
Quality Management of Software and Systems

Software Measurement

Contents

- ☐ Motivation
- ☐ Software Quality Experiments
- ☐ Software Measures
- ☐ Measuring Scales
- ☐ Cyclomatic Complexity
- ☐ Current Impact of Software Measurements
- ☐ Software Quality Measurement

Motivation

Measurement

- ☐ *„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 a meager and unsatisfactory kind.“*
(Lord Kelvin, Popular Lectures and Addresses, 1889)
- ☐ *„Was man messen kann, das existiert auch!“*
(Max Planck, 1858 - 1947)

Motivation

Measurements in Software Development

- Substitutes quantitative and reproducible statements for qualitative and usually intuitive statements about software
- Example
 - Qualitative, intuitive
 - The developer states: 'I have fully tested my software module.'
 - Quantitative, reproducible
 - 'My test tools states a branch coverage of **57% (70 of 123 branches)** at the moment. In our company modules are considered sufficiently tested with a branch coverage of **95%**. Thus, I have to test at least **47** additional branches with an estimated additional effort of **1.5 days** based on experiences with similar modules.'

Motivation

Measuring Quality in Software Development

- Today, software is used in application domains, where quantitative statements are common or necessary
 - Contracts: ‘We stipulate a minimum availability of 99.8%!’
 - Safety proof of a rail system for the Eisenbahnbundesamt: ‘What is the residual risk of software failures?’
 - Is the estimated number of residual faults sufficiently small to release the products?
 - Is the possibility of software faults in controllers causing a failure in our upper class limousine sufficiently small?
 - We need a failure free mission time of four weeks. Is this possible?

Motivation

Measuring Quality in Software Development: Problems

- Most quality characteristics not directly measurable!
 - Number of faults
 - Availability
 - Reliability
 - Safety
 - ...
- Quality characteristics may be
 - Determined experimental (e.g., reliability)
 - Calculated from directly measurable characteristics (e.g., number of faults)

Software Quality Experiments

Stochastic Analysis of Software Reliability: Situation

- ☐ Independent research area since approximately 30 years
- ☐ Sparse influence to software development in practice
- ☐ Mathematical foundation partly too complex
- ☐ A lot of different stochastic reliability models
- ☐ A priori selection of a model not possible
- ☐ Determination of model parameters necessary
- ☐ Theory application to practice needs powerful tool support

Software Quality Experiments

Stochastic Analysis of Software Reliability: Theory

$$m(t) = E(N(t))$$

- ☐ Musa and Goel-Okumoto model, respectively
- ☐ Generalized Goel-Okumoto model
- ☐ Musa-Okumoto model
- ☐ Generalized Musa-Okumoto model
- ☐ Duane and Crow model, respectively
- ☐ Log model
- ☐ Log power model
- ☐ Generalized log power model
- ☐ Yamada S-shape model
- ☐ Generalized Yamada S-shape model
- ☐ Geometric Moranda and deterministic proportional model, resp.
- ☐ Littlewood model
- ☐ Inverse linear model

