Preface

The central goal of this textbook is to provide readers with a framework for software measurement from a theoretical, practical and educational view. Software measurement is not a mature science, today. On the one side, there is a lack of a theoretical framework for software measurement, and on the other side, there is a lack of education of scientists, practitioners and students in the area of software measurement. The book was written with the intention to investigate software measurement principles and to give theoretical and practical guidelines for software measurement. We focus on a qualitative interpretation of quantitative results. For example, the models of complexity behind measures, the model of effort behind the COCOMO model, the model behind the Function-Point Method, and models of understandability of object-oriented measures, are discussed. For this investigations we use axiom systems in a descriptive way. These axiom systems give qualitative criteria of measurement of objects in the software engineering area. The final result are hypotheses of reality that support the development of empirical theories related to the quality of software.

By the time the reader reaches the end of this book, scientists, teachers, practitioners, and students should be able to define the basic terminology of software measurement, to explain the definition of a measure; the definition of a homomorphism, the ideas / secrets behind software measures, the empirical conditions behind software measures, the scale types, the use of qualitative and quantitative methods, the role of software measures during the software life-cycle, the idea behind software cost estimation models and the Function-Point method, the exact definitions of validation of software measures, the foundations of prediction models, the use of meaningful statistics, and the reader should be able to explain problems in the area of software measurement.

The necessity of software measurement can be derived from the definition of software engineering by IEEE: *The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.* As indicated by the word *quantifiable*, at least one key professional society has acknowledged measurement as an integral part of a well-conceived software approach, and not merely an adjunct. During the seventies and eighties, software measures mostly were used without a

proper theory. A qualitative interpretation of numbers mostly often was neglected. However, some authors, like Curtis, Dunsmore, Fenton, Bieman, the Grubstake Group, Schneidewind, Henderson-Sellers, Melton, and Weyuker discussed (theoretical) properties for software measures. The goal of these discussions was to formulate standardized properties for software measures and to put software measurement on firm basis. At the beginning of the nineties, more and more scientists required the development of fundamental principles of software measurement. Reasons for that were the difficulty to interpret the meaning of numbers and the confusing situation expressed by more than one thousand software measures of all kinds.

In 1985 the author introduced the use of measurement theory in the area of software measurement. Classic measurement theory, as described in the texts of Krantz et al., Luce et al., and Roberts, was extended to needs of software measurement by the author.

The first discussions of classic measurement principles at the department of computer science at the *Technische Universität Berlin* started in the middle of the seventies by Cherniavsky. The work of Cherniavsky was an important step forward to recognize the connections between a measure and what to measure. The work of Cherniavsky in analyzing measures was proceeded by that of Bollmann, also at the *Technische Universität* Berlin, in the research field of evaluation measures for information retrieval systems.

As a consequence of measurement in the area of information retrieval, the work of the author on software (complexity) measures started in 1981. The result of the research during 1981-1985 was the 1985 dissertation by the author: *Horst Zuse: Meßtheoretische Analyse von Statischen Softwarekomplexitätsmaßen* (Translated by the author*: Horst Zuse: Measurement Theoretic Analysis of Static Software Complexity Measures.*). From 1987 to 1988 the work in software complexity measurement was continued by the author at the *IBM Thomas J. Watson Research Center* in Yorktown Heights, USA. There, Tom Corbi, supported the use of software measurement in the Project PUNDIT (Program UNDerstanding Investigation Tool). The work on software measures was extended and the first version of the software system MDS was developed and implemented. The System MDS was used to demonstrate the use of software measures with a qualitative interpretation. Back at the Technische Universität Berlin, in 1988, the author intensified his research on software measures.

In 1991 the results of this research were presented in the first book of the author: *Horst Zuse: Software Complexity - Measures and Methods, DeGruyter Publisher, 1991.* While the book of 1991 considered exclusively intra-modular software complexity measures, this book focuses on a framework for software measurement from a theoretical, practical and especially an educational view. Only parts of Chapter 4 (measurement theory) of the previous book are also discussed in a very extended form in this book. This book considers measures of the specification, design, coding, testing and maintenance phase. The theoretical concepts and measurement structures are developed for this kind of measures, and especially for

the validation of software measures and prediction models. More than one hundred exercises shall help the reader to get a deeper understanding of software measurement.

Many ideas for this book came from discussions with students in the lecture: *Methoden des Messens für Informatiker (Methods of Measurement in Computer Science*). Also many ideas came from the many discussions on seminars for people of industry in the scope of DECollege and ORACLE, organized by Barbara Wix, in the years 1989-1994, and the many tutorials in North America as presented by the author from 1988 to 1995. From August 1994 to March 1995, the *Gesellschaft für Mathematik und Datenverarbeitung, St. Augustin (German National Research Center for Computer Science) (GMD)* invited the author as a guest researcher. During this time parts of this book were created.

In order to put through the concepts of software measurement to students, teachers and scientists in a more comprehensive form without mathematics, the System ZD-MIS (Zuse / Drabe Measure Information System) has been developed. It is an educational system with a large database of software measures, a database of literature and a glossary of terms in the software measurement area. It is running on a PC. It uses modern multi-media techniques and allows the user to learn theoretical and practical concepts of software measurement by a dialog.

