Bachelor Studies in Computer Science: UNIBE

# Software Metrics and Cost Estimation - an Introduction

2021-12-08 | Dr. Simon Moser | simon.moser@solutionboxx.ch

# Topics

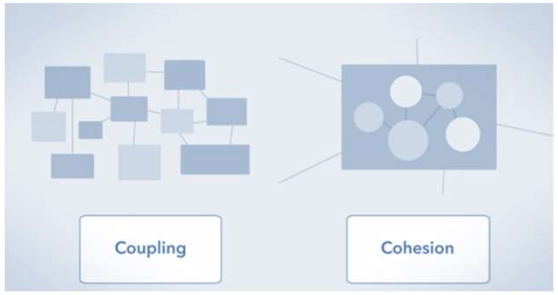
SolutionBoxX

1. **Why measure «things» in Software Engineering?**
2. Software Metrics Basics
3. Software Product Metrics – Measuring Code
4. Software Product Metrics – Measuring Specifications
5. Software Process Metrics
6. Estimation (of Project Effort and Duration )

SolutionBoxX

# To make Software Quality Principles measurable.

Examples:




$n \in \mathbb{N}$ $q \in \mathbb{Q}$

- Small **sized** methods/classes
- Low **coupling**
- High **cohesion**
- Enforcing code reviews
- Etc.

**You cannot control what you cannot measure.**

Tom DeMarco

As quoted in Software Metrics: A Rigorous and Practical Approach, page 11.

SolutionBoxX

# To make Software Quality Principles measurable ...

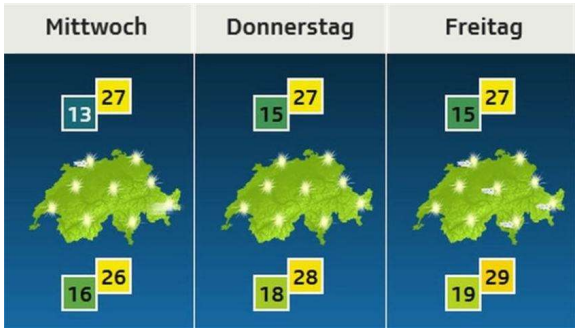
... **BUT**:

- Measurements just forecast quality or absence of quality (maintenance cost and defect rates)
- Warning thresholds must be established.
- No absolute world-wide accepted values for ‘good’ or ‘bad’.

SolutionBoxX

# To allow parametric (=measurement-based) Estimation.

Examples:




Imagine weather forecasts without measurements.

- Coding effort
- Also: Specification, Testing, Management effort
- Software development productivity
- Project duration
- Etc.

**Control = Loop of Measure, Estimate, Measure, … (“Learning”).**

# This introduction is not about ...

SolutionBoxX

- Response time or other
  software performance measurements

- CPU MHz or other
  hardware performance measurements

- Algorithmic complexity

# Topics

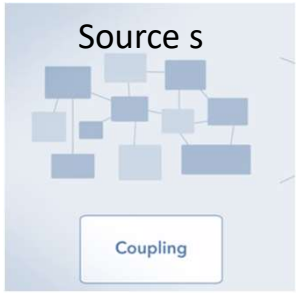
SolutionBoxX

1. Why measure «things» in Software Engineering?
2. **Software Metrics Basics**
3. Software Product Metrics – Measuring Code
4. Software Product Metrics – Measuring Specifications
5. Software Process Metrics
6. Estimation (of Project Effort and Duration )

SolutionBoxX

# Definition

Example:




$n \in \mathbb{N}$

function **coupling** (Source s) as Integer

[ C-style: int **coupling** (char* source); ]

## Software Metric

:= transforms a (possibly structured and multi-dimensional) system or management **artefact a of artefact type A** into a value v of a defined single-dimensional **scale S**.