$$m(t) = a(1 - e^{-bt})$$

$$m(t) = a(1 - e^{-bt^c})$$

$$m(t) = a \ln(bt + 1)$$

$$m(t) = a \ln(bt^c + 1)$$

$$m(t) = at^b$$

$$m(t) = a \ln(bt)$$

$$m(t) = a \ln^p(t + 1)$$

$$m(t) = a \ln^p(ct + 1)$$

$$m(t) = a(1 - (1 + bt)e^{-bt})$$

$$m(t) = a(1 - (1 + ct)e^{-bt})$$

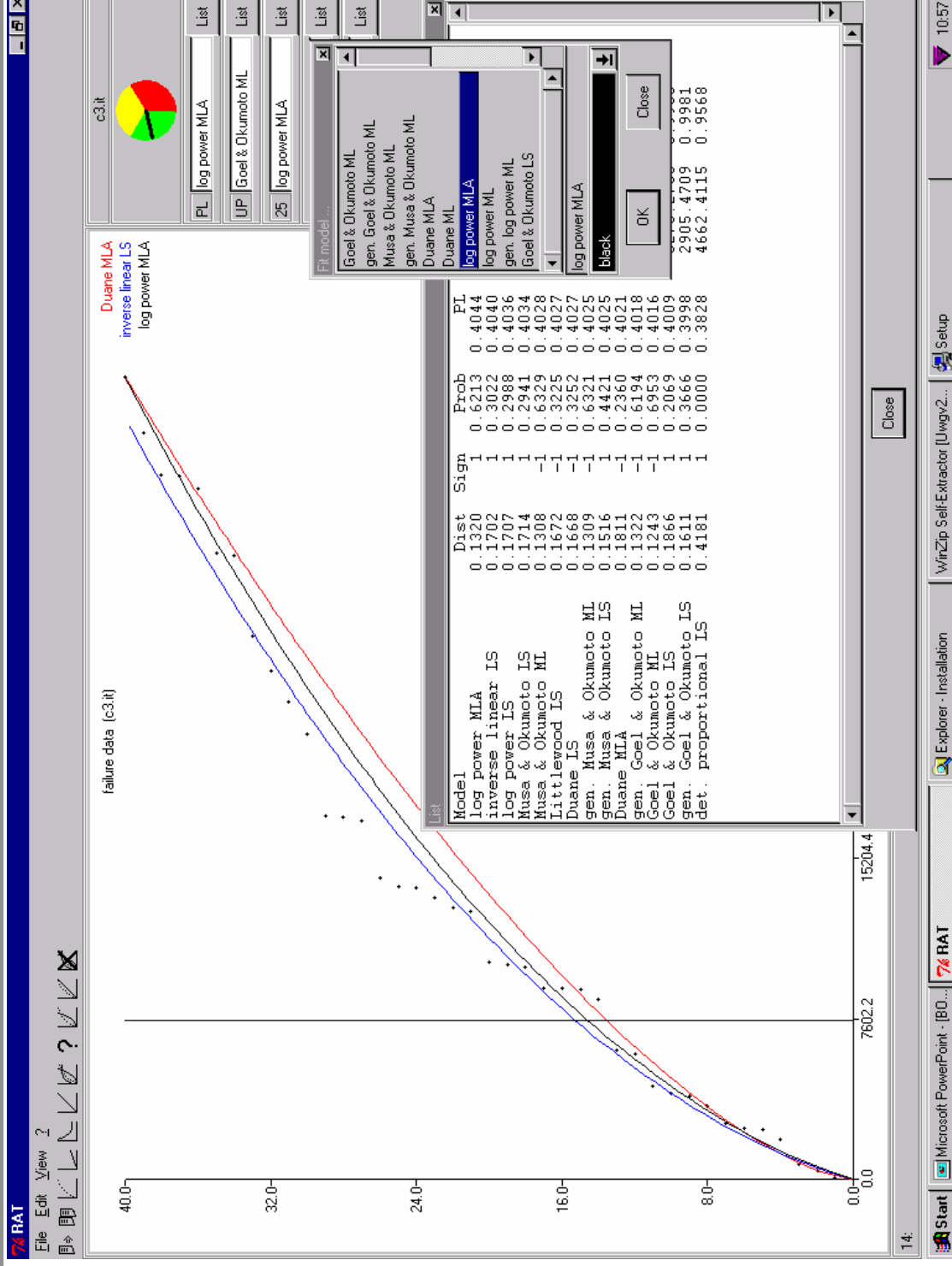
$$m(t) = a + b \ln(t + 1)$$