The book was firstly written in WORD6a, then in WORD7.0 and finished in WORD97. The graphics were created with VISIO 3.0/4.0. These software systems were firstly used with WINDOWS 3.11 and then with WINDOWS 95 under an *INTEL PENTIUM 90 / 166* Processor with 32MB main storage. The prints for the book were made on a HP 5L Laserprinter.

Berlin, November 1997 Horst Zuse

Acknowledgments

Writing a book of this type is a long and an expensive procedure. It involves an unbroken thread of research. The red thread of this book is a framework for software measurement in form of a textbook. The work on this book began at the end of the eighties, and was interrupted by many reasons. This book involves the collection of many questions and ideas by people on seminars and tutorials in Europe, North America and Australia, but also the many questions of students, and the many contributions of scientists, friends, etc. Without these many stimuli it would have been impossible to finish this work. This book also includes the reply of the many not investigated questions and research topics of my first book: *Horst Zuse: Software Complexity - Measures and Methods, DeGruyter Publisher, 1991.*

Too many people, like colleagues, friends, students, etc. have contributed to the development of my ideas for me to acknowledge them all individually. However, I would especially like to thank some people and institutions.

I thank Fevzi Belli from the Gesamthochschule Paderborn for recommending this book for publication. He already recommended my 1991's book for publication.

Again, I thank Peter Bollmann-Sdorra for the very helpful and detailed suggestions on my first book, for his continued suggestions and for the very constructive discussions over many years that supported the completion of this book. I first learned about the measurement theory in a lecture given by Peter at the end of the seventies.

Norman Fenton from the City University in London, UK, deserves much thanks. He always supported my work and especially the measurement theoretic framework applied to software measures. Also, Robin Whitty from Southbank University, London, UK, deserves much appreciation for supporting my work and publishing the book: *Denvir, Herman and Whitty (Eds.): Workshops in Computing: Proceedings of the BCS-FACS Workshop on Formal Aspects of Measurement, South Bank University, London, May 5, 1991*, where we had the possibility to publish an article of the foundations of software measurement.

I thank Jim Hemsley and Harry Sneed who asked me in 1989 to participate the ESPRIT II METKIT Project, which did run from 1989 till 1992. In this context I also thank Mike Kelly and Nick Ashley for the many discussions.

I also thank Taz Daughtrey who gave me the possibility to present my work on the International Conferences on Software Quality, (Dayton / Ohio 1991, Triangle Research Park / North Carolina 1992, and Lake Tahoe / Nevada 1993), which were sponsored by the American Society for Quality Control.

I thank Pierre Robillard of Ecole Poytechnic in Montreal / Canada, for the many discussions on software measurement and his care on my visit in 1992 in Montreal.

I am very thankful to Alain Abran (University of Quebec at Montreal), Pierre Bourque (University Quebec of Montreal) and Jean Mayrand (Bell Lab Canada) for their support to give software measurement tutorials in Montreal in 1992 and 1995. From November 27-28, 1995, the Joint Germany-Quebec Software Metrics Workshop took place at the Technische Universität Berlin. The goal of this workshop was to cooperate in several areas of software measurement from theoretical and practical views. In 1996 the 6th Workshop of the *GI-Arbeitsgruppe Softwaremetriken* and the *Germany-Quebec Workshop on Software Metrics* took place in Regensburg.

I thank Victor Basili for the many discussions about the sense, the meaningfulness of scales and the application of measurement theory in software measurement. I also appreciated the many discussions with Norman Schneidewind, John Munson, Taghi Khoshgoftaar, Paul Oman, and Warren Harrison.

Nancy Mead with the Software Engineering Institute (SEI) in Pittsburgh / Pennsylvania, deserves much thanks because she always supported my work.

Especially, I thank Jim Bieman of Colorado State University in Fort Collins / Colorado for the many discussions, for his engagement in measurement theory and his steady requirements for a framework for quantitative methods in software engineering. Since 1994 he is chair of the IEEE Council of Quantitative Methods in Software Engineering. This committee publishes the Q-Methods Newsletter and sponsors the Software Metrics Symposium as part of a growing series of programs to understand and promote quantitative approaches in software engineering.

Tom McCabe deserves my thanks for the many discussions we had about his work in software measurement. I met him the first time in 1988 at the National Security Agency (NSA) in Baltimore / Maryland, where I was invited for a talk about software measurement. In 1994 I met him in München / Germany where we could continue our discussions.

Barbara Wix deserves much thanks for her support of my work. Barbara organized from 1989-1993 in the scope of DECollege / München and from 1994 with ORACLE / München, together with us, many seminars and symposia about software measurement. I think, that Barbara in 1991 organized the first symposium on software measurement in Germany with an attendance of more than seventy people.

I thank Fernando Brito de Abreu for organizing a workshop of object-oriented software measurement in Aarhus / Denmark in August 1995. There, for example, special aspects of object-oriented approaches could be discussed together with Brian Henderson-Sellers, who also works in the area of object-oriented measurement.

I thank the *Gesellschaft für Mathematik und Datenverarbeitung (GMD) in St. Augustin (German National Research Center for Computer Science)* for their support of my work. They invited me from August 1, 1994 till March 31, 1995 as a guest

researcher in the Institute for Software Engineering (ISA). The many discussions during this time helped me to improve my work on software measurement.

