Software Measurement

Contents

Motivation

- 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)

- Remark

  Generally, the terms measure and metrics are used as synonyms. This is not quite correct. For this reason the correct term "measure" is used here

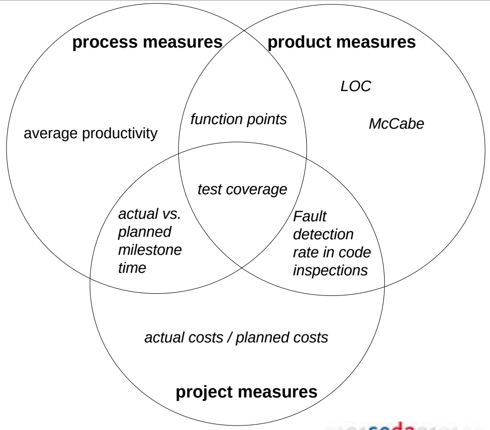
Motivation

- Software is an abstract, immaterial product
- Control of the quality, the complexity, the productivity, the development process, the costs and further important properties is difficult
- Idea: Definition of a quantified "sensor" which allows to draw conclusions w.r.t. interesting properties
- Measures quantify certain aspects of software. Measures can only indirectly point to potential sources of problems. A significant deviation of a measure from its usual value might be an indicator for a problem, but this is not guaranteed

Motivation
Application of Measures

- ☐ Control of software quality
- ☐ Control of software complexity
- ☐ Control of the software development process
- ☐ Costs and time prediction
- ☐ Costs and time tracing
- ☐ Definition of standards
- ☐ Early problem identification
- ☐ Comparison and evaluation of products
- ☐ Feedback concerning the introduction of new methods, techniques, and tools

Motivation
Application of Measures



Measure Types

- ☐ Product Measures
  - Information about properties of a product (complexity, size, ...).
  - Identification of critical product parts
  - Classification and comparison of products
- ☐ Process Measures
  - Information about properties of the software development process (productivity, failure costs, ...)
  - Control of the proper execution of process steps.
- ☐ Project Measures
  - Planning and tracking of a project

Measure Types

- ☐ Measures can involve several areas
  - Example: Function Point
  - Goal: evaluation of the development costs of a product on the basis of its functionality at an early stage, e.g. on the basis of the product specification
  - Measures the product size (function points)
  - Uses the maturity of the development process to convert function points into staff-months (table or curve)
  - Function point method involves product and process measures

Requirements

- Simplicity
  - Is the result so simple that it can be interpreted easily?
- Suitability
  - Does the measure show an appropriate correlation to the desired property?
- Robustness
  - Is the value of the measure stable with regard to minor changes of the measured software?
- Timeliness
  - Is the measure available early enough?
- Processability
  - Is it possible to process the measures (e.g. statistically ➔ scale type)?
- Reproducibility
  - The value of a measure should have an identical value for a particular product independently of the mode of generation

Requirements of Measures Reproducibility

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

    e = number of edges of a CFG; n = number of nodes of a CFG; CFG = control flow graph
    - Completely reproducible
  - Lines of Code (LOC)

    Count blank lines? Count comment lines?
    - Completely reproducible, if specified appropriately
  - Function Points: manual evaluation of complexities required
    - Not completely reproducible
  - Understandability
    - Not reproducible

Evaluation and Calibration of Measures Evaluation

- Suggesting lower or upper limits of measures is difficult
- Which values are to be regarded as "normal" might be determined based on expertise
- A deviation from the usual value might or might not be an indication of a problem

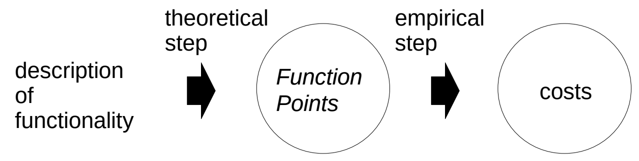
Evaluation and Calibration of Measures Calibration

- The assignment between measures and the relevant properties requires a calibration which has to be adapted to changed situations if necessary
- Empirical and theoretical models can be distinguished
- Examples
  - Theoretical model for costs (e.g., Halstead measures): $E = \ldots size^2 \ldots$

    The quadratic relation between costs and size was identified on the basis of theoretical considerations
  - Empirical model for costs: $E = \ldots size^{1,347} \ldots$

    The exponent 1,347 was determined on the basis of statistical data evaluation

Evaluation and Calibration of Measures
Example of a Mixed Theoretical and Empirical Model