**:=** triplet of **(A, S, measurement procedure P)**

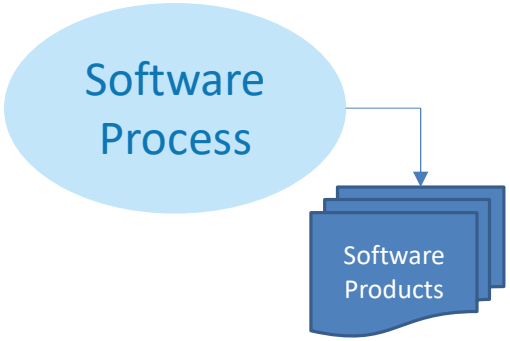
**:= function P(A) as S** and **v = P(a)**

## Software Quality Metric

:= A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which software possesses a given attribute that affects its quality. IEEE Std. 1061

SolutionBoxX

# Classification




**Product Metrics** measure

- Code
- (formal or modelled) Specifications/Requirements
- Test Cases, Test Executions

**Process Metrics** measure

- Effort (of some activity)
- Duration
- Defects, Reviews
- Productivity, and other derived metrics

Attn: other classifications exist

SolutionBoxX

# What makes a good metric?

**Well-defined:** A, S and P are defined

**Reliable:** repeating the measurement yields the same result on same input

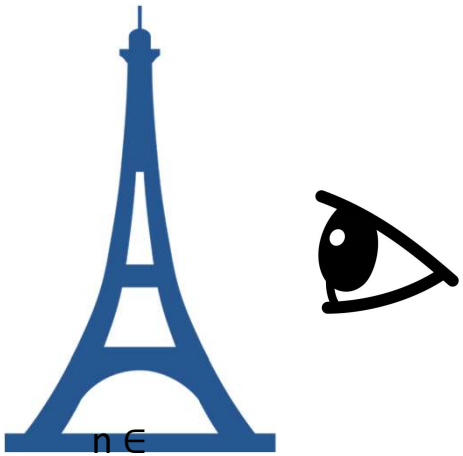
**Person-independent:** the result does not depend on an individual or a group of individuals

**Efficient:** the measurement takes minimal amounts of time and money

**Purposeful:** the measurement is used for the benefit of a software process stakeholder

SolutionBoxX

# Rating (also: Estimating)



n ∈

HeightInMeters (Building b) as Integer = Simon says n

**Well-defined:** A, S and P are defined

**Reliable:** repeating the measurement yields the same result on same input

**Person-independent:** the result does not depend on an individual or a group of individuals

**Efficient:** the measurement takes minimal amounts of time and money

**Purposeful:** the measurement is used for the benefit of a software process stakeholder

SolutionBoxX

# Scale Types

| | | | | | |
|---|---|---|---|---|---|
| 1 | 8 | **POLLINGER** JANNINE | 1995 | BANNALP-WOLFENSCHIESSEN | **48.29** | 00.00 |
| 2 | 1 | **ERNST** IRINA | 1995 | ESCHOLZMATT | **49.83** | 01.54 |
| 3 | 6 | **KNÜSEL** NICOLE | 1996 | ESCHOLZMATT | **50.93** | 02.64 |
| 4 | 13 | **GERBER** NICOLE | 1996 | ESCHOLZMATT | **51.58** | 03.29 |
| 5 | 11 | **NIEDERBERGER** MARIA | 1995 | BECKENRIED | **52.67** | 04.38 |
| 6 | 15 | **ZIHLMANN** CARLA | 1995 | SCHUEPFHEIM | **53.97** | 05.68 |
| 7 | 9 | **DAHINDEN** ADRIANA | 1996 | GOTTHARD-ANDERMATT | **54.00** | 05.71 |
| 8 | 7 | **WICKI** CARLA | 1996 | FLUEHLI | **54.89** | 06.60 |
| 9 | 14 | **SCHMID** YVONNE | 1995 | MALTERS | **55.62** | 07.33 |
| 10 | 12 | **WICKI** ROMINA | 1996 | MARBACH | **55.64** | 07.35 |
| 11 | 51 | **FELDER** KARIN | 1995 | WERTHENSTEIN | **57.03** | 08.74 |
| 12 | 10 | **SCHNIDER** KARIN | 1995 | HASLE | **57.28** | 08.99 |
| 13 | 5 | **MATHIS** ISABEL | 1995 | BANNALP-WOLFENSCHIESSEN | **57.69** | 09.40 |
| 14 | 2 | **SOMMER** STEFANIE | 1996 | MALTERS | **59.20** | 10.91 |
| 15 | 4 | **RENNER** CAROLINE | 1995 | GOTTHARD-ANDERMATT | **1:00.70** | 12.41 |

