would enable us to predict the degree of difficulty of testing for that module, and the number of residual errors in the module after a particular level of testing has occurred.
- Is it possible to extract quantifiable features from the representation of a software design to enable us to predict the amount of effort required to build the software described by that design.
- Are there quantifiable features that can be extracted from the program code of a subroutine that can be used to help predict the amount of effort required to test the subroutine.
- Are there features in order to predict the size of a project from the specification phase.
- What properties of software measures are required in order to determine the quality of a design.

- What are the right software measures to underlie the software quality attributes of the ISO 9126 norm by numbers.

The major question for people who want to use software measures or using software measures is: what are the benefits of software measurement for practitioners. We think, that Grady formulated the advantage and the need of software measurement in a clear way. In 1992 Grady /GRAD87/, /GRAD87a/, /GRAD90/, formulated the relevance of software measurement. Software measures are used to measure specific attributes of a software product or software development process.

1. We use software measures to derive:
2. A basis for estimates,
3. To track project progress,
4. To determine (relative) complexity,
5. To help us to understand when we have archived a desired state of quality,
6. To analyze or defects,
7. and to experimentally validate best practices.

8. In short: they help us to make better decisions.

Of course, there are many other statements of the need of software measurement, but it would be outside the scope of this book to mention them all.

1.2 Groundworks of Software Measurement

The groundwork for software measures and software measurement was established in the sixties and mainly in the seventies, and from these earlier works, further results have emerged in the eighties and nineties.

The reasons for creating or inventing software measures is based on the knowledge that program structure and modularity are important considerations for the development of reliable software. Most software specialists agree that higher reliability is archived when software systems are highly modularized and module structure is kept simple /SCHN77/. Modularity has been discussed early. Parnas /PARN75/ suggested that modules should be structured so that a module has no knowledge of the internal structure of other modules, Myers /GLEN75/ described the concept of module strength, and Yourdon discussed modularity in 1975 (YOUR75/. These factors affect cost and quality of software, as, for example, Porter et al. point out /PORT90/.

The earliest software measure is the Measure LOC, which is discussed and used till today /PARK92/. In 1974 Wolverton /WOLV74/ made one of the earliest attempts to formally measure programmer productivity using lines of code (LOC). He proposed object instructions per man-month as a productivity measure and suggested what he considered to be typical code rates. The basis of the Measure LOC /SHEP93/, p.3, is that program length can be used as a predictor of program characteristics such as reliability and ease of maintenance. Despite, or possibly even because of, simplicity of this metric, it has been subjected to severe criticism. Estimated and actual source lines are used for productivity estimates, as described by Walston and Felix /WALS77/, during the proposal and performance phases of software contracts. In the sixties SLOC (Source Lines of Code) were counted by the number of 80-column cards. In 1983 Basili and Hutchens /BASI83/ suggested that the Metric LOC should be regarded as a baseline metric to which all other metrics be compared. We should expect an effective code metric to perform better than LOC, and so, as a minimum, LOC offers a "null hypothesis" for empirical evaluations of software metrics. The Measure LOC is mentioned in more than ten thousand papers.

It may be, that the earliest paper about software complexity was published by Rubey et al. /RUBE68/ in 1968. In the list of literature in this paper is no reference to an earlier publication. In 1979 Belady /BELA79/ mentioned a dissertation about software complexity in 1971. Belady writes: *In 1974 there was very little known work on complexity in general, and even less on the complexity of programming. At that time Prof. Beilner and myself shared an office at Imperial College and, according to our mutual interest, discussed problems of "complex" programs and of large scale programming. Sometimes we asked: what is complexity? Is it possible to come up with a reasonable definition? Is the intuitive concept of complexity quantifiable? Since we could not answer these questions, we turned to literature. And soon, among the many rather philosophical essays on complexity, we hit upon a very interesting doctoral thesis by Van Emden: An Analysis of Complexity" /EMDE71/.*The work of Van Emden was based on the concept of conditional probabilities on the formalism of information theory, and appeared suitable to model complexity of interconnected systems, such as programs built of modules.

Early in the sixties, the cost estimation models Deplhi /HELM66/ and Nelson's SDC (NELS66/ were introduced.

In 1975, the term Software Physics was created by Kolence /KOLE75/, and in 1977 Halstead introduced the term software science. The idea behind this terms was to apply scientific methods to the properties and structures of computer programs. Kolence 's theory connects such traditional performance measures as turnaround time, system availability, and response time with traditional management measures such as productivity, cost per unit service, and budgets. Software Physics was among the first theories to deal exclusively with computer sizing and workloads /MORR76/.

The most famous measures, which are continued to be discussed heavily today and which were created in the middle of the seventies are the Measures of McCabe /MCCA76/ and of Halstead /HALS77/. McCabe derived a software complexity measure from graph theory using the definition of the cyclomatic number. McCabe interpreted the cyclomatic number as *the minimum number of paths* in the flowgraph. He argued that the minimum number of paths determines the complexity (cyclomatic complexity) of the program:. *The overall strategy will be to measure the complexity of a program by computing the number of linearly independent paths v(G), control the "size" of programs by setting an upper limit to v(G) (instead of using just physical size), and use the cyclomatic complexity as the basis for a testing methodology.* McCabe also proposed the *measure essential complexity,* which may be the first measure which analyzes unstructuredness based on primes. In 1989 Zuse et al. /ZUSE89/ /ZUSE91/ showed that the idea of complexity of the Measure of McCabe can be characterized by three simple operations. The authors derived this concept from measurement theory (See also Section 1.5).

The Measures of Halstead are based on the source code of programs. Halstead showed that estimated effort, or programmer time, can be expressed as a function of operator count, operand count, or usage count /HALS76/. Halstead's method has been used by many organizations, including IBM at its Santa Teresa Laboratory /CHRI81/, General Electric Company /FITS78/, and General Motors Corporation /HALS76/, primarily in software measurement experiments. Today the most used Measures of Halstead are the Measures Length, Volume, Difficulty and Effort. Halstead died at the end of the seventieth and it is a pity that he cannot defend his measures today.

In 1977 Laemmel and Shooman /LAEM77/ have examined Zipf's Law, which was developed for natural languages, and extended the theory to apply the technique to programming languages. Zipf's Law is applied to operators, operands, and the combinations of operators and operands in computer programs. The results show that Zipf's Law holds for computer languages, and complexity measures can be derived which are similar to those of Halstead.

In 1978 followed a proposal of software complexity measurement by McClure /MCCL78/. Two other software complexity measures (Interval-Derived-Sequence-Length (IDSL) and Loop-Connectedness (LC)) were proposed in 1977 by Hecht /HECH77/ and are discussed in /ZUSE91/, p.221. They are based on the reducibility of flowgraphs in intervals. However, they are not well known. The works of Rubey, Van Emden and the Measures of Hecht have been largely forgotten. However, in 1992 the Measure of Van Emden was used as a basis of a complexity measure by Khoshgoftaar et al. /KHOS92/.

In 1977 Gilb /GILB77/ published a book *entitled Tom Gilb: Software Metrics, which* is one of the first books in the area of software measures. In 1979 Belady /BELA79/ proposed the Measure BAND which is sensitive to nesting.

Already in 1978 /JONE78/ published a paper where he discusses methods to measure programming quality and productivity. Interesting, among others, in this paper are his considerations of units of measures.

Albrecht /ALBR79/ introduced in 1979 the Function-Point method in order to measure the application development productivity.

In 1980 Oviedo /OVIE80/ developed *a Model of Program Quality*. This model treats control flow complexity and data flow complexity together. Oviedo defines the complexity of a program by the calculation of control complexity and data flow complexity with one measure.

In 1980 Curtis /CURT80/ published an important paper about software measurement. Curtis discusses in the Chapter: Science and Measurement, some basic concepts of measurement. He points out that in a less-developed science, relationships between theoretical and operationally defined constructs are not necessarily established on a formal mathematical basis, but are logically presumed to exist. And, among others, he writes: The more rigorous our measurement techniques, the more thoroughly a theoretical model can be tested and calibrated. Thus progress in a scientific basis for software engineering depends on improved measurement of the fundamental constructs. Here, Curtis refers to Jones /JONE78/. Jones /JONE77/, /JONE78/, /JONE79/ is also one of the pioneers in the area of software measurement.

In 1981 Ruston /RUST81/ proposed a measure which describes a program flowchart by means of a polynomial. The measure takes into account both the elements of the flowchart and its structure. Ruston's method appears to be suitable for network measurement, but has not been used as widely as McCabe's method.

In 1981 Harrison et al. /HARR81/ presented software complexity measures which are based on the decomposition of flowgraphs into ranges. Using the concept of Harrison et al. it is possible to determine the nesting level of nodes in structured and especially unstructured flowgraphs. This is an important extension of the Measures of Dunsmore, Belady, etc. /ZUSE91/, p.269, p.362, which were only defined for flowgraphs consisting of D-Structures (Dijkstra-Structures). In 1982 Piwowarski /PIWO82/ suggested a modification of the Measures of Harrison et al. because these measures have some disadvantages, for example unstructured flowgraph can be less complex than structured flowgraphs.

Troy et al. /TROY81/ proposed in 1981 a set of 24 measures to analyze the modularity, the size, the complexity, the cohesion and the coupling of a software system. Especially cohesion and coupling are fundamental criteria of the understandability of a software system. The basic division of software (complexity) measures into inter-modular and intra-modular components and the specific conceptual measures of coupling and cohesion are based on a work of Constantine (CONS68/. Measures for cohesion were proposed in 1984 by Emerson /EMER84a/ and /EMER84b/. In 1982 Weiser presented the concept of slices /WEIS82/, /WEIS84/. Based on the concept of slices measures for cohesion were discussed in 1986 by Longworth et al. /LONG86/ and in 1991 by Ott et al. /OTT91/. However, there are some other papers which deal with the term cohesion in the context of measures like: Dhama /DHAM94/, Lakhotia /LAKO93/, Patel et al. /PATE92/, Rising et al. /RISI92/, Selby et al. /SELB88/, and Thuss /THUS88/. The articles of Ott et al. /OTT89/ /OTT91/ /OTT92/ /OTT92a/ /OTT92b/ /OTT92c/ and /OTT92d/ are fundamental considerations of the term cohesion by software measures.