I thank the ASMA (Australian Software Metric Association) for their invitation to the *Third Australia Conference on Software Metrics* in Melbourne in November 1996 as a keynote speaker. I also thanks Brian Henderson-Sellers for inviting me in his institute in Hawthorne close to Melbourne.

Herman Flessner from the University of Hamburg deserves much thanks for recommending a lecture of mine in a course of economics in computers science in the area of software quality and measurement at the University of Stettin / Poland in the summer term 1997.

Shortly after the wall between West- and East-Germany on November 9, 1989, was gone, I met Reiner Dumke and Achim Winkler from the *Guericke Universität* in Magdeburg. Reiner Dumke also worked on software measurement since many years, but we did not hear anything of each other. Since this time many cooperations have been established, and, among others,we are organizing many workshops on software measurement in the scope of the GI (Gesellschaft für Informatik). I thank Reiner Dumke for his engagement in software measurement, for his steady support of my work, for the many suggestions, and for reviewing this book. In 1996 R.Dumke, F.Ebert, H.Rudolph, and H.Zuse published the first edition of the Journal: *Metric News - Journal of the GI-Interest Group on Software Metrics. Otto von Guericke University of Magdeburg, Volume 1, Number 1, September 1996*. This journal informs researchers and practitioners about news in the software measurement area.

I thank all the students of the lecture *Methods of Measurement Theory in Computer Science* for the many discussions. Andreas Mennert wrote two theses of software measurement, and in 1993 he was five months with Siemens Corporate Research / New Jersey / USA. There he implemented many measures of my first book under UNIX and analyzed big software systems. Thomas Fetcke followed me to the *Gesellschaft für Mathematik und Datenverarbeitung (GMD)* in 1994, where he wrote a diploma thesis about object-oriented software measures. Thomas joined the group of Alain Abran (University of Quebec at Montreal) as a PhD student for one year from May 1996 to May 1997. He deserves much thanks for reviewing this book together with his colleagues. In 1995 Lutz Dierbach implemented the axiomatic approach of the first book under a PC under WINDOWS 3.11.

Finally, I have to thank very much my colleagues at the Scientific Computer Center *Informatik RechnerBetrieb* (IRB13) at the department of computer science at the *Technische Universität Berlin* for their support to complete this book. I especially thank Dr. Heinz Seidlitz and Siegfried Bürk.

1 Introduction

The perception of measures

and harmony is surrounded
by a peculiar magic.

Carl Friedrich Gauß (1777-1855)

For more than fifty years computers have played a more and more important role in our life. It was estimated that, by 1990, one half of American work force will rely on computers and software to do its daily work. As computer hardware costs continue to decline, the demand for new applications software continues to increase at a rapid rate.

In general, a producer should be interested in the quality of a product, if he produces cars, video recorders, computers, etc. In the area of software we have a similar situation. Several aspects of quality of software were discussed in the past. Almost thirty years ago, software engineering has been established. Much progress improving the quality of software has been made during the last twenty five years. With the size and complexity of software ever on the rise, the software crisis, first mentioned as far back as the 1968 NATO Software Engineering Conference /NAUR69/, is even more apparent now /SHAP97/.

The field of software measurement provides approaches to the quantification of quality aspects of software, related to the product, the process and the resources. The most famous software measures, among others, are lines-of-code, the Halstead's measures, McCabe's measures, the Measure Defect-Density, the COCOMO model and the Function-Point Method. However, there are many other software measures, and it is necessary to be able to decide when to use which measure and how. Software measures are used to measure specific attributes of a software product of the software development process. We use them to derive a basis for estimates, to track project progress, to determine complexity, to help us to understand when we have achieved a desired state of software quality, to analyze the defects, and to experimental validate best practices. In short, they help us making better decisions.

Software measurement is a field of software engineering, it belongs to experimental software engineering, that means the understanding of strength and weaknesses of methods and tools in order to tailor them to specific goals of a particular software project. A cornerstone in this approach is measurement. The necessity of software measurement can be derived from the definition of software engineering by IEEE /IEEE90/: *The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software.* As indicated by the word *quantifiable*, at least one key professional society has acknowledged measurement as an integral part of a well-conceived software approach, and not merely an adjunct.

The terminology in the area of software measurement is not unique. We use the terms measures and metrics as synonyms. We do not use them in the sense of a space but as a mapping.

Software Costs and the Maintenance Problem

Today, we have still the fact that over 70% of the software development effort /BOEH79/, /PAGE88/, /SCHA93/, p.11, is spent in testing and maintenance of software. Schedule and cost estimates of software are grossly inaccurate, software has still a poor quality and the productivity rate for software is increasing more slowly than the demand for software. Schneidewind /SCHN83/ points out that there exists a maintenance problem because 70 - 80% percent of existing software was produced prior to significant use of structured programming, It is difficult to determine whether a change in code will affect something, It is difficult to relate specific programming actions to specific code. The major problem in doing maintenance is that we cannot do maintenance on a system which is not designed for maintenance. Roger Pressman writes about software maintenance /PRES92/: *By most estimates, the average software engineering organization spends 60 to 70 percent of its overall resources correcting, adapting, enhancing, and reengineering existing programs—an activity that we call "software maintenance." Yet this extremely important subject receives relatively little attention in the technical literature.*