- Nominal
- Ordinal (ranking)
- Cardinal – relative (correct differences)
- Cardinal – absolute (counting, 0 anchor)

➔ Statistics only ‘safe’ when a scale is absolute.

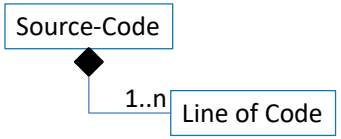
# Topics

SolutionBoxX

1. Why measure «things» in Software Engineering?
2. Software Metrics Basics
3. **Software Product Metrics – Measuring Code**
4. Software Product Metrics – Measuring Specifications
5. Software Process Metrics
6. Estimation (of Project Effort and Duration )

SolutionBoxX

# Lines of Code (LOC)




:= (Source-Code, $\mathbb{N}$, count LOC)

- Specific per Programming Language
- Variants: including/excluding comments
- Sometimes: sub-expressions counted
- C/C++: counting «;» (but what about the , operator?)

➔ Recommendation: max 100 per method

Also:
- KLOC = 1’000 LOC
- Delivered Source Instruction (DSI), KDSI

SolutionBoxX

# Lines of Code (LOC)

## Positive Aspects

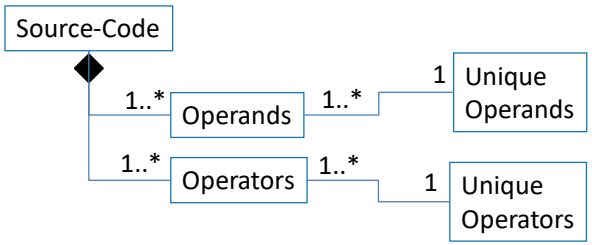
- Simple, automated measurement
- Absolute scale
- Predicts Re-Engineering / Porting effort

## Negative Aspects

- Too simple
- Available late in the development process
- Dangerous usage:
  derived productivity metric LOC/Person Day

SolutionBoxX

# Halstead's Volume (V)




Examples:
https://verifysoft.com/de_cmtpp_mscoder.pdf
X.N.Cullmann, K. Lambertz

:= (Source-Code $\mathbb{R}$ , V (siehe unten))

- $N_1$ = #Operands, $N_2$ = #Operators
  ≈ the 'words' used
- $\eta_1$ = #Unique operands, $\eta_2$ = #unique operators
  ≈ the 'vocabulary' used
- $\text{Log}_2 (\eta_1 + \eta_2)$ // number of binary decisions for 'which operator/operand to choose?' answer
- $V = (N_1 + N_2) * \text{Log}_2 (\eta_1 + \eta_2)$

➔ Recommendation: max. 1'000 per method

Also:
- more Halstead metrics: length, difficulty, effort

SolutionBoxX

# Halstead's Volume (V)

## Positive Aspects

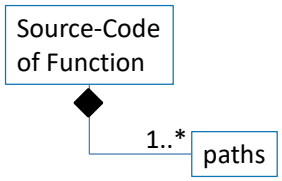
- automated measurement
- Absolute scale
- Reflects cognitive effort to understand (and write) software
- Language-independent
- Better for 'size' than LOC
- Predicts Code-Review effort

## Negative Aspects

- Available late in the development process
- Dangerous usage:
  derived productivity metric V/Person Day
- Only usable per function

SolutionBoxX

# McCabe's Cyclomatic Complexity




Examples:
https://verifysoft.com/de_cmtpp_mscoder.pdf
X.N.Cullmann, K. Lambertz

:= (Source-Code of 1 Function, $\mathbb{N}$ , count paths)