In 1981 a Study Panel /PERL81/ consisting of people of the industry and universities (Victor Basili, Les Belady, Jim Browne, Bill Curtis, Rich DeMillo, Ivor Francis, Richard Lipton, Bill Lynch, Merv Müller, Alan Perlis, Jean Summet, Fred Sayward, and Mary Shaw) discussed and evaluated the current state of the art and the status of research in the area of software measures. During this panel DeMillo et al. /DEMI81/ discussed the problem of measuring software compared to other sciences. They discussed the problem of meaningfulness. However, they did not give conditions for the use of software measures as an ordinal and a ratio scale (See also Section 1.5).

In the eighties several investigations in the area of software measures were done by the Rome Air Development Center (RADC) /RADC84/. In this research institute the relationships of software measures and software quality attributes (usability, testability, maintainability, etc.) were investigated. The goal of

these investigations is the development of the Software Quality Framework which quantifies both user- and management-oriented techniques for quantifying software product quality.

NASA with the SEI (Software Engineering laboratory) also started very early with software measurement /NASA84/. It is one of the few institutions who uses software measurement since more than 15 years /NASA90/, p. 6-48. Closely connected with NASA is the work of Basili /BASI75/, /BASI77/, /BASI79/.

1.3 Software Design Measurement

The simplest software design measure has been proposed by Gilb /GILB77/ in 1977, namely the number of modules, however, more sophisticated design measures were proposed in 1978 by Yin et al. /YIN78/ and Chapin /CHAP79/. These measures maybe the first measures which could be used in the software design phase of the software life-cycle. Software measures, which can be used in the design phase, were proposed among others by Bowles /BOWL83/, Troy et al. /TROY81/, McCabe et al. /MCCA89/, Card et al. /CARD90/, Sheppard et al. /SHEP93/, Ligier /LIGI89/, /KITC90/, /KITC90a/, and Zuse /ZUSE92a/.

Based on the works of Stevens, Myers and Constantine /STEV74/ much work has been done to create software (complexity) measures for the Interconnectivity analysis of large software systems. Software systems contain multiple types of interrelations between the components, like data, control, and sequencing among others. In 1981 the *Interconnectivity Metric* of Henry and Kafura /HENR81/ was proposed. This measure is based on the multiplication of the fan-in and fan-out of modules. At the end of the eighties a modification of this measure /HENR88/ was proposed by creating a hybrid measure consisting of a intra-modular measure, like the Measures of Halstead and McCabe and the "Interconnectivity Metric". Other approaches, for example of Selby et al. /SELB92/ and Hutchens et al. /HUTC85/ base on the early works of Myers /MYER76/ and Stevens et al. /STEV74/, and propose software measures based on data binding. Other works can be found in /BOLO88/, /ROBI89/ and /HARR87/

Hall /HALL83/, /HALL84/ proposed in 1983/84 software complexity measures in order to analyze large systems. Although this approach is very intuitive, the measure are largely forgotten.

At the end of the eighties and the beginning of the ninetieth some effort was spent to create standardizations of software measures. Two IEEE-Reports /IEEE89/, /IEEE89a/ appeared which propose standardized software measures. Unfortunately, most of the discussed software measures in these reports are process measures and measures for the maintenance phase of the software life-cycle. Only some software complexity measures can be found there.

1.4 Cost Estimation Measures

Cost estimation models were one of the first concepts using measurement. The following table gives an overview of some cost estimation models:

Name of Model	Organization	Year	Reference
Delphi	Rand Corp.	1966	/HELM66/
Nelson's SDC	SDC	1966	/NELS66/
Wolverton	TRW	1974	/WOLV74/
RCA Price-S System	RCA	1976	/FREI79/

Halstead		1977	/HALS77/
Walston and Felix	IBM	1977	/WALS77/
Function-Point Method		1979	/ALBR79/
Parr Model		1980	/PARR80/
COCOMO -Model	TRW	1981	/BOEH81/
SOFTCOST	JPL	1981	/TAUS81/
Bailey and Basili	NASA	1981	/BAIL81/
Bang Metrics		1982	/DEMA82/
MARK II Function Points		1988	/SYMO88/
Pfleeger Model		1989	/PFLE91/

Figure 3.x: Table of cost estimation models.

In 1979 Albrecht /ALBR79/, /ALBR83/ proposed the Function Point method. Function points are derived from requirements specification. This method, which can be used in the specification phase of the software life-cycle, is today widely used in USA and Europe. In 1988 Jones /JONE88/, a strong proponent of Albrecht's function points, proposed an extension of function points to attempt to "validate" the metric for real-time and embedded applications. Jones called his extensions feature points. He introduced a new parameter, called algorithms, and reduced the weights assigned to data file parameters. In 1988 and 1991 this model was modified by Symons /SYMO88/, /SYMO91/. He proposed an extension called Mark II function points to address the following difficulties which he identified with Albrecht's method: The system component classifications are oversimplified, the weights of the components were determined by debate and trail, the internal complexity seems to be rather inadequate, the 14 factors are overlapping and the range of the weights are always valid.

In 1981 Boehm /BOEH81/, /BOEH84/ proposed the COCOMO (Constructive Cost Model)- model which is based in the Metric LOC. A good overview of cost estimation models and measures is given by Kitchenham in /FENT91/, Chapter 9. There are four common methods of cost estimation: Expert opinion, analogy, decomposition and estimation equations. Very interesting is the method of decomposition: :. This involves either breaking a product up into its smallest components or breaking a project up into its lowest level tasks. Estimates are made of the effort required to produce the smallest components or perform the lowest level task. Project estimates are made by summing the components the component estimates.... :. . For this type of prediction only software measures are appropriated which assume an extensive structure, like LOC and McCabe /ZUSE92/, /ZUSE94/.. A possible prediction model, which assumes an extensive structure, is the COCOMO-model /BOEH81/ were the relationship between the external variable EFFORT to develop a program and the Measure LOC is defined as:

$$EFFORT(P)=a\ LOC^{b},$$

where P is a program, and a,b>0. For LOC other software measures, which assume an extensive structure /ZUSE94/, can be used. Other cost estimation models are the Raleigh curve model of Putnam /PUTN78/ and the Bang metric of DeMarco /DEMA87/.

1.5 Goal-Question-Metric Paradigm, User-View and Viewpoints

In 1984 Basili et al. /BASI84/, /BASI87/, /ROMB90b/ proposed the GQM (Goal-Question-Metric) paradigm. GQM is used for defining measurement goals. The basic idea of GQM is to derive software measures from measurement goals and questions. The GQM approach supports the identification of

measures for any type of measurement via a set of guidelines for how to formulate a goal comprehensible, what types of questions to ask, and how to refine them into questions.

The idea that the use of software measures depends on the view of the humans is also supported by Fenton /FENT91a/, p.253 (called: user-view), and Zuse et al. /ZUSE89/, /ZUSE91/, (called: a viewpoint of complexity).

1.6 Measurement Theory and Software Measures

Based on articles of Bollmann and Cherniavsky /BOLL81/, and Bollmann /BOLL84/, in which measurement theory was applied to evaluation measures in the area of information retrieval systems, in 1985 Bollmann and Zuse /ZUSE85/, /BOLL85/ transferred this measurement theoretic approach to software complexity measures. They used measurement theory, as described by Roberts /ROBE79/, Krantz et al. /KRAN71/ and Luce et al. /LUCE90/, which give conditions for the use of measures. In /BOLL85/, /ZUSE85/, /ZUSE87/, /ZUSE89/, /ZUSE91/, and /ZUSE92/ the conditions for the use of software complexity measures on certain scale levels, like ordinal, interval or ratio scale were presented. Additionally, measurement theory gives an empirical interpretation of the numbers of software measures by the hypothetical empirical relational system and conditions for concatenation and decomposition operations, which are major strategies in software engineering. This concept is also applied in /ZUSE91/ for more than 90 software measures. In 1993/94 Bollmann and Zuse extended the measurement theoretic approach to prediction models and in 1994 Zuse /ZUSE94/, /ZUSE94c/ presented the empirical conditions when validating a software measure. In 1995 Zuse et al. /ZUSE95/ showed the empirical conditions behind software measures. It could be shown that measures in the object-oriented area have completely different properties as software measures for imperative languages. Many software measures follow the structure of the function of belief from artificial intelligence.

Similar approaches of using measurement theory followed from 1987 by the Grubstake Group /BAKE87/, /BAKE90/ consisting of the scientists Norman Fenton (City University, London), Robin Whitty (CSSE, South Bank Polytechnic, London), Jim Bieman (Colorado State University), Dave Gustafson (Kansas State University), Austin Melton (Kansas State University), and Albert Baker. The Grubstake Group used measurement theory to describe the ranking order of programs created by software measures. Measurement theory is also proposed as an appropriated theory for software measures by Fenton /FENT91/, p.16.

1.7 European Projects

In 1986 in UK a research project started (Alvey-Project SE/069) entitled "Structured-Based Software Measurement" /ELLI88/. This project was intended to build on existing research into formal modeling, analysis and measurement of software structure. It was carried out at South Bank Polytechnic's Centre for Systems and Software Engineering in London, UK. Among others, results of this project can be found in Fenton /FENT91/. In this project mostly software measures based on the prime decomposition were considered /FENT91/, /ZUSE91/, p.296.