In 1995 the *Bundesministerium für Forschung und Technologie /*BMFT95/ created a software initiative program. Among others,we can find there: *Software übernimmt wesentliche Aufgaben der Steuerung von Anlagen und Geräten und prägt damit zunehmend sowohl die Funktionalität als auch die Qualität der Erzeugnisse. In exportorientierten Branchen der deutschen Wirtschaft übersteigt der Software-Anteil an der Wertschöpfung der Produkte häufig die 50%-Marke. Bei Anlagen der digitalen Vermittlungstechnik entfallen bis zu 80% der Entwicklungskosten auf Software, so daß schon jetzt mehr als die Hälfte der Wertschöpfung von Siemens auf Software- Leistungen entfällt. Diese Entwicklung geht* weiter (Heinrich von Pierer SIEMENS AG, im FAZ Unternehmergespräch 10.8.1992. (Translation by the author: *Software takes over essential tasks by controlling installations and equipments and forms more and more the functionality and quality of products. In export oriented areas of the German economy the proportion of software production work exceeds the 50% label. In the digital communication techniques 80% of development costs are software costs. Today more than half the production work is based on software performance. This development proceeds.* (Heinrich von Pierer SIEMENS AG, in an interview with FAZ (Frankfurter Allgemeine Zeitung), August, 1992) /PIER92/.

However, there is not only a maintenance problem, as Card /CARD90/ and Page-Jones /PAGE88/ point out: *Although traditional rules of thumb estimate the proportion of development effort spent in design at 40 percent /CARD90/, recent estimates show an increase to about 60 percent for new methods. Thus, getting a good design is ever more essential to successful software development.* In /PAGE88/, p.26, a good summary about inflexible and non-maintainable systems is given. Page-

Jones writes there: *Sixty percent of the whole lifetime cost of the system - twice as much was spent at development - is spent on maintenance. Testing and debugging account for further 15 percent of the total cost; so 82 percent of the money spent on a system is consumed by putting it right and keeping it right. Only 15 percent is spent in constructing it in the first place.*

The citations above show that software maintenance effort is one major problem, but getting a good design is more also very essential to get a successful software development.

Needs for Determining Quality of Software with Engineering Methods

The facts presented in the previous section have generated the need of determining the quality of software with engineering methods. For this reason, computer scientists and engineers have begun to place increasing attention on quantitative methods as an information source of the quality of software. Already in 1976, Belady and Lehman stated in their classic research on the development of the IBM OS/360 operating system /BELA76/: *Law of increasing entropy: the entropy of a system (its unstructuredness) increase with time, unless specific work is executed to maintain or reduce it.* Entropy can result in severe complications when a project has to be modified and is generally an obstacle of maintenance. These statements above characterize the need of quantitative methods in order to understand the function of a large software system.

Since software engineering was first proposed as a discipline in its own right, see Naur /NAUR69/, the primary focus of attention has moved progressively earlier in the so-called project life-cycle. In the 1960’s and 1970’s the major debates focused on code, the structure of code, etc. Since the end of the eighties, interest focuses more and more on design aspects of software /PARN71/, /PARN72/, /PARN79/. There the criteria for modularization of software in the context of maintenance are discussed. Design methodologies such as structured design by Stevens et al. /STEV74/, Jackson's structured programming /JACK76/ and the object-oriented design of Booch /BOOC86/ attempt to address the problem of very late feedback by providing guidelines for the generation of systems structures and design evaluation criteria. The dichotomy between the importance of applying quantitative methods to the design of system architecture, and the absence of consistent results, or widely accepted measures, typifies the area of software measurement at present.

We feel that if software engineering is truly to become an engineering discipline, then quantitative models and the use of measurement are imperative. Yet, at present, there appears to be little progress in this direction.

Large Software Systems and Software Reuse

One of the major problems of large software systems is the understandability, complexity or comprehensibility of programs or entire software systems. Mostly, the three terms are used as synonyms. Shapiro /SHAP97/ points this out very clearly: *The complexity associated with software technology, however, is not that straightforward. Instead, it involves numerous facets and dimensions. Complexity's various contexts include algorithmic efficiency, the structure of procedures and data, and the psychological effort of problem comprehension. translation, and system design. Those contexts have manifested themselves in issues concerning structured programming, software metrics, program verification, formal methods generally, programming languages, the software life cycle, and programming environments. No solution aimed at a single area could provide the degree of relief many were seeking. Moreover, agreeing on singular approaches with respect to any of these issues also frequently proved difficult in the face of incommensurable philosophies and inescapable trade-offs. Recognition of the futility of technical singularity in any realm of software technology was slow in dawning.*

Many methods have been developed to increase the degree of understandability. A lot of time is spent reading and understanding programs in order to remove faults or to adapt the program to changed requirements. Many factors in the program code affect the comprehensibility of the program, such as the language used, the naming of variables, the structure, the indentation, explanatory documentation, the experience of the programmer, etc. Keywords, like coupling between modules, cohesion of modules, modularization, structured design, object-oriented design, etc. shall show some methods. However, the major goal is to map such keywords or qualitative attributes into quantitative attributes. Here, software measures can help to capture certain attributes of software quality.