$$m(t) = c(t + a)^{-b}$$

$$m(t) = a(\sqrt{bt + 1} - 1)$$

Software Quality Experiments

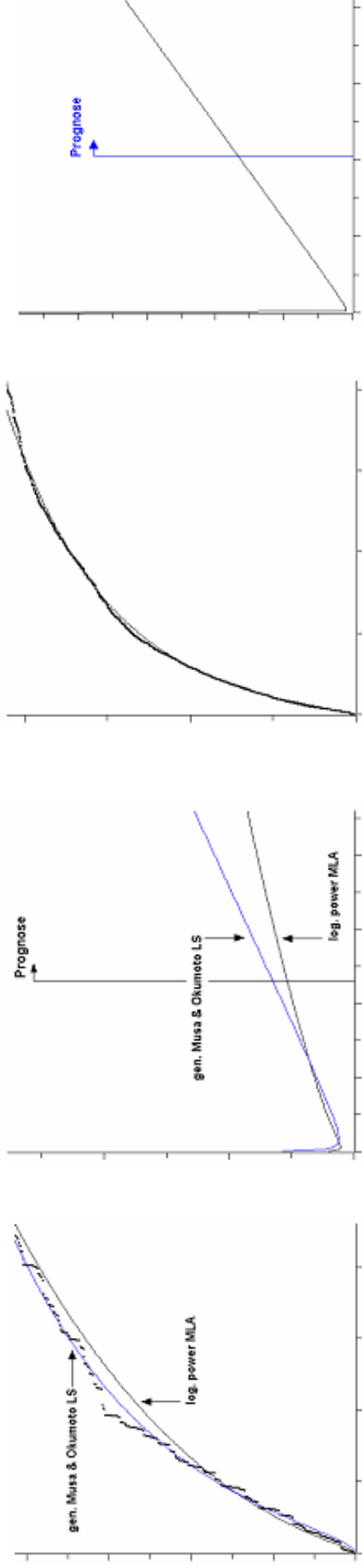
Stochastic Analysis of Software Reliability: Practical Method of Resolution



Software Quality Experiments

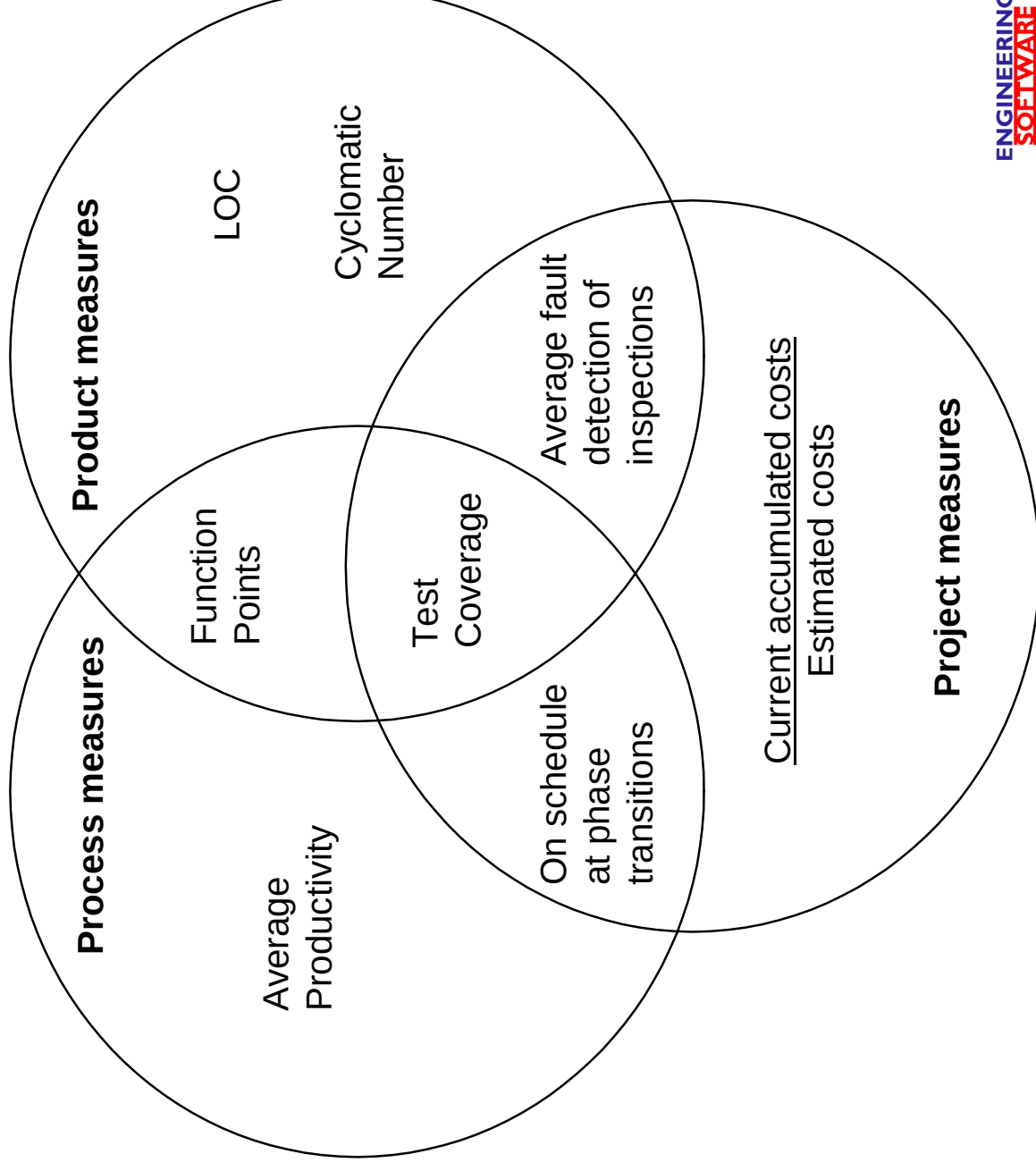
Stochastic Analysis of Software Reliability: Practical Method of Resolution