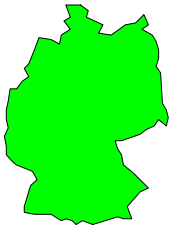
From 1989 till 1992, the Project METKIT (Metrics Educational Toolkit 2384) /METK93/ of the European Community was created. METKIT was a collaborative project part-funded by the European Commission under their ESPRIT programme. The aim of METKIT was to raise awareness and increase usage of software measures within European industry by producing educational material aimed to both industrial and academic audiences. An outcome of METKIT was the book of Fenton /FENT91/

which gives an excellent overview of the area of software measures. Other ESPRIT Project dealing with software engineering measurement were: AMI from Nov. 1990-92 (Applications of Metrics in Industry), MUSIC from Nov. 1990-93 (Metrics for Usability Standards in Computing), MUSE from 1987-90 (Software Quality and Reliability Metrics for Selected Domains: Safety Management and Clerical Systems), PYRAMID from Oct 1990-92 (Promotion for Metrics), with the following task: Improvement of quality and productivity of European software-intensive systems development and maintenance by the use quantitative methods ... in the application of measures by 1995, COSMOS from February 1989-94 (Cost Management with Metrics of Specification), and MERMAID from October 1988-92 (Metrication and Resource Modelling Aid).

The primary objective of METKIT was to promote the use of measurement throughout Europe. The project developed an integrated set of educational materials to teach managers, software developers and academic students how to use measurement to understand, control and then improve software.

1.8 Software Measurement in Germany

In Germany software measurement activities are on a very low level. Since 1993 a software measurement

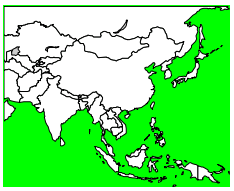


Group exists with GI (Gesellschaft für Informatik). The head of this group is Dr. Dumke. There are only some universities in Germany, where lecture of software measurement are offered. At the Technische Universität Berlin, Bollmann and Zuse offer a lecture of software measurement, in Magdeburg, Dumke teaches, among others, software measurement, and in Kaiserslautern, Rombach invests much effort to increase the quality of software. Education in software measurement for companies in the past were sponsored by Barbara Wix from DECollege, now with ORACLE. From 1989 Zuse in the scope of DECollege gave three times a year 3-day courses in software measurement for companies. Several companies introduced software measurement programs during the last years, examples are Siemens, Bosch, Alcatel, BMW, etc. However, in 1994, the German Government announced a research program, where software measurement is a major part.

1.9 Research in the Area of Software Metrics in USA

Software measurement begun early in the seventieth in US and Canada. In US various software measurement groups and activities has been established since the mid-seventieth. It is not possible to mention all the activities in US. Many measurement programs have been established under the auspices of the Software Engineering Institute (SEI) at the Carnegie Mellon University to provide a platform from which increased used of measurement within organizations can be promoted. A number of major companies use software measures extensively, as AT&T, NASA, Motorola, Hewlett Packard, etc. A long tradition (more than 15 years) of measurement has the SEL-lab in Maryland /NASA81/, /NASA83/. /NASA84/, /NASA84a/, /NASA84b/, /NASA86/, /NASA87/, /NASA89/, /NASA90/. Other activities are described in the other Section of this Chapter.

1.10 Research in the Area of Software Measurement in Japan



It is not known very much from software measurement research in Japan. One exception is the work of Azuma /AZUM92/ in the context of the ISO-norm 9126. Together with this norm more than 100 software measures where proposed to quantify the software quality criteria.

1.11 Desirable Properties for Software Measures

Considering the software life-cycle and software measures many researchers proposed desirable properties for software measures. Such desired properties can be found among others in Fenton /FENT91/, p.218, Weyuker /WEYU88/, Kearney et al. /KEAR86/, and Lakshmanan et al. /LAKS91/. Many of these required properties are contradictory. Some of them are identical with the conditions of the extensive structure in measurement theory /ZUSE91/. A intensive discussion of desirable properties of measures can be found in /ZUSE91/, Chapter 6 /ZUSE92b/ and Chapter 5 of the book.

1.12 Validation of Software Measures and Prediction Models

Validation of software measures is another very important topic in the area of software measures because the acceptance of software measures in practice depends on whether the measures can be used as an predictor for a software quality attribute. Schneidewind /SCHN91/ writes about measure validation: *to help ensure that metrics are used appropriately, only validated metrics (i.e., either quality factors or metrics validated with respect to quality factors) should be used.* Validation of software measures and prediction models are based on the following questions:

1. Is it possible to predict the error-proneness of a system using software measures from its design phase.
2. Is it possible to extract quantitative features from the representation of a software design to enable us to predict the degree of maintainability of a software system.
3. Are there any quantifiable, key features of the program code of a module that would enable us to predict the degree of difficulty of testing for that module, and the number of residual errors in the module after a particular level of testing has occurred.
4. Is it possible to extract quantifiable features from the representation of a software design to enable us to predict the amount of effort required to build the software described by that design.
5. What properties of software measures are required in order to determine the quality of a design.
6. Are there features in order to predict the size of a project from the specification phase
7. What are appropriate software measures to underlie the software quality attributes of the ISO 9126 norm by numbers
8. What are the criteria for the internal and external validation of software measures. What are the criteria for prediction models?

Till the middle of the eighties mostly intra-modular software measures (Measures which are applied to modules or programs) were applied in the coding phase of the software life-cycle. From the middle of the eighties more attention has been directed to the measurement in the early phases of the software life-cycle, like the design phase. Software measures should be applied in every phase of the software life-cycle. The idea of applying software measurement in the early phases of the software life-cycle is to improve software development in the software design phase by a feedback process controlled by software measures in order to get a better implementation of software in the coding phase and a less complicated and less expensive maintenance.

In the middle of the eighties, researchers, like Kafura et al. /KAFU88/ and the NASA /NASA84/ tried to verify the hypothesis that there exists a correlation between software measures and development data (such as errors and coding time) of software. In order to verify this hypothesis they used the data from the SEL-Lab /NASA81/. Similar investigations can be found in /NASA84/ and /NASA86/. Rombach /ROMB90/ summarized some of the results doing measurement in the early phases of the software life-cycle. He points out that there is, using the Spearman correlation coefficient, a high correlation (0.7, 0.8) between the architectural design documents and the maintainability (factors of maintainability). Architectural design can be captured with "architectural measures", (inter-modular measures) like system design and modularity measures. There is a low correlation between "algorithmic measures" (intra-modular measures: code complexity, like McCabe) and maintainability. There is also a high correlation (0.75, 0.8, Spearman correlations) between "hybrid measures" (algorithmic and architectural measures). Other studies can be also found in /CONT86/, Chapter 4.

From 1989 the use of factor analysis in order to analyze the properties of existing software complexity measures and new dimensions of program complexity were used by Munson et al. /MUNS89/ and Coupal et al. /COUP90/.

We have to distinguish between internal and external validation of a software measure /BIEM92/, p.39. Internal validation considers the homomorphism, that means: does the measure what the user wants. External validation of a measure means whether the measure has any relationship to a software quality attribute (external variable). Internal validation of measures are considered by /SHEP91/, /SHEP93/ using an algebraic validation and Zuse /ZUSE91/ using so-called atomic modifications which can be seen as necessary conditions for the ordinal scale.

A widely used concept of external validation of software measures related to an external attribute is the calculation of correlations. Early investigations were done by Dunsmore et al. /DUNS77/, by Curtis et al. /CURT79/ who made predictions of programmers performance with software measures, Basili et al. /BASI81/, /BASI82/, /BASI83/, /BASI84/, /BASI84a/ who made empirical investigations of software measures related to an external attribute, like errors, Hamer et al. /HAMA82/ who investigated the Halstead measures, Kafura et al. /KAFU85/ who made investigations with the Metrics LOC, the Metrics of Halstead and the Metric of McCabe, Ince et al. /INCE89/ who investigated the information flow metric. Other investigations can be found in /SHEN83/, /JENS85/, /KHOS90/, /HENR90/, /LI87/, /LIND89/, /SCHN91/, /KHOS92a/, /VALE92/, and /SCHN91a/. The results of these investigations were contradicting.

Cost estimation models were validated by Kemerer /KEME87/ and /KEME93/,

In 1993 and 1994 Bollmann et al. /BOLL93/ and Zuse /ZUSE94a/ considered prediction models of software measures and the validation of software measures from a measurement theoretic view, showing that the validation of software measures depends on the scale type of the external variable. The authors showed that the (basic) COCOMO-model is the only one which can exist between two ratio scales (the measurement values and the external variable are considered as ratio scales). Zuse et al. also showed that *wholeness: (The whole must be at least as big as the sum of the parts)* is a pseudo-property without any empirical meaning Wholeness is only a numerical modification of a measure without changing the empirical meaning. This result is important for the validation of measures and in the context of prediction models. The authors discussed the consequences whether the cost or time for maintenance can be determined from the components of the software system, too.

In 1993 appeared the Standard 1061 of IEEE /IEEE93/ /SCHN91/ which gives rules/ideas how to validate software measures. the Standard IEEE 1061 has been written by Schneidewind and covers terms like

discriminate power, tracking, validation, predictability, consistency. Important to notice is that software measures are not validated for ever, the have to re-validated in a continuous process.

1.13 Software Measures in an Object-Oriented Environment

At the end of the eighties software measures for the object-oriented environment (OO-Measures) were proposed. A very early investigation of OO-Measures can be found by Rocacher /ROCA88/. In 1989 Morris /MORR89/ discussed software measures for an object-oriented application, Bieman /BIEM91/ discussed software measures for software reuse in an object-oriented, environment, Lake et al. /LAKE92/ discussed measures for C++ applications, Chidamber et al. /CHID93/ evaluated different Smalltalk applications, Sharble et al. /SHAR93/ discussed measures for an object-oriented design (OOD), Li and Henry /, /LI93a/ evaluated ADA-Programs, Chen and Lu /CHEN93/ evaluated OO-Measures related to the OOD-method of Booch /BOOC91/. Karner /KARN93/ wrote a master thesis of measurement in an object-oriented environment. Other papers of this area are /CALD91/, /JENK93/, /JENS91/, /LARA90/, /RAIN91/, /BARN93/, /TEGA92/, /TEGA92a/, /ABRE93/, /WHIT92/, /LAKE94/ and /TAYL93/.