Software reuse has been touted as a means of overcoming the software crisis. It promises to significantly improve end-product quality and process productivity in software development. Poulin /POUL97/ published an excellent book considering software measures for the reuse of software components. On p.2 he points out: *We normally consider "software" reuse as the use of existing components of source code to develop a new software program, or application. In this sense, a group of people, or organisation, the existing components /GART91/. However, reusable software components can take many other forms, including executable programs, code segments, documentation, requirements, design and architectures, test data, test plans, and experience. With this in mind, we define a reusable component as a group of functionally related software modules and their associated documentation. For example, a reusable component may take the form of a software "building block" of routines and documentation that offers primitive operations on top of which programmers can develop more complex and specific capabilities /LENZ87/.*

It is our understanding that measurement has to show whether reuse of software components really leads to lower maintenance costs. The proposed measures in the book of Poulin can help to achieve this goal.

The Importance of Software Measures

Some twenty or twenty-five years ago the area of software measures was a curiosity confined to a few university researchers and some industrial organizations. The first paper which exists in the research area of software measurement probably is from Rubey et al. /RUBE68/. No earlier reference can be found in this paper. Now, software measurement is a well-established discipline with a growing band of practitioners and adherents. Since the end of the seventies software measurement is becoming recognized as a useful way to soundly plan and control the properties and development of software and software projects. The research area of software measurement is also called software metrics or software engineering metrics. Here the term engineering indicates that engineering methods shall be used in the area of software measurement.

In 1991 Rombach /ROMB91/, p.217, wrote that software measurement is an essential component of mature software technology. It supports quality as well as project management. As far as quality management is concerned, measurement can help to investigate software related phenomena and thus contribute to building better software products, processes and quality models. As far as project management is concerned, measurement can help state software requirements unambiguously, assess their proper implementation throughout the software project, and achieve convincing product certification. The measurement goal of interest determines which measures are appropriate. The benefits of software measurement from a software engineering perspective can be summarized in the following way /GILE95/: G*ives you control of products and processes, demonstrate the productivity, effectiveness, and quality of the work, gain customer respect and credibility with management, identify where improvements are needed, let you know when to laugh and to when to cry.*

From a management perspective, measures support up-front estimation of time, resources, and quality, provide true status of projects to permit early insight into potential problems, enable the correction of problems before the consequences consume you, guide decisions on resources, propriety adjustments, and schedule stability based on data, not best guesses, and provide hard data on which processes can be improved and where to improve them.

In 1997, Shari Pfleeger mentioned in a guest editor's introduction /PFLE97/ about software measurement, among other: *Measurement is becoming an integral part of development and maintenance activities. Measurement is used not only for understanding, controlling, and improving development, but also for determining the best ways to help practitioners and researchers. Empirical software engineering will help us fine-tune our measurement activities so that we get the most information for the least measurement effort. As in most other sciences, we are moving along a measurement continuum; just as temperature measurement began as an index finger in the water (and a scale of not hot enough, hot enough, and too hot) and grew to sophisticated scales, tools, and techniques, so too is software measurement maturing and leading to a more sophisticated understanding of better ways to produce better products.*

Very often, people cite Lord Kelvin, in order to make clear that measurement is an essential concept of science. As Lord Kelvin (Original name: William Thomson)

/KELV91/ (1824-1907) said a century ago: *When you can measure what you are speaking about, and express it in numbers, you know something about it, but when you cannot measure it, when you cannot express it in numbers, your knowledge is of meager and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely in your thoughts advanced to the stage of science.* The statement above is certainly true, but it focuses too much on measurement and leaves out other formal methods, which are essential in software engineering, too. We have to mention here, that software measurement alone does not guarantee an excellent software product. Software measurement is essential in areas, where empirical views, like quality, cost estimation, understandability, etc., are important. From our view, theory building is the major task of science and measurement in all areas is an important tool for that.

Current State of Software Measurement

The field of software measurement is too large to describe the current state completely (See for that Chapter 3.). For this reason we give some highlights. While software measures have not yet achieved a degree of scientific maturity, it is still a valid concept and much work has been undertaken in the field. Although the current state of software measurement is still confused, software measures are more and more applied during every phase of the software life-cycle. Prediction of costs or effort is one of the major requirements. The purpose of prediction is to provide software managers with a forecast of the quality of the operational software early in the development process so that corrective actions can be taken, if necessary, when the cost is low. Other important areas of software measurement are cost estimation, software size estimation, effort for testing, and effort of maintenance after delivery of the product. It is also widely accepted that software measurement plays a dominant part with ISO9000-3 and ISO9126 standards that require measurement of processes and products. Ince et al. /INCE94/ summarize in their paper about software measures and ISO9000-3: *Thus, the software developer who is able to measure has either reached the standard of ISO9000-3 or is not far from it.*

In some areas remarkable progress in the area of software measurement has been done. One example is the Function Point Method of Albrecht /ALBR83/, which estimates the size of a software project. Thousands of companies all over the world use the Function-Point Method, and, mostly in UK, the modified MARK II Method of Symons /SYMO88/ is used. The Function-Point method is a good example for a pragmatically created software measure. The validity of this measure has been tried to show by many experiments. However, there are many unclear properties of the Function-Point Method. The complexity factors seem to be arbitrary and there is no empirical evidence, if a factor is 2 or 4. The COCOMO model of Boehm /BOEH81/ is a prediction model based on the Measure LOC. It is not really a specification measure, but it can be used in this phase of the software life-cycle. Initially, the basic COCOMO model was applied to the 700K LOC software system in question. Of note is that the basic COCOMO applies to small and medium-sized software products that

are developed in a familiar in-house environment. Success can be reported in the area of code analysis and measurement in the software design phase

Today, the hypothesis is undisputed that a good software design causes lower maintenance costs. Page-Jones /PAGE88/ supports this view by presenting dozens of examples of a good software design using the Constantine Method without any measurement. The question is here how well this hypothesis can be validated with software measures.