Measure Scales

- If abstract properties are expressed as numerical values it has to be considered which operations are useful with the numerical values
- Examples
  - Measuring of length
    - Board a is one meter in length. Board b is two meter in length. Therefore, board b is twice as long as board a
    - This statement makes sense
  - Measuring of temperature
    - Today it is 20°C. Yesterday it was 10°C. Thus, today it is twice as warm as yesterday
    - This is wrong, the right answer is: Today the temperature is about 3,5 % higher than yesterday
  - Obviously there is a difference between the scale of the temperature in °C and the length in meters which leads to the fact that certain operations are not applicable to the temperature scale

Measure Scales

- Nominal scale
  - Free description of certain properties with labels
  - Inventory numbers of books of a library (DV 302, PH 002, CH 056, ...)
  - Names of different requirements engineering methods (SA; SADT, OOA; IM, ...)
- Ordinal scale
  - Mapping of an ordered aspect of a property to an ordered number of measurements in such a way that the order is maintained
  - Mapping of the arrival of patients to the waiting numbers in a doctor's surgery
- Interval scale
  - A scale which is still valid if transformations $g(x) = ax + b$, with $a > 0$ are applied to it
  - Temperature scales in degree Celsius or Fahrenheit. If F is a temperature in the Fahrenheit-scale the temperature C in the Celsius-scale can be calculated as follows: $C = {}^5/_9\,(F - 32)$. The relations between temperatures are maintained

Measure Scales

- Rational scale
  - A scale where measurements can be correlated (statements like double, half, three times as much, ... make sense)
  - Length in meters (it is twice as far from a to b as from c to d)
  - Temperature in Kelvin
- Absolute scale
  - A scale which is the only possibility to measure the issue
    - Counting
    - Probabilities

Measure Scales
Determination of Scales: Criteria for Ordinal Scales

- Ordinal scales are characterized by the property that the relation of the properties of two objects retaining the relation is mapped to the measurements
- The empirical relation concerning the properties is mapped to a corresponding formal relation of the measurements
- Empirical relation: for the software modules a and b the binary relations •≥ (more complex or equal complex), •> (more complex) and •≈ (equal complex) which can be applied to the modules are referred to as empirical relations
- The intuitive idea of complexity, as people would decide it, determines the empirical relation
- A is to be the set of all modules with a, b, c ∈ A. It is written

  a •> b; (a is more complex than b)

  a •≈ b; (a is as complex as b)

  a •≥ b ⇔ a •> b or a •≈ b;

Measure Scales
Determination of Scales: Weak Order

- A relation •≥ on a set A is called *order* if
  - a) ∀ x, y, z ∈ A: x •≥ y ∧ y •≥ z ⇒ x •≥ z (transitivity)
  - b) ∀ x ∈ A, ∃ y ∈ A: x •≥ y or y •≥ x (comparability)
  - Example: the relation "is ancestor of" on the set of persons
- An order is called *quasi order* if
  - c) ∀ x ∈ A: x •≥ x (reflexivity)
  - c implies b: every x is at least comparable to itself
  - Quasi orders can contain elements which cannot be ordered
  - Example: the identity "=" on every not empty set
- A *quasi order* is called *half order* if
  - d) x •≥ y ∧ y •≥ x ⇒ x = y (anti-symmetry)
  - Half orders also can contain elements which cannot be ordered
  - Example: the relation „≥" on the set of integers

Measure Scales
Determination of Scales: Weak Order

- A half order is called *linear* if
  - e) ∀ x, y ∈ A: x •≥ y or y •≥ x (connectivity, completeness)
  - Example: the relation "≥" on the set of integers
- Orders which fulfill the axioms a, c (and thus also b) and e, but not necessarily d, are called *weak order*
- In the following the empirical relation •≥ is considered. It is demanded that it generates a weak order on the set of the modules A, fulfilling the following axioms
  - axiom 1: reflexivity: a •≥ a, ∀ a ∈ A
  - axiom 2: transitivity: a •≥ b, b •≥ c ⇒ a •≥ c, ∀ a, b, c ∈ A

    (If the complexity of module a is greater equal the complexity of module b and the complexity of b is greater equal the complexity of c also the complexity of a is greater equal the complexity of c.)
  - axiom 3: connectivity (completeness): a •≥ b or b •≥ a, ∀ a, b ∈ A