An interesting investigation of object-oriented measures presented in 1994 Cook /COOK94/. He used factor analysis in order to figure out major factors in object-oriented programs. Last not least, we should mention the first book about object-oriented software metrics by Lorenz et al. /LORE94.

In 1995 Zuse /ZUSE95a/ and Fetcke /FETC95/ investigated the structures and the behavior of object-oriented software measures. The result is that object-oriented measures mostly do not assume an extensive structure. In order to characterize object-oriented software measures above the ordinal scale level, Zuse et al. used the Dempster Shafer function of belief the Kolmogoroff axioms, and the De Finetti axioms.

However, in the area of object oriented systems it is not clear what an object oriented program makes difficult to understand, to test or to maintain.

1.14 Data Dependency Measurement

Most of research directed towards measuring the complexity of the program programmers interface has centered on the measurement of control flow complexity (MCCA76/. Weiser /WEIS82/ found that programmers seem to work backwards when debugging, examining only instructions that affect the variables in error. Weiser's result demonstrated the importance of data dependencies and indicate that a measure of data dependency would be a useful tool for the development of reliable software. Bieman /BIEM84/ gives a good overview of data dependency measurement.

1.15 Entropy Measures

Information theory to software measures has been applied the first time in 1972 /HELL72/. Information theory based measures have been applied to most phases of the software life-cycle. However, there are existing relatively few papers. Entropy measures are another kind of measures which are sometimes used. One of the few proposals are from Coulter /COUL87/, from Chen /CHEN78/, /BERL80/, /MOHA81/, and a collection of these measures can be found in /SAMA87/.

1.16 Software Measures for Distributed Systems

Software measures for distributed and real-time systems are discussed by Kodres /MOHA79/, /KODR78/, /ROSE92/, /SHAT86/, /SHAT88/ and /DAME92/.

1.17 Neuronal Networks and Software Measures

In 1994 Khoshgoftaar et al. /KHOS94/ presented a neural network model to classify program modules as either high- or low-risk based upon multiple criterion variables.

1.18 National and International Conferences in the Software Measurement Area

Software measurement is since years a major topic on conferences and workshops. We mention some of them:

1. International Conference on Software Engineering (ICSE).
2. International Workshop on Software Reliability Engineering (ISSRE).
3. IEEE Software Metrics Symposium (firstly 1993).
4. International Software Engineering Standards Symposium (ISESS).
5. International Conference on Software Reusability (ICSR)
6. Workshop on Program Comprehension.
7. International Conference on Software Maintenance (ICSM), about 200 - 250 people
8. International Conference on Software Quality (ICSQ), ca. 250-300 Teinehmer).
9. Software Quality Week, San Francisco, about 500-600 people.
10. NASA Software Engineering Workshop, about. 800 people
11. Annual Oregon Workshop on Software Metrics (AOWSM). Since 1989.
12. International Software Engineering Standards Symposium (ISESS 95) and Forum, Montreal, 1995.
13. First International Software Metrics Symposium (IEEE), May 21-22, Baltimore/USA, 1993.
14. und 2. Software Metric Symposium of DECollege and *Software Metriken 94* of ORACLE (1992, 1993 und 1994 in München).
15. On July 1, 1994, the new Technical Council on Software Engineering (TCSE) was chartered by the Technical Activities Board of the IEEE Computer Society as the first of a new breed of organization fo innovative programs and services. Among, others, a Committee on Quantitative Methods were founded.

1.19 Software Measurement Tools

With the increasing importance of the area of quantitative methods (i.e. software measurement,..) many tools appeared on the market. However, there is a confusing situation. Since no standardized set of measures exist, every maker of a tool has its own software measure set. A good overview of tools can be find in /DAIC95/. Daich et al. also characterizes measurement tools by *tool taxonomy terms*. We list them here:

Universal Metrics Tool:
Niche Metrics Tool:
Static Analysis:
Source code static analyzer:
Size Measurer:

I

1.20 Software Measures and Reengineering

Software measurement plays an essential role in the Reengineering discipline. An overview can be found in Arnold /ARNO92/, /ARNO92a/, and /ARNO92b/.

1.21 ISO 9000-3 and Software Measurement

ISO 9000 is a written set of standards for quality created by the International Organization for Standardization. Certification does not guarantee a level of product quality; instead, it indicates that a company has documented quality practices and procedures in place. This holds true for software as well. Registration does not guarantee that the software is problem-free, only that the vendor has a process in place to correct errors or provide missing function

The American variant of ISO 9000 is known as ANSI/ASQC Q9000 or Q90.00. This standard is jointly sponsored by the American Society of Quality Control (ASQC), a trade organization, and the American National Standards Institute (ANSI), the U.S. voting representative to ISO.

In Section 6.4 in the ISO 9000-3 norm we can find about Product and Process Measurement: *Product Measurement. Metrics should be reported and used to manage the development and delivery process and should be relevant to the particular software product. There are currently no universally accepted measures of software quality. However, at a minimum, some metrics should be used which represent reported field failures and/or defects from the customer's viewpoint. Selected metrics should be described such that results are comparable.*
The supplier of software products should collect and act on quantitative measures of the quality of these software products. These measures should be used for the following purposes:

a) to collect data and report metric values on a regular basis.
b) To identify the current level of performance of each metric,
c) to make remedial action if metric levels grow worse or exceed established target levels,
d) to establish specific improvement goals in terms of the metrics.

In Section 6.4 we can find about process measurement the following statements: *The supplier should have quantitative measures of the quality of the development and delivery process. These measures should reflect:*

a) how well the development process is being carried out in terms of milestones and in-process quality objectives being met on schedule.
b) how effective the development process is at reducing the probability that faults are introduced or that any faults introduced go undetected
.
The choice of metrics should fit the process being used and, if possible, have a direct impact on the quality of the delivered software. Different metrics may be appropriate for different software products produced by the same supplier.

1.22 Cognitive Processes and Measures

The complexity of the programmer / program interface depends upon the amount of mental energy required for a programmer to correctly affect or modify a program. Cognitive activities has been described by Newell, Simon and Brooks /NEWE72/, /BROO77/. The most salient feature of Newell's model is the limitation of human short memory. Miller claims /MILL56/ that short memory is limited to 7 ± 2 chunks of information /MILL56/ and Stroud notes that humans are limited to eighteen mental discriminations per second /STRO67/. Curtis argues that the Stroud and Miller numbers are relevant only for simple tasks and substantially lower for the complex cognitive processes involved in programming /CURT80/.

1.23 Future

It is no question that software measurement is an important method in order to get higher quality of software. Dieter Rombach, who was working at this time with the Software Engineering Laboratory (SEL) in the USA, said at the Eurometrics 1991 in Paris: *we should no longer ask if we should measure, the question today is how.* .

Although in the past much research has been done in the area of software measurement, there are many open questions. Firstly, there is still a lack of maturity in software measurement. Secondly, there is still no standardization of software measures. The proposed software measures in /IEEE89/ are not widely accepted. Validation of software measures in order to predict an external variable is still a research topic for the future. Calculations of correlations and regression analysis require a discussion of measurement scales.

In the future, theory building (hypotheses about reality) becomes more and more important. The axiom systems of measurement theory can help here to get a better understanding what behind software quality and cost estimation is hidden.

Literature

/ABRE93/ Abreu, F.B.:
Metrics for Object-Oriented Environment. Proceedings of the Third International Conference on Software Quality, Lake Tahoe, Nevada, October 4-6, 1993, pp. 67-75.

/ALBR79/ Albrecht, A.J.:
Measuring Applications Development Productivity. Proceedings of IBM Applic. Dev. Joint SHARE/GUIDE Symposium, Monterey, CA, 1979, pp.83-92.

/ALBR83/ Albrecht A.J.; Gaffney, S.H.:
Software Function, Source Lines of Code and Development Effort Prediction: A Software Science Validation. IEEE Transactions of Software. Engineering Volume 9, No. 6, 1983, pp. 639-648.

/ARNO92/ Arnold, Robert, S.:
Software Reuse and Reengineering. In: Arnold, Robert, S.: Software Reegineering. IEEE Computer Society, 1992, pp. 476-484.

/ARNO92a/ Arnold, Robert, S.:
Software Reegineering. IEEE Computer Society, 1992.

/ARNO92b/ Arnold, Robert, S.:

A Road Map Guide to Software Reengineering Technology. In: Arnold, Robert, S.: Software Reengineering. IEEE Computer Society, 1992, pp. 3-22.

/BAIL81/ Bailey, C.T.; Dingee W.L.:
A Software Study Using Halstead Metrics. Bell Laboratories Denver, CO. 80234, 1981

/BAKE87/ Baker, A.L.; Bieman, J.M.; Gustafson, D.A.; Melton, A.; Whitty, R.A.:
Modeling and Measuring the Software Development Process. Proceedings of the Twentieth Annual International Conference on System Sciences, 1987, pp. 23-29.

/BAKE90/ Baker, A.L.; Bieman, J.M.; Fenton, N.; Gustafson, D.A.; Melton, A.; Whitty, R.A.:
A Philosophy for Software Measurement. The Journal of Systems and Software, Volume 12, No 3, 1990, pp.277-281.

/BARN93/ Barns, Michael, G.:
Inheriting Software Metrics. Journal of Object-Oriented Programming, November-December 1993, pp. 27-34.

/BASI75/ Basili, V; Turner, A.J.:
Iterative Enhancement: a Practical Technique for Software Development. Transactions on Software Engineering, Volume SE-1, No. 4, December 75, pp. 390-396.

/BASI79/ Basili, Victor, R.; Reiter, Robert:
Evaluating Automatable Measures of Software Development. Proceedings of Workshop on Quantitative Software Models, pp. 107-116, 1979.

/BASI81/ Basili, V.; Reiter, R.W.
A Controlled Experiment Quantitatively Comparing Software Development Approaches. IEEE Transactions on Software Engineering Volume SE-7, No. 3, May 1981, pp. 299-320.