There is another interesting aspect in the area of software measurement. Object-oriented techniques were introduced as a new discipline in the area of software engineering, that shall avoid, among others,disadvantages of imperative languages. We will not discuss here the advantages and disadvantages of object-oriented techniques, but till today we found more than two hundred software measures which were developed or defined for object-oriented programming techniques. Remarkable is, that these software measures are mostly focused on understandability of object-oriented systems.

Problematic of Software Measurement

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

The question is why is software measurement so problematic? One answer may be, following Roche et al. /ROCH94/, that software engineering is a highly complex process producing highly complex products. Moreover, each project and its products tend to be something of *one off* in nature, a point highlighted by Schneidewind as a difficulty in validating a methodology /SCHN91/. Other problems are that people do not like to be controlled by software measures. And, last not least, there is a lack of an intensive education of people in software measurement regarding both: a theoretical framework for software measurement and a soundly planning of experiments.

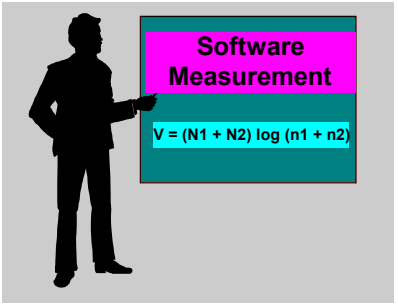
The major problem of measurement in software engineering, but also in the area of artificial intelligence, is a skepticism of using numerical values because there is no satisfaction in the interpretation the numbers and a semantic of the values is missing. This lack may be true in some cases, but not generally. The assignment of simple numbers to hypotheses without knowing the empirical evidence of these numbers is a major mistake. The empirical evidence of numbers can be characterized, among others,by several empirical conditions and scale types. Numbers are elements of a scale, that means, they are subject of a homomorphic mapping of an empirical to a numerical relational system and vice versa. Mostly, these facts are neglected.

However, we think, today it is widely accepted that software measurement is a valuable technique for understanding, guiding, controlling and improving software development. It is an interesting phenomenon that the Measure LOC and the Measures of McCabe /MCCA76/ today still are the most used and discussed software measures. The Measure of McCabe was defined for single module complexity but also for the entire system complexity. The question is still discussed whether the Measure of McCabe is a *good* or a *bad* measure. Another unsolved question is whether the Measure of McCabe can be used as a predictor for software maintenance attributes. We think the reasons for these discussion are the following: firstly, there is a lack of education in the area of software measurement, secondly, many people believe that software measurement is an easy thing, and thirdly, although there exists a proper theory for software measurement - called measurement theory (see for that Zuse /ZUSE89/, /ZUSE91/, /ZUSE92/, /ZUSE94/, Bollmann-Sdorra and Zuse /BOLL93/, Baker et al. /BAKE89/, Fenton et al. /FENT91/, /FENT96/) - only a few people consider and apply this theory.

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

Poor Education

As mentioned above, the education of students and scientists in software measurement is a major problem. Our impression is that people sometimes think, that measurement is a simple procedure. With some exceptions a theory of software measurement is not presented and used by scientists. Many scientists point out, that the current state of software measures is not satisfying. In Mills /MILL88/ we can find: *Unfortunately, most of the metrics defined have lacked one or both characteristics: a sound conceptual, theoretical basis, statistically significant experimental validation*. Jones /JONE94a/ states very clearly: *All graduate software engineers and computer scientists should understand the basic tools of measurement and metrics.* Jones furthermore points out, that it is a pity to hear from a graduate student: *I'm a graduate student in computer science at Stanford, and I've never heard of Function-Points. Can you tell me what they are?*



In 1995 Rubin mentioned five myths of software measurement: the first myth is that measurement is overhead: measurement is actually a value-added, real-time part of information technology practice and process. The second is that there is a *best* measure and that selecting it will solve all measurement problems, but there is no single measure that can assess all aspects of an information technology organization. The third myth is that the most critical success factor for a measurement program is

selecting the right measure, but more important are the fit of the measures with business goals, their fit as part of the workflow process, their intrinsic quality, and the organization's willingness to use them. The fourth myth is that measures last forever, but in fact they need to be designed to accommodate enhancements. The fifth myth is that information technology should focus on its internals, but in fact it needs to be measured as part of a business.

These statements show that the education of people in the field of software measurement is poor. In order to use software measurement in the sense of an engineering discipline, like electrical engineering, a theoretical framework for software measurement is necessary and the education of people has to be improved. Experimental results can only be understood, if we have hypotheses of reality. For this reason, this book focuses on a framework for software measurement.