- Each IF creates an additional path
- Each CASE creates an additional path
- Each loop condition (FOR, WHILE) creates an additional path
- Each CATCH creates an additional path

➔ Recommendation: max. 10 per function (method)

SolutionBoxX

# McCabe's Cyclomatic Complexity

## Positive Aspects

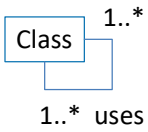
- automated measurement
- Absolute scale
- Language-independent
- Predicts Test complexity / effort

## Negative Aspects

- Available late in the development process
- Only covers 1 aspect of complexity
- Only usable per function

SolutionBoxX

# Class Coupling




:= (Class (Source-Code), $\mathbb{N}$ , count used other classes)

- subclassing
- object types (parameters, variables)
- return value types

➔ Recommendation: max. 5 per class

SolutionBoxX

# Class Coupling

## Positive Aspects

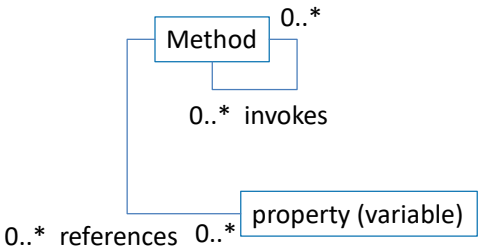
- automated measurement
- Absolute scale
- Language-independent
- Predicts Test complexity / effort
- Predicts Re-Test maintenance effort

## Negative Aspects

- Available late in the development process
- Only covers 1 aspect of coupling
- Only usable per class (and in object-oriented languages)

SolutionBoxX

# Lack of Cohesion of Methods (LCOM)




:= (set of methods (Source-Code), $\mathbb{N}$, count unlinked sub-sets)

- 2 methods are linked when 1 invokes the other
- 2 methods are linked when they reference the same property (variable)

Normally, the set of all methods in a class are attributed the LCOM value. But the definition is open for any set of methods.

➔ Recommendation: max. 1 per class, i.e. all methods should be linked

Note that high values for LCOM are bad.

SolutionBoxX

# Lack of Cohesion of Methods (LCOM)

## Positive Aspects

- automated measurement
- Absolute scale
- Language-independent
- Predicts Re-Test maintenance effort

## Negative Aspects

- Available late in the development process
- Does not look at method usage outside the given set of methods
- Needs ‘exception case’ handling e.g. when 2 is used as the warning threshold:

  e.g. a persistence class (‘entity’) typically has unlinked methods per property

# Topics

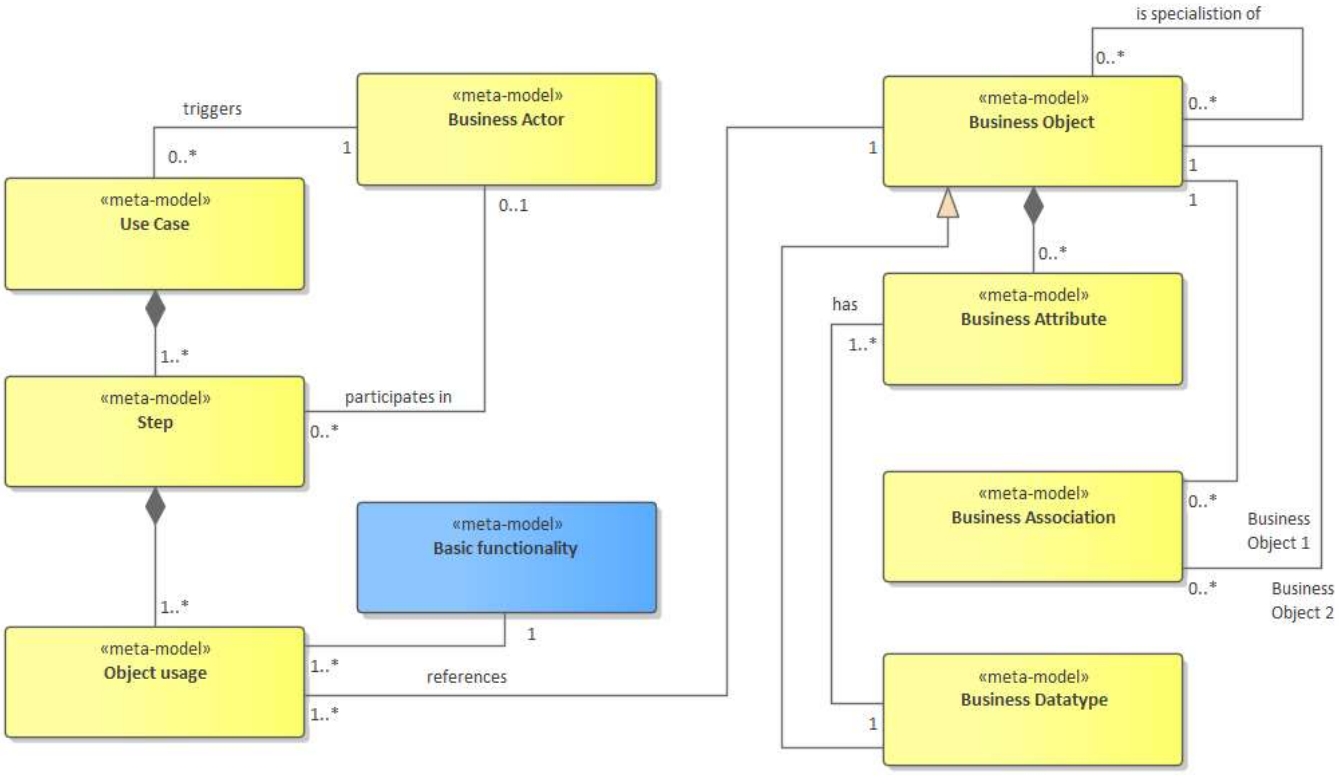
SolutionBoxX

1. Why measure «things» in Software Engineering?
2. Software Metrics Basics
3. Software Product Metrics – Measuring Code
4. **Software Product Metrics – Measuring Specifications**
5. Software Process Metrics
6. Estimation (of Project Effort and Duration )

SolutionBoxX

# What is a Specification?

**Dynamic Model** + **Structural Model**



SolutionBoxX

# Function Points (FP)

Background Information:

- The original function points were developed and published in 1979 by Allan J. Albrecht (IBM). Originally, these were direct person-days of coding. Then they wanted to prove productivity progress and "froze" the person-days to points.
- The original function points still had an 'adjustment factor' of 70%-130%. But this was removed in 2003, mainly because it made something mathematically 'undefined' out of an absolute scale..
- There is a standard (ISO standard ISO/IEC 20926) and a user group (www.ifpug.org).

:= (Specification, $\mathbb{N}$, count FP of Use Cases and Business Objects)

«Old-fashioned» terminology:

- Just 3 Basic functionalities allowed:
  - Input = create, update or delete information
  - Query = output information without ‘much’ logic
  - Output = output information with transformation logic
- Use Cases are typed as Input, Output or Query
- Business Objects are called
  - Internal Logical Files (ILF), if Input-ed in 1..* Use Case
  - External Logical Files (ELF), else