/BASI82/ Basili, Victor, R.; Selby, Richard, W.; Phillips, Tsai-Yun:
Metric Analysis and Data Validation Across FORTRAN Projects. Technical Report TR-1228, Department of Computer Science, University of Maryland, 1982. Also in: Software Engineering Laboratory Series SEL-83-003, November 1983 /NASA83/.

/BASI83/ Basili, V.; Hutchens, D.:
An Empirical Study of a Complexity Family. IEEE Transactions on Software Engineering, Volume 9, No. 6, November 1983, pp. 664-672.

/BASI84/ Basili, V.; Perricone, Barry T.:
Software Errors and Complexity: An Empirical Investigation. Communications of the ACM, Volume 27, No. 1, January 1984, pp. 42-52.

/BASI84a/ Basili, V.; Weiss, D.:
A Methodology for Collecting Valid Software Engineering Data. IEEE Transactions on Software Engineering, SE-10, No. 6, pp. 728-738, 1984..

/BASI87/ Basili, V.; Rombach, Dieter H.:
TAME: Integrating Measurement into Software Environments. TR-1764 ; TAME-TR-1-1987

/BELA79/ Belady, L.A.

On Software Complexity In: Workshop on Quantitative Software Models for Reliability. IEEE No. TH0067-9, New York, N.Y., pp.90-94, October 1979.

/BERL80/ Berlinger, Eli:
An Information Theory Based Complexity Measure. Proceedings of the National Computer Conference, 1980, pp.773-779.

/BIEM84/ Bieman, J.M.:
Measuring Software Data Dependency Complexity. Ph.D. Thesis, University of Louisiana, 1984.

/BIEM91/ Bieman, J.M.:
Deriving Measures of Software Reuse in Object Oriented Systems. Technical Report #CS91-112, July 1991, Colorado State University, Fort Collins/ Colorado, USA.

/BIEM92/ Bieman, J.M.; Schultz, J.:
An Empirical Evaluation (and Specification) of the all-du-paths. Testing Criterion. Software Engineering Journal, Volume 7, No. 1, pp. 43-51, January 1992.

/BOEH81/ Boehm, B.W.:
Software Engineering Economics. Prentice Hall, 1981

/BOEH84/ Boehm, B.W.:
Software Engineering Economics. IEEE Transactions on Software Engineering 10(1), pp. 7-19, 1984. Also in: Sheppard, Martin (Editor): Software Engineering Metrics - Volume I: Measures and Validations. McGraw Hill Book Company, International Series in Software Engineering, 1993, pp. 112-136.

/BOLL81/ Bollmann, P.; Cherniavsky, V.S.:
Measurement-Theoretical Investigation of the MZ-Metric. In: R.N. Oddy, S.E. Robertson, C.J. van Rijsbergen, P.W. Williams (ed.) Information Retrieval Research, Butterworth, 1981

/BOLL84/ Bollmann, Peter:
Two Axioms for Evaluation Measures in Information Retrieval. Research and Development in Information Retrieval, ACM, British Computer Society Workshop Series, pp. 233-246, 1984

/BOLL85/ Bollmann, Peter; Zuse, Horst:
An Axiomatic Approach to Software Complexity Measures. Proceedings of the Third Symposium on Empirical Foundations of Information and Software Science III, Roskilde, Denmark, October 21-24, 1985. Reprinted in: Empirical Foundations of Information and Software Science III, Edited by Jens Rasmussen and Pranas Zunde, Plenum Press, New York and London, 1987, pp.13-20.

/BOLL93/ Bollmann-Sdorra, P.; Zuse, H.:
Prediction Models and Software Complexity Measures from a Measurement Theoretic View. Proceedings of the 3rd International Software Quality Conference, Lake Tahoe, Nevada, October 4-7, 1993.

/BOLO88/ Boloix, Germinal; Robillard, Pierre:
Inter-Connectivity Metric for Software Complexity. Information Systems and Operation Research, Volume 26, No. 1, 1988, pp. 17-39.

/BOOC91/ Booch, G.:
Object-Oriented Design with Applications. Benjamin/Cummings, 1991.

/BOWL83/ Bowles, Adrian John:
Effects of Design Complexity on Software Maintenance. Dissertation 1983, Northwestern University, Evanston, Illinois.

/CALD91/ Caldiera, G.; Basili, V.:
Identifying and Quantifying Reuseable Software Components. IEEE Software, Feb. 1991, pp. 61-70. Also in: Arnold, Robert, S.: Software Reengineering. IEEE Computer Society, 1992, pp. 485-494..

/CARD90/ Card, David N.; Glass, Robert L.:
Measuring Software Design Quality. Prentice Hall, Engewood Cliffs, New Jersey, 1990

/CHAP79/ Chapin, N.:
A Measure of Software Complexity. Proc of the AFIPS National Computer Conference, Spring 1979, pp.995-1002.

/CHEN78/ Chen, E.:
Program Complexity and Programmer Productivity IEEE Transactions on Software Engineering Volume SE-4, pp. 187-194, May 1978.

/CHEN93/ Chen, J.Y.; Lu, J.F.:
A new Metric for Object-Oriented Design. Journal of Information and Software Technology, Volume 35., No. 4, April 1993, pp.232-240.

/CHID93/ Chidamber, Shyam, R.; Kemerer, Chris, F.:
A Metrics Suite for Object Oriented Design. M.I.T. Sloan School of Management, E53-315, 30 Wadsworth Street, Cambridge, MA 02139, CISR Working paper No. 249, Revised Version of February 1993, December 1993, 31 pages.

/CHRI81/ Christensen, K.; Fitsos, G.P.; Smith, C.P.:
A Perspective on Software Science. IBM Systems Journal, Volume 20, No. 4, pp. 372-387, 1981.

/CONT86/ Conte, S.D.; Dunsmore, H.E.; Shen, V.Y.:
Software Engineering Metrics and Model. Benjamin/Cummings Publishing Company, Menlo Park, 1986.

/COUL87/ Coulter, N.S.; Cooper, R.B.; Solomon, M/K.:
Information-Theoretic Complexity of Program Specifications. The Computer Journal, Volume 30, No. 3, 1987, pp. 223-227.

/COUP90/ Coupal, Daniel; Robillard, Pierre:
How meaningful are Software Metrics? In: Quality Engineering Workshop, Le Centre Sheraton, Montreal, October 4-5, 1990.

/CURT79/ Curtis, B.:
In Search of Software Complexity. In: Workshop on Quantitative Software Models for Reliability, PP.95-106, 1979.

/CURT80/ Curtis, Bill:
Measurement and Experimentation in Software Engineering. Proceedings of the IEEE, Volume 68, No. 9, September 1980, pp. 1144-1157.

/DAIC95/ Daich, Gregory, T.; Price, Gordon; Dawood, Mark:

Metric Tools: Size. CROSSTALK, April 1995, pp. 21-25.

/DAME92/ Damerla, S.; Shatz, Sol, M.:
Software Complexity and ADA Rendevous: Metrics Based on Nonterminsm. Journal of Systems and Software, Volume 17, 1992, pp. 117-127.

/DEMA82/ DeMarco, Tom:
Controlling Software Projects - Management, Measurement and Estimation. Englewood Cliffs, N.J.: Prentice Hall, 1982.

/DEMA87/ DeMarco, Tom; Buxton, John:
The Craft of Software Engineering. Addison Wesley Publishing Company, 1987.

/DEMI81/ DeMillo, Richard A.; Lipton, Richard J.:
Software Project Forecasting. In /PERL81/, pp.77-94, 1981.

/DHAM94/ Dhama, Harpal:
Quantitative Models of Cohesion and Coupling Software. AOWSM (Annual Oregon Workshop on Software Metrics, April 10-12, 1994, Silver Falls State Park, Oregon, 1994.

/DUMK92/ Dumke, Reiner:
Softwareentwicklung nach Maß- Schätzen - Messen - Bewerten. Vieweg Verlag, 1992.

/DUNS77/ Dunsmore, H.E; Gannon J.D.:
Experimental Investigation of Programming Complexity. Proceedings ACM-NBS 16th Annual Tech. Symp.: System Software, Washington, DC, June 1977, pp. 117-125.

/EJIO91/ Ejiogu, L.:
Software Engineering with Formal Metrics. QED Technical Publishing Group, 1991.

/ELLI88/ Elliott, J.J (Editor); Fenton, N.E.; Linkman, S.; Markham:
Structure-Based Software Measurement. Alvey Project SE/069, 1988, Department of Electrical Engineering, South Bank, Polytechnic, 103 Borough Road, London, SE1 OAA, UK.

/EMDE71/ Van Emden, M.H.:
An Analysis of Complexity. Mathematisches Zentrum, Amsterdam, 1971

/EMER84a/ Emerson, Thomas J.:
Program Testing, Path Coverage, and the Cohesion Metric: IEEE COMPSAC, 1984, pp. 421-431

/EMER84b/ Emerson, Thomas J.:
A Discriminant Metric for Module Comprehension. 7th International Conference on SW-Engineering 1984, pp.294-431.

/FENT90/ Fenton, Norman; Melton, A:
Deriving Structurally Based Software Measures. Journal of Systems and Software, 12(3), pp. 177-187, July 1990.

/FENT91/ Fenton, N.:
Software Metrics: A Rigorous Approach, Chapman & Hall, 1991.

/FENT91a/ Fenton, Norman:
The Mathematics of Complexity in Computing and Software Engineering. In: The Mathematical Revolution Inspired by Computing. J.H. Johnson & M.J. Looms (eds), 1991, The Institute of Mathematics and its Applications, Oxford University Press.

/FETC95/ Fetcke, Thomas:
Software Metriken bei der Object-orientierten Programmierung. Diploma thesis, Gesellschaft f³r Mathematik und Datenverarbeitung (GMD), St. Augustin, and TU-Berlin, 1995.

/FREI79/ Freidman, F.R.; Park, R.E.:
Price-S Software Model Overview. Internal paper, RCA, Cherry Hill, 1979.

/GILB77/ Gilb, T.:
Software Metrics. Winthrop Publishers, Cambridge, Massachusetts, 1977.