Our Framework for Software Measurement

Not measurement itself, but building (empirical) theories is one of the major goals in science. Undoubted, measurement is one of the most important foundations for the development of empirical theories. Today, it is a fact, that empirical theories in the area of software measurement are neglected. Endres /ENDR89/ formulates the empirical and theoretical role of software engineering very clearly: *Like any other engineering discipline, software engineering has both its theoretical and empirical foundations.* In order to create hypotheses of reality and to build empirical theories, we decided to present a framework for software measurement. Our framework for software measurement is based on measurement theory, that is based on the texts of Roberts /ROBE79/, Krantz et al. /KRAN71/ and Luce et al. /LUCE90/. The author extended the basic assumptions of classic measurement theory especially for the needs in the software engineering area. Measurement theory gives clear definitions of terminology, a sound basis of software measures, criteria for experimentation, conditions for validation of software measures, foundations of prediction models, empirical properties of software measures, and criteria for measurement scales. From our view it is important to mention that measurement scales are not the major focus of measurement theory. The major advantages of measurement theory are hypotheses about reality, an empirical interpretation of measurement values, and the translation of numerical properties back to empirical properties, which can lead to theory building. The interpretation of measurement values is essential because people measure in order to get objective information. You cannot get objective information and make decisions based on measurement values without a theory or a framework for software measurement.

We think, it is essential to have a unique terminology of software measurement, a mechanism for interpreting numbers, criteria for selecting software measures, conditions for comparing cost estimation models, theoretical foundations of the validation of software measures and the models for prediction. In short: a framework for software measurement shall help teachers, practitioners and students to understand the numbers, the results of experimentation and the proper use of statistics.

The past in Science showed that a real Progress cannot be made without Theory Building

We use an axiomatic approach in order to characterize software measures or problems in the software measurement area. Axioms are basic assumptions of reality. The use of an axiomatic approach does not imply that the axioms always hold in reality. The advantage of an axiomatic approach is that there are well defined conditions under which some mathematical models hold. These conditions can be criticized or falsified by discussions or experiments. This might be difficult if only a mathematical formula is given. We do not believe that a mathematical model for something should hold in every situation. Even in classical physics many mathematical models only hold in special situations. Our proposed axioms can be discussed by experts. Hence, the axioms might be true. On the other hand, for every expert opinion there is mostly another expert with the opposite opinion. But even then a mathematical model is not useless because to clarifies the situation.

Most scientists today agree upon the fact that observation always implies certain assumptions, concepts, etc. - in short: that is conducted by theory.

The axiom systems established by measurement theory and applied to software measurement are like a language. Scientists, practitioners and students can talk about properties of measures in every phase of the life-cycle. Without such a language it is more a feeling than science to talk about measures. This language in form of axioms covers all types of measures, it allows to give criteria for the validation of measures, presents theorems for prediction models, and even the empirical assumptions of the Function-Point Method can be clarified. The results of our methodological investigations are qualitative models behind measures.

In short: We hope that our work will encourage a more fundamental look at software measurement at all and ultimately lead to a scientific progress in the software measurement area.

Mathematics and Software Measurement

For some authors, there exists the problem to apply mathematics in an empirical science. In literature we can find the statement: *Science is not mathematics*. But, it is a fact that mathematics was introduced in software engineering long before measurement theory. Until now approximately 1500 software measures were defined. Measures are mathematical functions. Each of these measures define by itself a mathematical model of complexity, maintainability, cost estimation, etc. So, why not

use mathematics and especially measurement theory in order to study these measures and models? Doing this, we can see that there are several contradicting positions in this area.

Glass /GLAS96/ pointed out in an article of the relationship between theory and practice in software engineering: *Theory and practice are traveling very different roads on the topic of software metrics. Theoreticians have proposed a number of metrics, but most of them are unverified and there are considerable differences among theorists as to their value. Practitioners, on the other hand, seldom use metrics, but when they do the set used is often unrelated to the set proposed by theorists. At the present time, it is difficult to say whether theory or practice is leading in the topic of metrics, but it is possible to say that the field is still in turmoil. In the next decade, we should find out which is ahead, theory or practice.*

We do not agree to Glass saying that persons who propose measures are theoreticians. From our view proposing a measure is a method, but it is not a theory of measures or measurement. Our view of theoreticians in the software measurement area is based, among others,on a theory of empirical laws combined with numerical properties. Such empirical laws can be denoted as models behind measures. As we will show in this book, it is our view that behind every measure a qualitative model of complexity, cost estimation, maintainability, etc. is hidden. A theory of software measurement includes at least the question: What does a measure measure? The statement of Glass: *Practitioners, on the other hand, seldom use metrics, but when they do the set used is often unrelated to the set proposed by theorists,* shows very clearly that we need a theory for software measurement. Our axiom systems support to clarify the different views of theoreticians and practitioners.

We use as mathematics algebraic systems, homomorphic mappings of one system into another system, empirical and numerical relations, sets, and measurement theory, like the extensive structure. The mathematics are not well known among software engineers. For this reason we present the foundation of mathematics, too. A major goal of this book is to introduce the reader in this new class of empirical and mathematical results of measurement theory and an application to software measurement. Proposing software measures is not theory. From our view, a theory of software measures considers, among others,the models behind measures. From such models, like a model of complexity, empirical theories can be derived.

Ordinal Scales or Higher Scale Levels for Software Measurement

There is a ongoing discussion, whether software measurement should take place only on the ordinal scale level. Cherniavsky /CHER91/ points out: *At best, we view software complexity measures to be ordinal measures.* This view is questionable because in the software measurement area measures are heavily used, which assume more sophisticated structures than only pure ranking properties, like the extensive structure. Examples are the COCOMO model and all the many measures based on the extensive structure. For example, the Measure LOC is a ratio scale measure if it is

an ordinal scale measure. Measurement theoretic axioms show that. We address this topic in almost each Chapter of this book, but especially we discuss it in Chapter 7.