- Numerous application domains (traffic engineering, medical engineering, telecommunication, ...)



Software Measures

Applying Measures



Software Measures

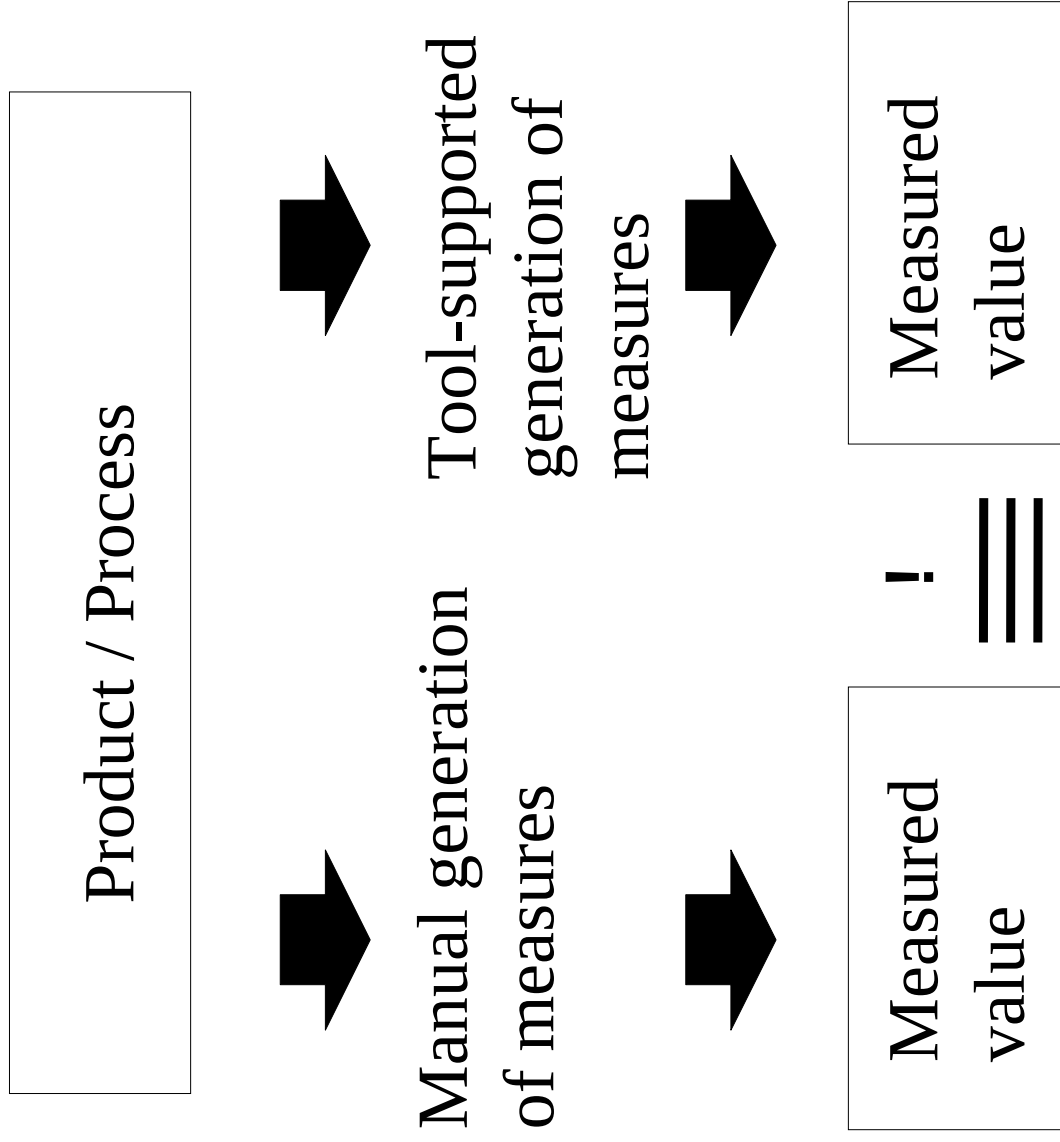
Requirements of Measures

- ☐ Simplicity
 - Is the result so simple that it could be easily interpreted?
- ☐ Adequacy
 - Covers the measure the desired characteristic?
- ☐ Robustness
 - Is the value of the measure stable against manipulations of minor importance?
- ☐ Timeliness
 - Can the measure be collected in sufficient time to allow a reaction to the process?
- ☐ Analyzability
 - Is the measure statistically analyzable (e.g., numeric domain) (For this requirement the type of the measure scale is crucial)

Software Measures

Requirements of Measures: Reproducibility

- Normally, a measure is reproducible, independent of the generation mechanism, if it is defined in a precise way



Software Measures

Requirements of Measures: Reproducibility

□ Examples

- McCabe's cyclomatic number: $e - n + 2$

e = Number of edges in a CFG; n = Number of nodes in a CFG; CFG = Control flow graph

- Completely reproducible

- Lines of Code (LOC)

Count empty lines? Count lines with comment?

- Completely reproducible, if adequately defined
- Function Points: manual evaluation of complexities needed
 - Not completely reproducible in principle
- Understandability
 - Poor reproducibility

Software Measures

Evaluation of Measures

- ☐ A recommendation of lower and upper bounds for measures is difficult
- ☐ Which values are 'normal' must be determined by experience
- ☐ A deviation from usual values may indicate a problem, not necessarily, though

Software Measures

Calibration of Measures and Models

☐ The correlation between measures and relevant characteristics demands a calibration, which has to be adapted to changing situations if necessary

☐ Empirical and theoretical models can be distinguished