/GOOD92/ Goodman, Paul:
Practical Implementation of Software Metrics. McGraw Hill Company, 1992.

/GRAD87/ Grady, Robert B.; Caswell, Deborah L.:
Software Metrics: Establishing a Company-Wide Program. Prentice Hall 1987.

/GRAD87a/ Grady, Robert B.:
Measuring and Managing Software Maintenance. IEEE Software, September 1987, pp. 35-45.

/GRAD90/ Grady, Robert B.:
Work-Product Analysis: The Philosopher's Stone of Software. IEEE Software, March 1990.

/GRAD92/ Grady, Robert B.:
Practical Software Metrics for Project Management and Process Improvement. Prentice Hall 1992

/HALL83/ Hall, Nancy R.:
Complexity Measures for Systems Design. Doctoral Dissertation, Dept. of Mathematics. Polytechnic Institute New York, June 1983.

/HALL84/ Hall, N.; Preiser, S.:
Dynamic Complexity Measure for Software Design. IEEE Computer Society, 1109 Spring Street, Suite 200, Silver Spring, MD 20910, USA, 1984.

/HALS76/ Halstead, M.H.; Gordon, R.D.; Elshoff, J.L.:
On Software Physics and GM's PL.I Programs. GM Research Publication GMR-2175, General Motors Research Laboratories, Warren, MI, 1976.

/HALS77/ Halstead, M.H.:
Elements of Software Science. New York, Elsevier North-Holland, 1977.

/HAMA82/ Hamer, P; Frewin, G.:
Halstead's Software Science - A Critical Examination. Proceedings 6th International Conference on Software Engineering, pp. 197-206, 1982

/HARR81/ Harrison, Warren; Magel, Kenneth:

A Complexity Measure Based on Nesting Level. ACM SIGPLAN Notices, Volume 16, No. 3, pp. 63-74, 1981.

/HARR87/ Harrison, Warren; Cook, Curtis:
A Micro/Macro Measure to Software Complexity. The Journal of Systems and Software, No. 7, pp. 213-219, 1987

/HECH77/ Hecht, M.S:
Flow Analysis of Computer Programs. Elsevier, New York, 1977.

/HELL72/ Hellerman, L.:
A Measure of Computational Work. IEEE Transactions on Computers. Volume 21, No. 5, pp. 439-448, 1972.

/HELM66/ Helmer-Heidelberg, O:
Social Technology, Basic Books, New York, 1966.

/HENR81/ Henry, S. M.; Kafura, D. G.:
Software Structure Metrics Based on Information Flow IEEE Transactions on Software Engineering Volume 7, No. 5, 1981, pp. 510-518.

/HENR88/ Henry, S.; Kafura, D.; Mayo, K.; Yerneni, A.; Wake, S.:
A Reliability Model Incorporating Software Quality Factors. TR 88-45, 1988, Department of Computer Science, Virginia Polytechnic, Blacksburg, Virginia, USA.

/HENR90/ Henry, S.; Humphrey, M.:
A Controlled Experiment to Evaluate Maintainablity of Object-Oriented Software. Proceedings Conference on Software Maintenance, 1990, San Diego, CA, November, 26-29, 1990, IEEE Computer Society Press, Los Alamitos, CA, 1990, pp. 37-45.

/HETZ93/ Hetzel, Bill:
Making Software Measurement Work - Building an Effective Measurement Program. QED, 1993.

/HUTC85/ Hutchens, D.; Basili, V.:
System Structure Analysis: Clustering with Data Bindings. IEEE Transactions on Software Engineering, Volume 11, No. 8, August 1985, pp. 749-757.

/IEEE89/ IEEE:
Standard Dictionary of Measures to Produce Reliable Software. The Institute of Electrical and Electronics Engineers, Inc 345 East 47th Street, New York, NY 10017-2394, USA IEEE Standard Board, 1989.

/IEEE89a/ IEEE:
Guide for the Use of Standard Dictionary of Measures to Produce Reliable Software. The Institute of Electrical and Electronics Engineers. Inc 345 East 47th Street, New York, NY 10017-2394, USA IEEE Standard Board, Corrected Edition, October 23, 1989.

/IEEE93/ IEEE Computer Society:
IEEE Standard for a Software Quality Metrics Methodology. IEEE Standard 1061. IEEE Standards Office, 445 Hoes Lane, P.O. Box 1331, Piscataway, NJ 08855-1331.

/INCE89/ Ince, Darrel:

Software Metrics. In: Measurement for Software Control and Assurance. Edited by Kitchenham, B.A.; B. Littlewood, Elevier Applied Science, London and New York, 1989.

/JENK93/ Jenkins, J.:
Software Metrics won't eliminate the Productivity Crisis. American Programmer, Volume 6, Feb. 1993, pp. 2-5.

/JENS85/ Jensen, Howard A.; Vairavan, K.:
An Experimental Study of Software Metrics for Real-Time Software. IEEE Transactions on Software Engineering, Volume SE-11, No. 2, Feb. 1985, pp. 231-234.

/JENS91/ Jensen, R.; Barteley, J.:
Parametric Estimation of Programming Effort: An Object-Oriented Model. Journal of Systems and Software, Volume 15, 1991, pp. 107-114.

/JONE77/ Jones, Capers:
Program Quality and Programmers Productivity. IBM Technical Report TR 02.764, 1977, pp. 42-78. Also in: Tutorial on Programming Productivity: Issues for the Eighties, IEEE Computer Society, Second Edition, 1986.

/JONE78/ Jones, Capers:
Measuring Programming Quality and Productivity. IBM Systems Journal, Volume 17, No. 1, 1978. Also in: Tutorial on programming Productivity: Issues for the Eighties, IEEE Computer Society, Second Edition, 1986.

/JONE79/ Jones, Capers:
A Survey of Programming Design and Speicification Techniques. Proceedings of Specifications of Reliable Software, April 1979, pp. 91-103. In: Tutorial on Programming Productivity: Issues for the Eighties, IEEE Computer Society, Second Edition, 1986.

/JONE88/ Jones, Capers:
A Short History of Function Points and Feature Points. Software Productivity Research Inc., Technical paper, Cambridge, Mass., 1988.

/JONE91/ Jones, Capers:
Applied Software Measurement: Assuring Productivity and Quality, McGraw Hill, New York, NY, 1991.

/KAFU85/ Kafura, Dennis:
A Survey of Software Metrics. Proceedings 1985 Annual Conference of the ACM, Denver, Colorado, October 14-16, ACM Press, New York, NY 1985, pp. 502-506.

/KAFU88/ Kafura, Dennis; Canning, James:
Using Group and Subsystem Level Analysis to Validate Software Metrics on Commercial Software Systems. TR 88-13, 1988, Polytechnic, Blacksburg, Virginia, USA.

/KARN93/ Karner, Gustav:
Metrics for Objectory. Master Thesis at the Link³ping University, S-581 83 Link³ping, Sweden, 1993.

/KEAR86/ Kearney, Joseph K.; Sedlmeyer, Robert L.; Thompson, William B.; Gray, Michael A.; Adler, Michael, A.:

Software Complexity Measurement. Communications of the ACM Volume 29, No. 11, 1986, pp. 1044-1050.

/KELV91/ Kelvin, W.T.:
Popular Lectures and Addresses. 1891-1894.

/KEME87/ Kemerer, Chris F.:
An Empirical Validation of Software Cost Estimation Models. Communications of the ACM, Volume 30, No. 5, pp. 461-429, 1987.

/KEME93/ Kemerer, Chris F.:
Reliability of Function Points Measurement - A Field Experiment. Communications of the ACM, February 1993.

/KHOS90/ Khoshgoftaar, Taghi, M.; Munson, John:
The Lines of Code Metric as a Predictor of Program Faults: A Critical Analysis. COMPSAC 1990, pp. 408-413.

/KHOS92/ Khoshgoftaar, T.M.; Munson, J.C.:
An Aggregate Measure of Program Module Complexity. Annual Oregon Workshop on Software Metrics, March 22-24, 1992, Silver Falls, Oregon, USA.

/KHOS92a/ Khoshgoftaar, T.M.; Munson, J.C.; Bhattacharaya, Bibhuti, B.; Richardsen, Gary, D.:
Predictive Modeling Techniques of Software Quality from Software Measures. IEEE Transactions on Software Engineering, Volume 18, No. 11, November 1992.

/KHOS94/ Khoshgoftaar, T.M.; Allen, Edward, B.:
Applications of Information Theory to Software Engineering Measurement. Technical Report, Department of Computer Science & Engineering, Florida Atlantic University, Boca Raton, FL 33431, USA. Also in: Software Quality Journal, No. 3, 1994, pp. 79-103.

/KITC89/ Kitchenham, B.; Littlewood, B.:
Measurement for Software Control and Assurance. Elsevier, 1989.

/KITC90/ Kitchenham, B.; Linkman, S.J.:
Design Metrics in Practice. Information and Software Technology, Volume 32, No. 4, May 1990.

/KITC90a/ Kitchenham, B.; Linkman, S.J.:
An Evaluation of some Design Metrics. Software Engineering Journal, January 1990, pp. 50-58.

/KODR78/ Kodres, Udo R.:
Discrete Systems and Flowcharts. IEEE Transactions on Software Engineering, Volume SE-4, No. 6, November 1978, pp. 169-179.

/KOLE75/ Kolence, K.W.:
Software Phsics. Datamation, June 1975, pp. 48-51.

/KRAN71/ Krantz, David H.; Luce, R. Duncan; Suppes; Patrick; Tversky, Amos:
Foundations of Measurement - Additive and Polynominal Representation, Academic Press, Volume 1, 1971

/LAEM77/ Laemmel, A.; Shooman, M.:
Statistical (Natural) Language Theory and Computer Program Complexity. POLY/EE/E0-76-020, Dept, of Electrical Engineering and Electrophysics, Polytechnic Institute of New York, Brooklyn, August 15, 1977.