Measure Scales
Determination of Scales: Ordinal Scale

- If the axioms 1 to 3 for the empirical relation •≥ concerning A are valid an ordinal scale exists
  - ((A, •≥), (ℜ, ≥), f), with a •≥ b ⇔ f(a) ≥ f(b), ∀ a, b ∈ A
  - (A, •≥) is the empirical relational system (modules and their empirical relation)
  - (ℜ, ≥) is the formal relational system (the numerical values of the measures and the corresponding formal relation ≥)
  - f is a measure

Measure Scales
Determination of Scales: Rational Scale

- A rational scale has to meet all criteria of an ordinal scale. The empirical and formal relational system has to be enhanced as follows
  - ((A, •≥, °), (ℜ, ≥, +), f),
  - with a •≥ b ⇔ f(a) ≥ f(b), (ordinal scale)
  - and f(a ° b) = f(a) + f(b), (rational scale)
  - ∀ a, b ∈ A
- ° is a binary operation for the empirical relational system. + is the corresponding binary operation for the formal relational system

Measure Scales
Determination of Scales: Rational Scale

- A measure f: A → ℜ which meets the requirements of the relational scale mentioned above exists when
  - (A, •≥) fulfils the axioms 1, 2, 3 (reflexivity, transitivity, connectivity)
  - axiom 4: a ° (b ° c) •≈ (a ° b) ° c, ∀ a, b, c ∈ A (associativity)
  - axiom 5: a •≥ b ⇔ a ° c •≥ b ° c ⇔ c ° a •≥ c ° b , ∀ a, b, c ∈ A (Monotony)
  - axiom 6: if c •> d, it is valid: ∀ a, b ∈ A, ∃ n ∈ ℵ, a ° nc •≥ b ° nd (archimedic axiom)

Measure Scales
Empirical Relation

- As the empirical relation •≥ is used in the definition of the scales it is required to determine it precisely
- Problem: a general definition is not possible, as the empirical relation reflects an intuitive idea of complexity
- But: It is possible to define the empirical relations with the aid of small modifications applied to an object to be measured, by considering whether these modifications lead to an increased, reduced or identical complexity
- Example: Lines of Code (LOC)
  - modification 1: add code line
  - modification 2: interchange code lines
  - modification 3: move code line

Measure Scales
Empirical Relation

- Idea concerning the measure LOC: The size is to be measured. The modification 1 increases the complexity of the modified module b compared to a, while the modifications 2 and 3 generate an identical complexity
  - M1: b •> a ⇒ LOC (b) > LOC (a)
  - M2: b •≈ a ⇒ LOC (b) = LOC (a)
  - M3: b •≈ a ⇒ LOC (b) = LOC (a)
- In this way the empirical relation •≥ was defined for the measure LOC
- If these properties of the modifications 1 to 3 are accepted LOC fulfils the criteria of the ordinal scale, i.e., then the measurements can be used as ordinal scale

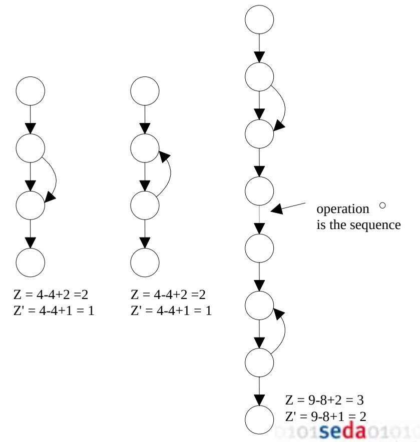
Measure Scales
Empirical Relation

- The measure LOC further fulfils the axioms 1 to 6 if as binary operation ° the textual chaining is used
- Further it is valid, that
  - LOC (a ° b) = LOC (a) + LOC (b) (additive)
- The values of the measure LOC can be used as a rational scale w.r.t. the agreed operation

Measure Scales
Example for the Measure Discussion – the Cyclomatic Number

- The cyclomatic number Z of a control flow graph g is defined as
  - Z = e - n + 2p
  - e = number of edges, n = number of nodes, p = number of the considered control flow graphs
- For a single module (p = 1) we get
  - Z = e - n + 2

Measure Scales
Example for the Measure Discussion – the Cyclomatic Number



Measure Scales
Discussion of the Measure Z: Ordinal Scale

- M1: add a node and an edge
  M2: move/displace an edge
  M3: add an edge
- M1: b •≈ a ⇒ Z (b) = Z (a)
  M2: b •≈ a ⇒ Z (b) = Z (a)
  M3: b •> a ⇒ Z (b) > Z (a)
- With regard to the specified modifications we see

  b •≥ a ⇔ Z(b) ≥ Z(a);

  i.e. the values can be used as ordinal scale

Measure Scales
Discussion of the Measure Z: Rational Scale

- Obviously Z does not fulfill the condition of the additivity concerning the operation ° (sequence), i.e., $Z(a) + Z(b) \neq Z(a \circ b)$
- The values of the measures Z concerning the operation ° (sequence) cannot be used as rational scale. On the other hand Z fulfils the axioms 1 to 6. Thus, a measure Z' must exist which is additive

  $Z' = Z - 1 = e - n + 1$

  (Can be used as rational scale concerning the operation *sequence*.)