☐ Example

- Theoretical effort model (cp. Halstead-Measures)
 $E = \dots \text{size}^2 \dots$

The square correlation between effort and size was identified by theoretical considerations

- Empirical effort model: $E = \dots \text{size}^{1.347} \dots$

The exponent of 1.347 was determined by statistical data analysis

Measuring Scales

- While expressing abstract characteristics as numerical value, it is necessary to figure out which operations can be reasonably performed on the values
- Example
 - Measuring length
 - Board *a* has a length of one meter. Board *b* has a length of two meters. Thus, board *b* is two times as long as board *a*.
 - This statement makes sense
 - Measuring temperature
 - Today, we have 20°C. Yesterday it was 10°C. Hence, today it is twice as hot as yesterday
 - That is wrong. The correct answer would be: Today is approximately 3.5 % warmer than yesterday
 - Obviously, there is a difference between the temperature scale in °C and the length in meters, which leads to operations not applicable to the temperature scale

Measuring Scales

- ☐ Nominal scale
 - Free labeling of specific characteristics
 - Inventory numbers of library books (DV 302, PH 002, CH 056, ...)
 - Names of different requirements engineering methods (SA; SADT, OOA; IM, ...)
- ☐ Ordinal scale
 - Mapping of an ordered attribute's aspect to an ordered set of measurement values, such that the order is preserved
 - Mapping of patient arrivals to the waiting list in a medical practice
- ☐ Interval scale
 - A scale, which is still valid if transformations like $g(x) = ax + b$, with $a > 0$ are applied
 - Temperature scales in degree Celsius or Fahrenheit. If F is a temperature in the Fahrenheit scale, the temperature in the Celsius scale can be determined as follows: $C = \frac{5}{9}(F - 32)$. The relations between temperatures are preserved