/LAKE92/ Lake, Al:
A Software Complexity Metric for C++. Annual Oregon Workshop on Software Metrics, March 22-24, 1992, Silver Falls, Oregon, USA.

/LAKE94/ Lake, Al:
Use of Factor Analysis to Develop OOP Software Complexity Metrics. Annual Oregon Workshop on Software Metrics, April 10.12, 1994, Silver Falls, Oregon, USA.

/LAKO93/ Lakhotia, Arun:
Ruled-based Approach to Compute Module Cohesion. In Proceedings of the 15th International Conference on Software Engineering, pp. 35-44, 1993.

/LAKS91/ Lakshmanan, K.B.; Jayaprakash, S.; Sinha, P.K.:
Properties of Control-Flow Complexity Measures. IEEE Transactions on Software Engineering, Volume 17, No.12, December, 1991, p.1289-1295.

/LARA90/ Laranjeira, L.:
Software Size Estimation of Object-Oriented Systems. IEEE Transactions on Software Engineering, May 1990, pp. 510-522.

/LEMM88/ Lemmerich, Jost:
Maß und Messen, Austellungskatalog aus Anlaßder Gr³ndung der Physikalisch-Technischen Reichsanstalt am 28. März 1887. Braunschweig and Berlin, Physikalisch-Technische Bundesanstalt, 1987.

/LIGI89/ Ligier, Yves:
A Software Complexity Metric System Based on Intra- and Inter-modular Dependencies. IBM RC 14831 (#65457) 5/11/89.

/LIND89/ Lind, Randy, K.; Vairavan, K.:
An Experimental Investugation of Software Metrics and Their Relationship to Software Development Effort. IEEE Transactions on Software Engineering, Volume 15, No. 5, pp. 649-653, 1989.

/LONG86/ Longworth, H.D.; Ottenstein, L.M.; Smith, M.R.:
The Relationship between Program Complexity and Slice Complexity During Debugging Tasks. IEEE COMPSAC, October 1986, pp. 383-389.

/LUCE90/ Luce, R. Duncan; Krantz, David H.; Suppes; Patrick; Tversky, Amos:
Foundations of Measurement. Volume 3, Academic Press, 1990

/MAYR90/ Mayrhauser, Anneliese von:
Software Engineering - Methods and Management. Academic Press, Inc., 1990.

/MCCA76/ McCabe, T.:
A Complexity Measure. IEEE Transactions of Software Engineering, Volume SE-2, No. 4, pp. 308-320, December 1976.

/MCCA89/ McCabe, T; Butler, Charles W.:

Desgin Complexity Measurement and Testing. Communications of the ACM, Volume 32, No. 12, Dec 89, pp. 1415-1424.

/MCCL78/ McClure, Carma L.:
A Model for Program Complexity Analysis. 3rd International Conference on Software Engineering, May 1978, pp. 149-157.

/METK93/ METKIT:
METKIT - Metrics Educational Toolkit. Information and Software Technology, Volume 35, No. 2, February 1993.

/MILL56/ Miller, G.A.:
The Magic Number Seven, Plus or Minus Two. Psychological Review, 63, 1956, pp. 81-97.

/MILL88/ Mills, Everald, E.:
Software Metrics. SEI Curriculum Module SEI-CM-12-1.1, December 1988, Software Engineering Institute, Pittsburg, PA, USA.

/MOEL93/ M³ller, K.H.; Paulish, D.J.:
Software Metrics. Chapman & Hall, 1993.

/MOHA79/ Mohanty, Siba N.:
A Hypergraph Oriented Test and Evaluation Model for Real-Time Systems. ACM Computing Surveys, Volume 11, No. 3, pp. 251-275, 1979.

/MOHA81/ Mohanty, Siba N.:
Entropy Metrics for Software Design Evaluation. The Journal of Systems and Software, No. 2, pp. 39-46, 1981.

/MORR76/ Morris, M.F.:
Kolence true of false? Computer World, October 18, 1976.

/MORR89/ Morris, Kenneth, L.:
Metrics for Object-Oriented Software Development Environments. Massachusetts Institute of Technology, Master of Science in Management, May 1989.

/MUNS89/ Munson, J.; Khoshgoftaar, T.:
The Dimensionality of Program Complexity. Proceedings of the 11th Annual International Conference on Software Engineering, May 1989, pp. 245-253.

/MYER76/ Myers, G.L.:
Composite Design Facilities of Six Programming Languages. IBM Systems Journal, No. 3, 1976, pp. 212-224.

/NASA81/ National Aeronautics and Space Administration:
Software Engineering Laboratory (SEL),Data Base Organization and User's Guide, Software Engineering Laboratory Series SEL-81-002, September 1981.

/NASA83/ National Aeronautics and Space Administration:
Collected Software Engineering papers: Volume II. Software Engineering Laboratory Series SEL-83-003, November 1983.

/NASA84/ National Aeronautics and Space Administration:
Measures and Metrics for Software Development. Software Engineering Laboratory Series SEL-83-002, March 1984.

/NASA84a/ National Aeronautics and Space Administration:
Investigation of Specification Measures for Software Engineering Laboratory. Software Engineering Laboratory Series SEL-84-003, December 1984.

/NASA84b/ National Aeronautics and Space Administration:
An Approach to Software Cost Estimation. Software Engineering Laboratory Series SEL-83-001, February 1984.

/NASA86/ National Aeronautics and Space Administration:
Measuring Software Design. Software Engineering Laboratory Series SEL-86-005, November 1986.

/NASA87/ National Aeronautics and Space Administration:
Collected Software Engineering papers: Volume V. Software Engineering Laboratory Series SEL-87-009, November 1987.

/NASA89/ National Aeronautics and Space Administration:
Proceedings of the Fouteenth Annual Software Engineering Workshop. Software Engineering Laboratory Series SEL-89-007, November 1989.

/NASA90/ National Aeronautics and Space Administration:
Proceedings of the Fifteenth Annual Software Engineering Workshop. Software Engineering Laboratory Series SEL-90-006, November 1990.

/NELS66/ Nelson, E.A.:
Management Handbook for the Estimation of Computer Programming Costs. AD-A648750, Systems Development Corp., 1966.

/NEWE72/ Newell, Allen; Simon, Herbert:
Human Problem Solving. Prentice Hall, 1972.

/OVIE80/ Oviedo, Enrique I.:
Control Flow, Data Flow and Programmers Complexity. Proceedings of COMPSAC 80, Chicago IL, pp.146-152, 1980.

/PARK92/ Park, Robert, E.:
Software Size Measurement: A Framework for Counting Source Statements. Software Engineering Institute, Pittsburg, SEI-92-TR-020, 220 pages, May 1992.

/PARN75/ D.L.:
The Influence of Software Structure on Reliability. Proceedings of International Conference on Reliable Software, April 21-23, 1975, pp. 358-362.

/PARR80/ Parr, F.N.:
An Alternative to Rayleigh Norden Curve Model for Software Development Effort. IEEE Transactions on Software Engineering, SE-6, No. 3, May 1980.

/PATE92/ Patel, Sukesh; Chu, William, Baxter, Rich:
A Measure for Composite Module Cohesion. 15th International Conference on Software Engineering, 1992.

/PERL81/ Perlis, Alan; Sayward, Frederick, Shaw, Mary:
Software Metrics: An Analysis and Evaluation The MIT Press, 1981.

/PFLE91/ Pfleeger, S.L.; Fitzgerald, J.C.:
Software Metrics Tool Kit: Support for Selection, Collection and Analysis. Information and Software Technology, Volume 33, No. 7, September 1991, pp. 477-482.

/PIWO82/ Piwowarski, Paul:
A Nesting Complexity Measure. SIGPLAN Notices, pp. 44-50, Volume 17, no. 9, 1982.

/PORT90/ Porter, A.A.; Selby, R.W.:
Empirically Guided Software Development using Metric-Based Classification Trees. IEEE Software, Vol. 7, No. 2, pp. 46-54.

/PRES92/ Pressmann, Roger S.:
Software Engineering: A Practitioner's Approach. Third Edition, McGraw Hill, 1992.

/PUTN78/ Putnam, L.H.:
A General Empirical Solution to the Macro Software Sizing and Estimating Problem. IEEE Transactions of Software Engineering, SE-4 (4), pp. 345-361, July 1978.

/RADC84/ RADC (Rome Air Development Center):
Automated Software Design Metrics, RADC-TR-S4-27, 1984, Air Force System Command, Griffies Air Force Base, NY 13441.

/RAIN91/ Rains, E.:
Function Points in an ADA Object-Oriented Design? OOPS Messenger, ACM Press, Volume 2, No. 4, October 1991,

/RISI92/ Rising, Linda; Callis, Frank, W.:
Problems with Determining Package Cohesion and Coupling. Software Practice and Experience, Volume 22, No. 7, July 1992, pp.553-571.

/ROBE79/ Roberts, Fred S.:
Measurement Theory with Applications to Decisionmaking, Utility, and the Social Sciences. Encyclopedia of Mathematics and its Applications Addison Wesley Publishing Company, 1979.

/ROBI89/ Robillard, Pierre, N.; Boloix, Germinal:
The Interconnectivity Metrics: A New Metric Showing How a Program is Organized. The Journal of Systems and Software 10, 29-39, 1989, pp. 29-38.

/ROCA88/ Rocacher, Daniel:
Metrics Definitions for Smalltalk. Project ESPRIT 1257, MUSE WP9A, 1988.

/ROMB90/ Rombach, D.:
Design Measurement: Some Lessons Learned. IEEE Software, March 1990, pp.17-24.

/ROMB90b/ Rombach, D.:
Benefits of Goal-Oriented Measurement. In Tutorial CSR, 7th Annual Conference on Software Reliability and Metrics, September 1990.

/ROSE92/ Rose, J.:
Rigorous Appraoch for Representing and Measurement of Structural Properties of Concurrent Computer Programs. Information and Software Technology, Volume 34, No. 5, May 1992, pp. 326-332.