Data Acquisition for Measuring

- Measures are directly enumerable, calculated or evaluated parameters, if necessary also a corresponding combination

  ➔ input parameters (primary data) for the generation of measures have to be collected
- Example
  - enumerable measure: Lines of Code
  - calculated measure: MTTF
  - evaluated measure: function points
- Questions
  - Which primary data can be determined automatically (e.g. from the source code)?
  - Which primary data have to be collected manually?
  - Which primary data can only be gathered based on expertise?

Data Acquisition for Measuring

- Principle
  - Collect automatically and tool supported as much of the required information as possible!
  - good cost-value ratio
- Pure product measures often can be collected fully automatic from the product
- Collection of the primary data directly from the product
- Application of measuring tools
- Example
  - Lines of Code
  - McCabe's cyclomatic number
  - Halstead's measures

➔ basically complexity measures

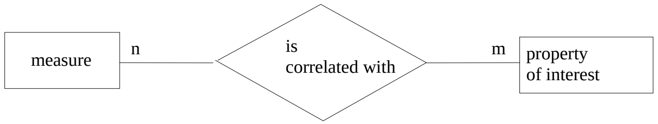
Data Acquisition for Measuring

- But: some measures relating to products cannot be derived from them
- Example
  - MTTF (mean time to failure) or
  - Faults / LOC
    - Relate to a product
    - Require an error statistics for this product

➔ basically quality measures

Data Acquisition for Measuring Process Measures

- Parts of the required data can be gained with corresponding tool application directly from the process
  - Example: test coverage



- Parts have to be taken manually
  - Example: costs, time, error statistics

Data Acquisition for Measuring Important Primary Data

- Quality-related
  - Number, type and cause of faults
  - Number of problem messages
  - Number of changes
- Costs-related
  - Costs of fault corrections
  - Costs and time exposure (development and testing costs per process step)
- Product-related
  - Size of the product using an appropriate measure (LOC, pages, processes, number of entries in the data dictionary, function points, number of modules, ...)

Data Acquisition for Measuring Correcting Side-Effects

- Many measures do not only measure a single property, but are influenced by several factors
- Collection of the primary influencing variable
- Collection of side effects
- Example
  If the increase of the MTTF is used as a measure for reliability of a system this relation may be distorted if only failure statistics are used
- Causes
  - During the observation period the number of the software systems in operation is usually not constant, so that the failure probability declines or rises
  - Larger modifications (new version, functional enhancement, etc.) increase the failure probability

Data Acquisition for Measuring Correcting Side-Effects

- Consequence
  - Major influencing parameters regarded as side-effects have to be measured in order to correct the primary measure from their influence
  - Example MTTF
    - Recording the number of installed systems over time
    - Recording important events: new version, etc.

Important Measures
The Halstead Measures

- Set of measures concerning different aspects, e.g., complexity, size, costs, etc.
- Are all based on theoretical considerations
- Are based on the program text (number of the different operands and operators and total number of the operands and operators)
- Halstead's costs measure E does not necessarily fulfill the criterion of timeliness
- No direct relation to natural parameters; unnatural measures
- Common as measures in analysis and test tools
- Remark: Halstead's costs measure determines a quadratic dependence between the size of a module and the costs for its implementation ➔ modularization

Important Measures
The Halstead Measures

- The four basic parameters of the Halstead measures are
  - η_1 – number of operators
  - η_2 – number of operands
  - N_1 – total number of operators
  - N_2 – total number of operands
- From these four measures two further simple measures can be derived
  - $\eta = \eta_1 + \eta_2$ – size of the vocabulary
  - $N = N_1 + N_2$ – length of the implementation
- By considering some combinatorial rules the formula for the calculated program length N is derived
  - $N = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$

Important Measures
The Halstead Measures

- Program volume V

  $V = N \log_2 \eta$
- V is the volume of the program in bits provided that a binary coding with a fixed word length of the vocabulary is used
- The potential program volume V* depends only on the algorithm, not on the programming language used for the implementation