Measuring Scales

- ☐ Rational scale
 - Scale, where numerical values can be related to each other (percental statements make sense)
 - Length in meters (It is twice as far from a to b than from c to d)
 - Temperature in Kelvin
- ☐ Absolute scale
 - Scale, providing the only possibility to measure circumstances
 - Counting

Cyclomatic Complexity

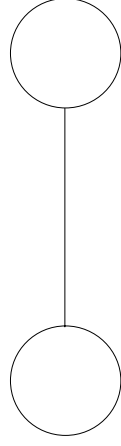
- ☐ Common measure of complexity
- ☐ Often surrounded with an aura of an 'important' key measure
- ☐ Originated from graph theory (strongly connected graphs) and thus relating to control flow graphs and programs
- ☐ Calculation: $e - n + 2$
(e = Number of edges, n = Number of nodes)
- ☐ Easy to calculate as it depends strongly on the number of decisions within the program
- ☐ Suited as complexity measure, if the number of decisions predicate the complexity of the program
- ☐ Probably the most common measure in analysis and testing tools

Cyclomatic Complexity

- Cyclomatic number is a measure of the structural complexity of programs
- Calculation based on the control flow graph
- Cyclomatic number $v(G)$ of a graph is: $v(G) = e - n + 2$
(e – Number of edges, n – Number of nodes)

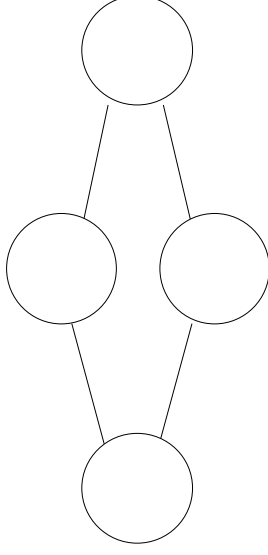
Cyclomatic Complexity

□ Cyclomatic complexity of graphs



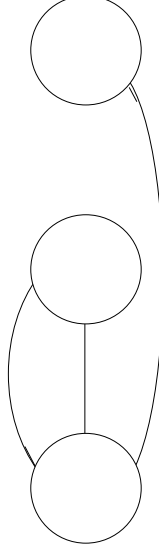
Sequence

$$v(G) = 1 - 2 + 2 = 1$$



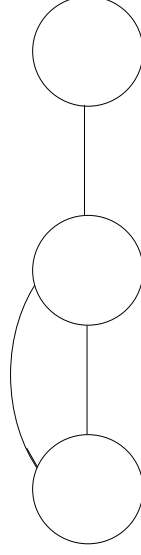
Selection

$$v(G) = 4 - 4 + 2 = 2$$



Pre-test loop

$$v(G) = 3 - 3 + 2 = 2$$



Post-test loop

$$v(G) = 3 - 3 + 2 = 2$$

Current Impact of Software Measurements

- Efficient software measurements are important for the following areas
 - Flat management structures
 - Standardizations with respect to software developments
 - Achieving a high Capability Maturity Level (Assessments)

Current Impact of Software Measurements

Software Measurements and Flatter Management Structures

- Trend for software management towards flat structures
 - One manager supervises significant more developer than before
 - Provision and summarization of information not through middle management, but automated measurement systems
 - Management intervention only necessary if measurement values indicates problematic situations
- ➔ Efficient measurement is an important requirement

Current Impact of Software Measurements

Software Measurements and Software Development Standards

- Standards become more and more important for the software development (e.g., ISO 9001)
 - Quality proof for potential customers
 - Marketing argument; differentiation from not certified competitors
 - Important with respect to product liability
 - In some domains requirement for the contract
 - All standards attach importance to systematic procedures, transparency, and control of the development process
- ➔ This can be proved by adequate measures