/RUBE68/ Rubey, R.J.; Hartwick, R.D.:
Quantitative Measurement Program Quality. ACM, National Computer Conference pp. 671-677, 1968

/RUST81/ Ruston, H.:
Software Modelling Studies: The Polynomical Measure of Complexity. RADC-TR-81-183, Rome Air Development Center, Air Force Systems Command, Griffis Air Force Base, Rome, NY, July 1981.

/SAMA87/ Samadzadeh-Hadidi, Mansur:
Measurable Characteristics of the Software Development Process Based on a Model of Software Comprehension. Dissertation, University of Southwestern Louisiana, USA, May 1987

/SCHN77/ Schneidewind, N.F.:
Modularity Considerations in Real Time Operating Structures. COMPSAC 77, pp. 397-403.

/SCHN91/ Schneidewind, Norman F.:
Validating Software Metrics: Producing Quality Discriminators. In: Proceedings of the Conference on Software Maintenance (CSM91), Sorrento, Italy, October 1991, and in: Proceedings of International Symposium on Software Reliability Engineering, 1991.

/SCHN91a/ Schneidewind, Norman F.:
Setting Maintenance Quality Objectives and Prioritizing Maintenance Work by Using Quality Metrics. In: Proceedings of the Conference on Software Maintenance (CSM91), Sorrento, Italy, October 1991.

/SELB88/ Selby, Richard W.; Basili, V.:
Analyzing Error-Prone System Coupling and Cohesion. Technical Report UMIACS-TR-88-46, Computer Science, University of Maryland, June 1988.

/SELB92/ Selby, Richard, W.:
Interconnectivity Analysis Techniques for Error Localization in Large Systems. Annual Oregon Workshop on Software Metrics (AOWSM), Portland State University, March 22-24, 1992.

/SHAR93/ Sharble, Robert, C.; Cohen, Samuel, S.:
The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods. ACM SIGSOFT SOFTWARE ENGINEERING NOTES, Volume 18, No. 2, April 1993, pp. 60-73.

/SHAT86/ Shatz, S.M.:
On Complexity Metrics Oriented for Distributed Programs. COMPSAC 86, pp. 247-253

/SHAT88/ Shatz, S.M.:
Towards Complexity Metrics for Ada Tasking. IEEE Transactions on Software Engineering, Volume 14, No. 8, August 1988.

/SHEN83/ Shen, V. Yu; Thebaut, S.; Paulsen, L.:

Software Science Revisited: A Critical Analysis of the Theory and its Empirical Support. IEEE Transactions on Software Engineering, Volume 9, No. 3, March 1983.

/SHEP91/ Shepperd, Martin; Ince, Darrel:
Algebraic Validation of Software Metrics. Proceedings of the 3rd European Software Engineering Conference, Milano, October 1991.

/SHEP93/ Shepperd, Martin (Editor):
Software Engineering Metrics - Volume I: Measures and Validations. McGraw Hill Book Company, International Series in Software Engineering, 1993.

/SHEP93a/ Shepperd, Martin; Ince, Darrel:
Derivation and Validation of Software Metrics. Clarendon Press - Oxford., 1993.

/SHOO83/ Shooman, Martin L.:
Software Engineering, McGraw Hill, 1983.

/SOMM92/ Sommerville, Ian:
Software Engineering. Fourth Edition, Addison Wesley, 1992.

/STEV74/ Stevens, W.P.; Myers, G.J.; Constantine, L.L.:
Structured Design. IBM Systems Journal, No. 2, 1974, pp. 115-139.

/SYMO88/ Symons, Charles, R.:
Function Point Analysis: Difficulties and Improvements. IEEE Transactions on Software Engineering, Volume 14, No. 1, January 1988, pp. 2-11.

/SYMO91/ Symons, Charles, R.:
Software Sizing and Estimating. MkII FPA, John Wiley, New York, 1991.

/TAUS81/ Tausworthe, R.C.:
Deep Space Network Software Cost Estimation Model. Publication 81-7, Jet Propulsion Library, Pasadena, CA, 1981.

/TAYL93/ Taylor, D.A.:
Software Metrics for Object Technology. Object Magazine, Mar-Apr. 1993, pp. 22-28.

/TEGA92/ Tegarden, David, P.; Sheetz, Steven, D.; Monarchi, David, E.:
A Software Model of Object-Oriented Systems. Decision Support Systems: The International Journal, 7/1992.

/TEGA92a/ Tegarden, David, P.; Sheetz, Steven, D.; Monarchi, David, E.:
Effectiveness of tradional Software Metrics for Object-Oriented Systems. Proceedings HICSS-92, San Dieago, 1992.

/THUS88/ Thuss, Jeffrey:
An Investigation into Sclice Based Cohesion Metrics. Master Thesis, Michigan Technology University, 1988.

/TROY81/ Troy, Douglas; Zweben, Stuart:

Measuring the Quality of Structured Design The Journal of System and Software. Volume 2, 113-120, 1981, pp.113-120.

/VALE92/ Valette, Veronique; Valee, Frederique:
Software Quality Metrics in Space Systems. In: Proceedings of Third International Symposium on Software Reliability Engineering, October 7-10, 1992, Research Triangle Park, North Carolina, IEEE Computer Society Press, Los Alamitos, California, pp. 296-301.

/WALS77/ Walston, C. E.; Felix, C.P.:
A Method of Programming Measurement and Estimation. IBM Systems Journal, Vol. 16, No. 1, pp. 54-73, 1977. Also in: Tutorial on Programming Productivity: Issues for the Eighties, IEEE Computer Society, Second Edition, 1986.

/WEIS82/ Weiser, M.D.:
Programmers Use Slices When Debugging. Communications of the ACM, 25(7), July 1982, pp. 446-452.

/WEIS84/ Weiser, M.D.:
Program Sclicing. IEEE Transactions on Software Engineering, SE-10, 4, July 1984, pp. 352-357.

/WEYU88/ Weyuker, Elaine J.:
Evaluating Software Complexity Measures. IEEE Transactions of Software Engineering Volume 14, No. 9, Sepr. 88.

/WHIT92/ Whitmire, Scott, A.:
Measuring Complexity in Object-Oriented Software. Third International Conference on Applications of Software Measures (ASM92), November 1992, La Jolla, California. In: Workshops in Computing: T.Denvir, R.Herman and R.Whitty (Eds.): Proceedings of the BCS-FACS Workshop on Formal Aspects of Measurement, South Bank University, London, May 5, 1991. Series Edited by Professor C.J. Rijsbergen. ISBN 3-540-19788-5. Springer Verlag London Ltd, Springer House, 8 Alexandra Road, Wimbledon, London SW19 7JZ, UK, 1992, pp. 116-141..

/WOLV74/ Wolverton, R.W.:
The Cost of Developing Large-Scale Software. IEEE Transactions on Computer, Volume C-23, No. 6, pp. 615-636, June 1974. Also in: Tutorial on Programming Productivity: Issues for the Eighties, IEEE Computer Society, Second Edition, 1986.

/ZUSE85/ Zuse, Horst:
Meßtheoretische Analyse von statischen Softwarekomplexitätsmaßen. TU-Berlin 1985, Fachbereich Informatik, Dissertation im FB 20 (Ph. D. Thesis).

/ZUSE87/ Zuse, Horst; Bollmann, P.:
Using Measurement Theory to Describe the Properties and Scales of Static Software Complexity Metrics. IBM Thomas Watson Research Center Yorktown Heights, RC 13504, 1987.

/ZUSE89/ Zuse, Horst; Bollmann, P.:
Using Measurement Theory to Describe the Properties and Scales of Static Software Complexity Metrics. SIGPLAN Notices, Volume 24, No. 8, pp.23-33, August 89.

/ZUSE91/ Zuse, Horst:
Software Complexity: Measures and Methods. DeGruyter Publisher 1991, Berlin, New York, 605 pages, 498 figures.

/ZUSE92/ Zuse, Horst; Bollmann-Sdorra, Peter:
Measurement Theory and Software Measures. In: Workshops in Computing: T.Denvir, R.Herman and R.Whitty (Eds.): Proceedings of the BCS-FACS Workshop on Formal Aspects of Measurement, South Bank University, London, May 5, 1991. Series Edited by Professor C.J. Rijsbergen. ISBN 3-540-19788-5. Springer Verlag London Ltd, Springer House, 8 Alexandra Road, Wimbledon, London SW19 7JZ, UK, 1992.

/ZUSE92a/ Zuse, Horst:
Properties of Software Design Metrics. Proceedings of the Annual Oregon Workshop on Software Metrics, March 22-24, 1992, Silver Falls, Oregon, USA.

/ZUSE92b/ Zuse, Horst:
Measuring Factors Contributing to Software Maintenance Complexity. Proceedings of the 2nd International Conference on Software Quality, Triangle Research Park, NC, October 4-7, 1992, ASQC (American Society for Quality Control) 611 East Wisconsin Avenue, Milwaukee, Wisconsin 53202, USA, pp. 178-190.

/ZUSE94/ Zuse, Horst:
Foundations of Validation, Prediction, and Software Measures. Proceedings of the AOWSM (Annual Oregon Workshop on Software Metrics), Silver Fall State Park, Oregon, 1994.

/ZUSE94a/ Zuse, Horst:
Software Complexity Metrics/Analysis. Marciniak, John, J. (Editor-in-Chief): Encyclopedia of Software Engineering, Volume I, John Wiley & Sons, Inc. 1994, pp. 131-166.

/ZUSE94c/ Zuse, Horst:
Foundations of the Validation of Object-Oriented Software Measures. In: Theorie und Praxis der Softwaremessung (Dumke, R.; Zuse, H. (Editors), Deutsche Universitätsverlag DUV, Gabler - Vieweg - Westdeutscher Verlag, 1994, pp. 136-214.

/ZUSE95a/ Zuse, Horst; Fetcke, Thomas:
Properties of Object-Oriented Software Measures. Proceedings of the Annual Oregon Workshop on Software Metrics (AOWSM), Silver State Park, June 3-5, 1995.