  $V^* = (N_1^* + N_2^*) \log_2 (\eta_1^* + \eta_2^*)$

  $= (2 + \eta_2^*) \log_2 (2 + \eta_2^*)$
- The quotient of the potential volume V* and V is called level

Important Measures
The Halstead Measures

- Every implementation has a level L which is smaller or at best equal one. The more L approximates the value one, the more appropriate is a programming language for the implementation of a given algorithm
- A measure for the difficulty to implement an algorithm in a programming language is the reciprocal D of the level (Difficulty)

  $$D = \frac{1}{L}$$
- A programming language inappropriate for the implementation of an algorithm causes a rise of the volume V and thus also of the difficulty D

Important Measures
The Halstead Measures

- The volumes L und D are a measure for the problem adequacy of the used programming language and for the difficulty to implement a given algorithm in a particular language
- The Effort E necessary to code an algorithm is proportional to the program volume and to the difficulty of the coding. Difficulty D is the reciprocal of the program level L
- Effort E then can be defined to

$$E = \frac{V}{L} = \frac{V^2}{V^*}$$

Important Measures
The Halstead Measures: Example

```
PROCEDURE CountChars          (VAR VowelNumber : CARDINAL;
                                    VAR TotalNumber : CARDINAL);
    VAR Char : CHAR;
    BEGIN
      READ (Char);
      WHILE ((Char ≥ „A“) AND (Char ≤ „Z“)
       AND TotalNumber < MAX (CARDINAL))) DO
             TotalNumber := TotalNumber + 1;
             IF ((Char = „A“) OR (Char = „E“) OR (Char = „I“)
                   OR (Char = „O“) OR (Char = „U“)) THEN
                                 VowelNumber := VowelNumber + 1;
             END; (* IF *)
             READ (Char);
      END; (* WHILE *)
    END CountChars;
```


Important Measures
The Halstead Measures: Example

- Number of operators: $\eta_1 = 20$

  Number of operands: $\eta_2 = 10$

  Total number of operators: $N_1 = 58$

  Total number of operands: $N_2 = 26$
- From this follows

  $N = N_1 + N_2 = 84$

  and

  $$N = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2$$
  $$= 20 \log_2 20 + 10 \log_2 10$$
  $$= 86{,}4 + 33{,}2$$
  $$= 119{,}6$$

Important Measures
Data Measures

- The primary purpose of programs is the processing of data
- Live Variables
  - Is based on the assumption that the generation of a statement is the more difficult the more variables have to be considered at the execution of this statement
  - Definition: a variable "lives" within a procedure from its first to its last reference

Important Measures
Data Measures

- Example

```
1    PROCEDURE MinMax (VAR Min: CARDINAL; VAR Max: CARDINAL);
2    VAR Help : CARDINAL;
3    BEGIN
4         IF Min > Max THEN
5              Help := Min;
6              Min := Max;
7              Max := Help
8         END;
9    END MinMax;
```


Important Measures
Data Measures

line	live variables	number
4	Min, Max	2
5	Min, Max, Help	3
6	Min, Max, Help	3
7	Help, Max	2

- LV: medium number of live variables = $\frac{\text{total number of live variables}}{\text{number of executable statements}}$
- In the example: $\overline{LV} = \frac{10}{4} = 2{,}5$

Important Measures
Data Measures: Variable Span

- Additionally, the span of the variable references is important
- Example
  - Min is referenced at lines 4, 5 and 6
    Max is referenced at lines 4, 6 and 7
    Help is referenced at lines 5 and 7
  - The spans of Min are: 1 line, 1 line;
    averaging: 1
  - The spans of Max are: 2 lines, 1 line;
    averaging: 1,5
  - The spans of Min are: 2 lines;
    averaging 2
  - The average span of all variables is 1,4

Important Measures
The Cyclomatic Complexity

- A common complexity measure
- Often has the aura of an "important" key value
- Derives from graph theory (strongly connected graphs) and thus can be related to control flow graphs and consequently to programs represented by these graphs
- Formula: e – n + 2
  (e = number of edges, n = number of nodes)
- Very easy to determine as for programs highly dependent on the number of decisions (it is simply the number of decisions + 1)
- Appropriate as a complexity measure if the number of decisions says much about the complexity of the program
- Probably the most widespread measure in analysis and test tools