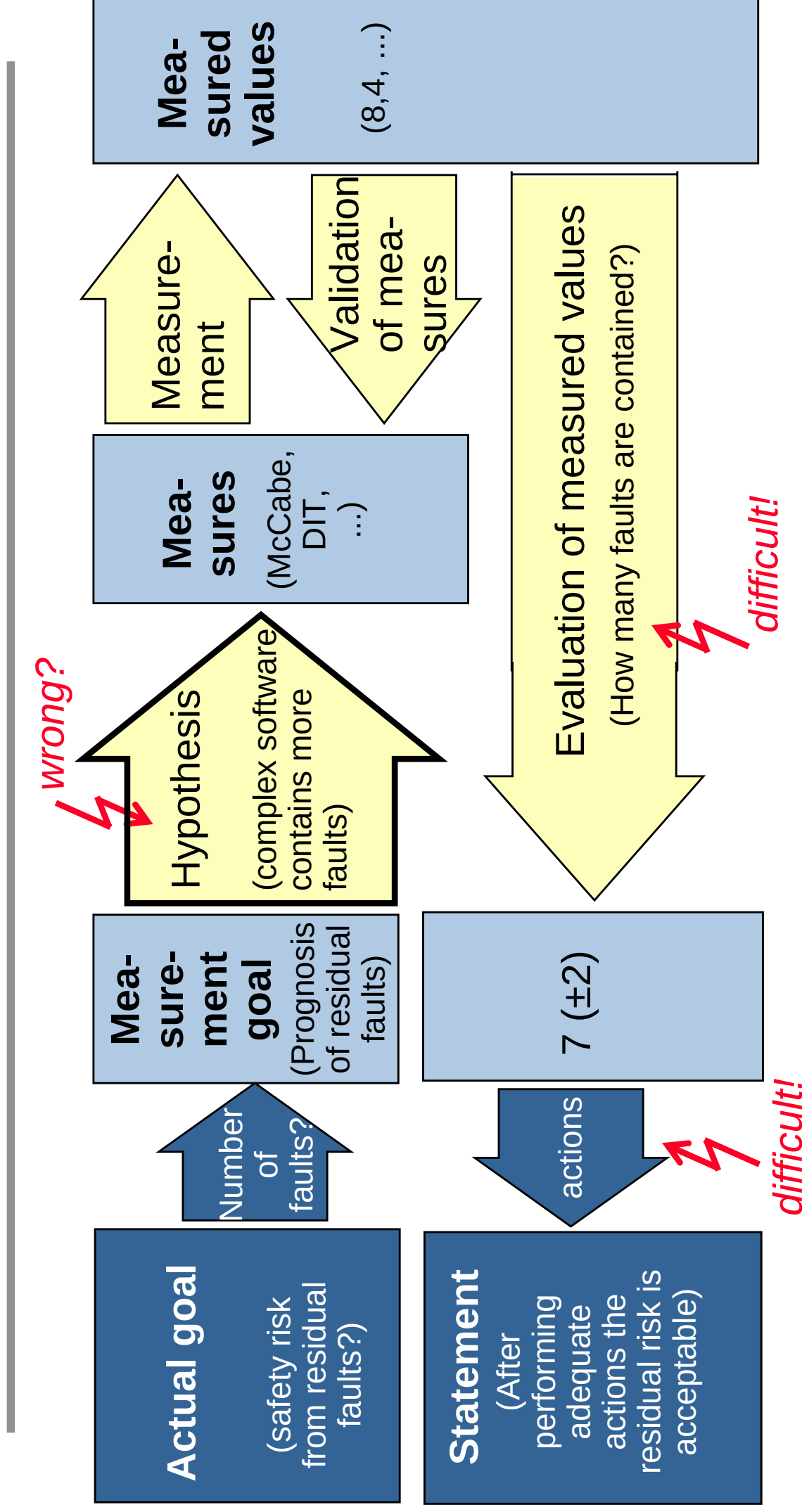
Current Impact of Software Measurements

Software Measurements and the Capability Maturity Model

- Capability Maturity Model assigns the maturity of a software development process to one of five levels. The possible levels are: 1-initial, 2-repeatable, 3-defined, 4-managed, 5-optimized
- Reaching level 4 or 5 is only possible if a measurement system exists and is used that provides the following tasks
 - Measurement of productivity and quality
 - Evaluation of project based on this measurements
 - Detection of deviations
 - Arrange corrective activities if deviations occur
 - Identification and control of project risks
 - Prognosis of project progress and productivity

