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 divided the scale of 100 degrees and 0 degrees in hundred equal sectors. But the question was for a long time whether 50°C 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 /CHR181/, General Electric Company /FIT78/, 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/, Lakhota /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 an 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:

<table>
<thead>
<tr>
<th>Name of Model</th>
<th>Organization</th>
<th>Year</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delphi</td>
<td>Rand Corp.</td>
<td>1966</td>
<td>/HELM66/</td>
</tr>
<tr>
<td>Nelson's SDC</td>
<td>SDC</td>
<td>1966</td>
<td>/NELS66/</td>
</tr>
<tr>
<td>Wolverton</td>
<td>TRW</td>
<td>1974</td>
<td>/WOLV74/</td>
</tr>
<tr>
<td>RCA Price-S System</td>
<td>RCA</td>
<td>1976</td>
<td>/FREI79/</td>
</tr>
</tbody>
</table>
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:

\[
\text{EFFORT}(P) = a \cdot \text{LOC}^b,
\]

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 Polytechnics 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 (Metrical 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 a 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).
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).
10. NASA Software Engineering Workshop, about 800 people
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 for 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:
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 $7 \pm 2$ 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


/ARNO92b/ Arnold, Robert, S.:


/BELA79/ Belady, L.A.

/BERL80/  Berlinger, Eli:  

/BIEM84/  Bieman, J.M.:  

/BIEM91/  Bieman, J.M.:  

/BIEM92/  Bieman, J.M.; Schultz, J.:  

/BOEH81/  Boehm, B.W.:  

/BOEH84/  Boehm, B.W.:  

/BOLL81/  Bollmann, P.; Cherniavsky, V.S.:  

/BOLL84/  Bollmann, Peter:  

/BOLL85/  Bollmann, Peter; Zuse, Horst:  

/BOLL93/  Bollmann-Sdorra, P.; Zuse, H.:  

/BOLO88/  Boloix, Germinal; Robillard, Pierre:  

/BOOC91/  Booch, G.:  


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


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

/DAME92/ Damerla, S.; Shatz, Sol. M.: 

/DEMA82/ DeMarco, Tom: 

/DEMA87/ DeMarco, Tom; Buxton, John: 

/DEMI81/ DeMillo, Richard A.; Lipton, Richard J.: 

/DHAM94/ Dhama, Harpal: 

/DUMK92/ Dumke, Reiner: 

/DUNS77/ Dunsmore, H.E; Gannon J.D.: 

/EJIO91/ Ejiogu, L.: 

/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.: 

/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: 

/FENT91/ Fenton, N.: 
Fenton, Norman:

Fetcke, Thomas:

Freidman, F.R.; Park, R.E.:

Gilb, T.:

Goodman, Paul:

Grady, Robert B.; Caswell, Deborah L.:

Grady, Robert B.:

Grady, Robert B.:

Grady, Robert B.:

Hall, Nancy R.:

Hall, N.; Preiser, S.:

Halstead, M.H.; Gordon, R.D.; Elshoff, J.L.:

Halstead, M.H.:

Hamer, P; Frewin, G.:

Harrison, Warren; Magel, Kenneth:

/HARR87/  Harrison, Warren; Cook, Curtis:

/HECH77/  Hecht, M.S:

/HELL72/  Hellerman, L.:

/HELM66/  Helmer-Heidelberg, O:

/HENR81/  Henry, S. M.; Kafura, D. G.:

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.:

/HETZ93/  Hetzel, Bill:

/HUTC85/  Hutchens, D.; Basili, V.:

/IEEE89/  IEEE:

/IEEE89a/  IEEE:

/IEEE93/  IEEE Computer Society:

/INCE89/  Ince, Darrel:


/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.


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

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


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


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.


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


McCabe, T; Butler, Charles W.:


Patel, Sukesh; Chu, William, Baxter, Rich:

Perlis, Alan; Sayward, Frederick, Shaw, Mary:

Pfleeger, S.L.; Fitzgerald, J.C.:

Piwowarski, Paul:

Porter, A.A.; Selby, R.W.:

Pressmann, Roger S.:

Putnam, L.H.:

RADC (Rome Air Development Center):

Rains, E.:

Rising, Linda; Callis, Frank, W.:

Roberts, Fred S.:

Robillard, Pierre, N.; Boloix, Germinal:

Rocacher, Daniel:

Rombach, D.:


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


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


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

/SHEP91/ Shepperd, Martin; Ince, Darrel:

/SHEP93/ Shepperd, Martin (Editor):

/SHEP93a/ Shepperd, Martin; Ince, Darrel:

/SHOO83/ Shooman, Martin L.:

/SOMM92/ Sommerville, Ian:

/STEV74/ Stevens, W.P.; Myers, G.J.; Constantine, L.L.:

/SYMO88/ Symons, Charles, R.:

/SYMO91/ Symons, Charles, R.:

/TAUS81/ Tausworthe, R.C.:

/TAYL93/ Taylor, D.A.:

/TEGA92/ Tegarden, David, P.; Sheetz, Steven, D.; Monarchi, David, E.:

/TEGA92a/ Tegarden, David, P.; Sheetz, Steven, D.; Monarchi, David, E.:

/THUS88/ Thuss, Jeffrey:

/TROY81/ Troy, Douglas; Zweben, Stuart:


/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.