Important Measures
More Control Structure Measures

- Nesting
  - Every statement is assigned a nesting level according to the following rules
    - To the first executable statement the value 1 is assigned
    - All statements that belong to a statement sequence are on the same nesting level
    - If a statement a is on the nesting level l and statement b is within a selection or loop controlled by a, statement b has the nesting level l + 1
  - The value of this measure is the arithmetic mean of the nesting levels of all statements

Importance of Software Measuring

- Software measurement is, e.g., important for the following areas
  - Flat management structures
  - Compliancy to certain software engineering standards
  - High capability maturity levels

Software Measuring and Flat Management Structures

- Flat management structures are a trend
  - A manager supervises more developers
  - The supply and aggregation of information is not done anymore via the middle management, but via automated measuring systems
  - Interventions of the management are required only if measurements indicate problems

Software Measuring and Software Engineering Standards

- Standards increasingly gain importance in software engineering (e.g. ISO 9001)
  - Proof of qualification for potential clients
  - Marketing criterion; differentiation from non-accredited competitors
  - Important in the context of product liability
  - In some areas definitely required
  - All standards underline the importance of a systematic procedure, transparency, and control of the development process

➔ This can be proven with the aid of corresponding measures

Software Measuring and the Capability Maturity Model

- Capability Maturity Model classify the maturity of a software development process using maturity levels. The model used by the SEI uses the following levels: 1-initial, 2-repeatable, 3-defined, 4-measured, 5-optimizing
- The attainment of the maturity levels 4 and 5 is possible only with the existence and use of a measuring system which enables the following operations
  - Measuring of productivity and quality
  - Evaluation of projects on the basis of these measuring
  - Identification of deviations
  - Corrective actions in the case of deviations
  - Identification and control of project risks

Case Study

- Failure lists are to be evaluated to make a decision for systematic techniques, methods, and tools due to the number of failures, their causes, and the costs for their correction
- The goal is
  - A clear reduction of the number of faults to reduce the problems of clients with the software and to demonstrate high quality
  - The prevention or early detection of costly failures in order to save money

Case Study

- Failure list

No.	problem description	date of message	corrected at	correction time (workdays)	correction costs (MDays)	fault cause
1	....	07.03.92	20.04.92	25	50	faulty/defective requirement
2		11.04.92	13.04.92	0,5	0,5	coding fault (loop)
3		13.04.92	05.05.92	5	5	module specification
4		04.05.92	06.05.92	0,3	0,3	wrong path requirement
5		23.05.92	05.06.92	7	7	interface between modules
6		01.06.92	28.06.92	15	15	missing functionality
7		02.06.93	15.06.93	0,2	0,2	missing initialisation
8		15.06.92	18.06.92	0,4	0,4	consecutive fault by fault correction
9		01.07.92	10.07.92	2,5	5	wrong modularisation
10		03.07.92	30.08.92	28	35	performance too low
11		02.08.92	05.08.92	0,6	0,6	previous fault correction
12		29.08.92	01.09.92	0,8	0,4	wrong data type used
13		04.09.92	06.09.92	1	1	algorithmic fault
14		28.09.92	18.11.92	22	22	requirement misunderstood
15		11.11.92	10.12.92	13	13	requirement wrong
16		20.12.92	23.12.92	0,2	0,2	required data not provided
17		02.01.93	31.01.93	9	2	coding bug
18		13.02.93	15.02.93	1	0,4	missing initialisation
	....					

observed period: 343 days

Case Study

- The average MTTF is selected to measure reliability
- One receives
  - MTTF = 343 days / 17 = 20,2 days
- The average correction costs are used additionally
  - The average fault correction costs 8,8 Man days

Case Study

- The faults have different causes which can be attributed to different phases
  - Definition phase (5 faults): 1, 6, 10, 14, 15
    Average costs: 27 MD
    Total costs: 135 MD
  - Design phase (3 faults): 3, 5, 9
    Average costs: 5,7 MD
    Total costs: 17 MD
  - Implementation phase (10 faults): 2, 4, 7, 8, 11, 12, 13, 16, 17, 18
    Average costs: 0,6 MD
    Total costs: 6 MD
- Costs reduction is achieved best by improvements in the definition phase, as here the major part of the correction costs is caused, although more faults are created in the implementation phase

Case Study

- A reduction of the number of faults is achieved best by improvements in the implementation or unit test phase

⇒ Application of corresponding techniques and tools
  - ⇒ SA, OOA, IM, RT, reviews, ...
  - ⇒ Structured programming, code generation, systematic testing, ...

- Further observation of the measures in order to control effects

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