Software Quality Measurement

Chain of Reasoning



Software Quality Measurement

Popular Hypotheses in Theory and Practice

	/Fenton, Ohlsson 00/	/Basili, et al. 96/	/Cartwright, Shepperd 00/	/Basili, Perricone 84/	/Abreu, Melo 96/
Few modules contain the majority of faults	++	++	(+)	++	/
Few modules generate the majority of failures	++	/	/	/	/
Many faults during the module test means many faults during the system test	+	/	/	/	/
Many faults during the test means many failures during usage	--	/	/	/	/
Fault density of corresponding phases are constant between releases	+	/	/	/	/
Size measures are adequate for the fault prediction	+	/	+	-	/

++: strong conformation; +: light conformation; 0: no statement;
-: light refusal; -- strong refusal; /: not evaluated; ?: unclear

Software Quality Measurement

Popular Hypotheses in Theory and Practice: Findings I

- ☐ Faults are not uniformly distributed among software modules, but concentrated in few modules
- ☐ This modules generate the majority of all problems
- ☐ Larger module size does not necessarily mean more faults
- ☐ Many discovered problems during the tests does not mean that the software shows a lack of quality during practice
- ☐ There seem to be rules guaranteeing that subsequent developments provide similar results
- ☐ **Question**
 - **How can the few modules that contain the majority of faults be discovered?**

Software Quality Measurement

Popular Hypotheses in Theory and Practice

	/Fenton, Ohlsson 00/	/Basili, et al. 96/	/Cartwright, Shepperd 00/	/Basili, Perricone 84/	/Abreu, Melo 96/
Code complexity measures are better means for fault prediction	Better than size measures: -	WMC: +	WMC: /	Better than size measures: -	MHF: +
		DIT: ++	DIT: ++		AHF: 0
		RFC: ++	RFC: /		MIF: +
		NOC: ?	NOC: ?		AIF: (+)
		CBO: ++	CBO: /		POF: +
		LCOM: 0	LCOM: /		COF: ++

□ Object-oriented measures

- WMC (*Weighted Methods per Class*)
- DIT (*Depth of Inheritance Tree*)
- NOC (*Number Of Children*)
- CBO (*Coupling Between Object-classes*)
- RFC (*Response For a Class*)
- LCOM (*Lack of Cohesion on Methods*)
- MHF: *Method Hiding Factor*
- AHF: *Attribute Hiding Factor*
- MIF: *Method Inheritance Factor*
- AIF: *Attribute Inheritance Factor*
- POF: *Polymorphism Factor*
- COF: *Coupling Factor*

Software Quality Measurement

Popular Hypotheses in Theory and Practice: Findings II

- ☐ Several simple complexity measures (e.g., McCabe's cyclomatic number) are not better than size measures (e.g., LOC)
- ☐ Specific complexity measures display a good quality of fault prediction
- ☐ **Conclusion**
 - **A suitable combination of adequate complexity measures enables a directed identification of faulty modules**

Software Quality Measurement

Popular Hypotheses in Theory and Practice

	/Fenton, Ohlsson 00/	/Basili, et al. 96/	/Cartwright, Shepperd 00/	/Basili, Perricone 84/	/Abreu, Melo 96/
Model-based (<i>Shlaer-Mellor</i>) measures are suited for fault prediction	/	/	Events: ++	/	/
Model-based measures are not suited for size prediction	/	/	States: ++	/	/

Software Quality Measurement

Popular Hypotheses in Theory and Practice: Findings III

- ☐ It is possible to derive measures from software design to predict code size and fault numbers at an early stage

Software Quality Measurement

Conclusions

- ❑ Statistic methods for deriving software reliability are theoretically funded and applicable in practice
- ❑ Several plausible hypotheses are empirically falsified, but there is evidence that
 - Faults concentrates in few modules
 - This modules can be identified through measurements of
 - Code complexity
 - Complexity of design models
- ❑ Prediction of faults based on single measures (so called univariate analysis) is not possible. A suitable combination of measures (so called multivariate analyses) can produce reliable propositions
- ❑ It can be anticipated, that prediction models can be generated based on finished projects, as the similarity between subsequent projects is empirically supported

Literature

- Halstead M.H., Elements of Software Science, New York: North-Holland 1977
- Zuse H., Software Complexity - Measures and Methods, Berlin, New York: De Gruyter 1991