Software Measures and Statistics

If you are interested in statistics and software measurement, then you only will find limited help in this book. We refer to the book of Kitchenham /KITC96/. However, it is our understanding that applying statistics to software measurement needs a theory behind software measures. A theory behind software measures - as presented in this book - and statistical investigations have to go hand in hand. It is our view that statistics and theories behind software measurement support each other.

The (Software) Measurement Literature

The amount of literature in the area of software measurement and related topics increases rapidly (See for that the end of Chapter 3 and the literature references on the CD). However, only a few books contain sections of measurement theory and software measures. Examples are Fenton /FENT91/, /FENT94/, /FENT95/, /FENT96/, Shepperd /SHEP93a/, /SHEP95/, Henderson-Sellers /HEND96/, the IEEE Standard 1061 of Schneidewind /IEEE93/ and Zuse /ZUSE91/. Fenton requires a theory behind software measurement, but he only considers the ordinal scale. Shepperd considers some aspects of measurement theory, but he does not use axiom systems above the ordinal scale level. Zuse introduced axiom systems for several scale types, the extensive structure and the modified function of qualitative belief.

Our approach of a framework for software measurement is based on the work of Roberts /ROBE79/, Krantz et al. /KRAN71/, Luce et al. /LUCE90/, but the formulated axiom systems, the presented empirical and numerical conditions, the discussed binary or concatenation operations, cannot be transferred to software measurement without essential modifications by the author for needs in software measurement. From a practitioners view, our goal is to present concepts, which are intuitive for the user, and can be described or characterized by empirical conditions. The red thread of this book is to combine needs of practitioners with a proper theory.

Organization of the Book

The book contains twelve chapters. Almost each chapter of this book contains exercises for undergraduate and graduate students. The solutions can be found in Chapter 11. The structure of the book is the following.

- Chapter 1 contains the introduction.
- Chapter 2 presents prerequisites for software measurement. Models of programs into flowgraphs, of software systems into the structure chart method, and models of object-oriented programming, are presented. Examples of mismeasurement in the area of software measurement are discussed. These examples shall help the

reader to understand, that software measurement without a sound theory is not possible.

- In Chapter 3 the history and the state-of-the-art of software measurement is presented.
- In the Chapters 4 and 5 a framework for software measurement is introduced, which allows a translation of numerical properties back to empirical properties. The framework for software measurement includes the analysis of properties behind software measures, the validation of software measures, which is important for the acceptance of software measures in practice, the evidence of empirical conditions behind software measures, the prerequisites for appropriate statistics, the meaning of measurement scales, the importance of hypotheses of reality, the role of concatenation operations in order to get more sophisticated measurement structures, the importance of combination rules, the reduction of wholeness to additive measures, and a discussion of the role of units.
- In Chapter 6 foundations for object-oriented measures are given. Weaker axiom systems than that of the extensive structure are introduced. They are based on the function of belief, qualitative belief, qualitative relation of belief and the DeFinetti axioms.
- In Chapter 7 desired properties of software measures, as proposed by many authors, like Kearney et al., Weyuker, etc., are discussed. These mostly verbally formulated properties are characterized with measurement theoretic conditions.
- Chapter 8 presents definitions and conditions for the internal and external validation of software measures. Among others,it is shown, what prediction of costs of maintenance in the context of the Measures LOC, the Measure of McCabe, and the Informationflow Measure of Henry and Kafura, means. The limits of the validation of software measures and prediction models are discussed and five theorems for prediction models are presented.
- Chapter 9 considers the application of software measurement in practice. Among others,the Function-Point Method and the COCOMO model are discussed in detail, the application of software measurement during the software life-cycle and software measurement in the context of the ISO9000-3 norm is considered. The ISO 9000-3 norm implies measurement of processes and products. More than seventy measures for the software life-cycle and for object-oriented applications are discussed.
- Chapter 10 contains the afterword.
- Chapter 11 demonstrates the solution of the exercises.
- Chapter 12 contains a glossary of more than two hundred terms used in this book.

After the chapters above some attachments, references to literature, the name and subject index are following.

- Attachment I presents a brief description of the System ZD-MIS

- Attachment II presents the proof of equivalence of the two extensive structures as discussed in the Chapters 4 and 5.
- Attachment III presents the proofs of the theorems for validation of measures and prediction models of Chapter 8.
- Attachment IV presents the proofs of the Theorems C1-C4 (Independence Conditions C1-C4) in Chapter 5.
- Attachment V presents the used notations, and
- Attachment VI gives an overview of the used axiom systems.
- A list of the used definitions, theorems, exercises can be found the end of the book.
- References to Literature.

This book contains a CD that makes users able to search in a list of more than 1500 references of literature. The literature is classified by pre-defined queries, like object-oriented measures, measures for cohesion, coupling, experimentation, etc. The CD also contains a small Demo-Version of the System ZD-MIS (Zuse / Drabe Measure Information System) /ZUSE95d/. It is a system that gives scientists and practitioners the necessary knowledge for education of software measurement for academics and practitioners, and it supports people of commercial institutions in selecting software measures for their needs (See the README.TXT file).