SolutionBoxX

# Function Points (FP)

| Classification | ILF<br>Business Object (BO) | ELF<br>read-only Business Object(BO) | Input-Use Case<br>UseCase with Persistence | Output-Use Case<br>Search/Display with Logic | Query-Use Case<br>1:1 Business Object Display |
|---|---|---|---|---|---|
| **EASY** | 7 FP<br>#Attributes <= 4 and #Associations <= 1 | 5 FP<br>#Attributes <= 4 and #Associations <= 1 | 3 FP<br>#Input-BO = 1 and (#Input-BO + #Output-BO + #Query-BO) <= 2 | 4 FP<br>#Output-BO = 1 and (#Output-BO + #Query-BO) <= 2 | 3 FP<br>#Query-BO <= 2 |
| **MEDIUM** | 10 FP<br>not EASY and not COMPLEX | 7 FP<br>not EASY and not COMPLEX | 4 FP<br>not EASY and not COMPLEX | 5 FP<br>not EASY and not COMPLEX | 4 FP<br>not EASY and not COMPLEX |
| **COMPLEX** | 15 FP<br>#Attributes >= 10 and #Associations >= 3<br>or<br>#Attributes >= 30 and #Associations >= 1<br>or<br>#Attributes >= 4 and #Associations >= 5 | 10 FP<br>#Attributes >= 10 and #Associations >= 3<br>or<br>#Attributes >= 30 and #Associations >= 1<br>or<br>#Attributes >= 4 and #Associations >= 5 | 6 FP<br>#Input-BO >= 3 and (#Input-BO + #Output-BO + #Query-BO) >= 6 | 7 FP<br>#Output-BO >= 3 and (#Output-BO + #Query-BO) >= 6 | 6 FP<br>Query-BO >= 6 |

SolutionBoxX

# Function Points (FP)

## Positive Aspects

- automated measurement (when the modelling is done)
- Absolute scale
- Available early (as soon as the specification is modelled)
- Predicts subsequent efforts (detailed design, development, testing, …)

## Negative Aspects

- Needs a modelled specification according to the Function Points conventions
- Limited modelling (e.g. no business objects subtyping)
- The ‘same’ system can be modelled differently by different persons

SolutionBoxX

# System Meter (factually 'System Description' Meter)

**Meta-Model**

| Name | Definition | New | External System Meter | Internal System Meter | System Meter |
|---|---|---|---|---|---|
| xxxxx | yyyyy aaaa bb cccccc ddd e ffff gggggg | Y | 1 | 1 | 2 |
| yyyyy | xx zzzzz xxx vvvv bb gbb cccccc ddd e ffff g | N | 1 | 2 | 1 |
| zzzzz | yyyy cccccc ddd e xxx zzzzz cccccc ddd e ffff cccccc ddd e ffff cccccc ddd e ffff | N | 1 | 1 | 1 |
| xxx vvvv | xxx zzzzz xxx vvvvbb cccccc ddd e ffff g | Y | 2 | 1 | 3 |
| xxx zzzzz | yyyy xxxxx zzzzz xxx vvvvbb cccccc | Y | 2 | 2 | 4 |

**System Meter :=**

$\Sigma_{all}$ **token (Name)** // external System Meter

**+**

$\Sigma_{new}$ **#Links (Definition)** // internal System Meter

TOTAL

**11 System Meter**

SolutionBoxX

# System Meter

## Positive Aspects

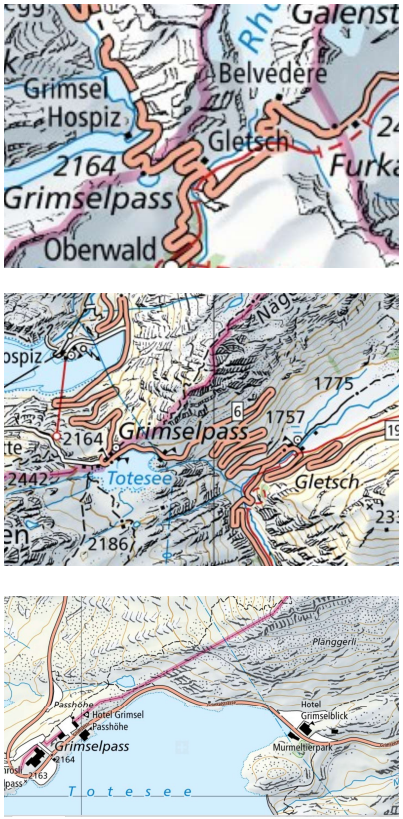
- automated measurement (when the modelling is done)
- Absolute scale
- Available very early (minimal modelling effort)
- Takes reuse into account
- Predicts subsequent efforts (detailed design, development, testing, …)
- Can be applied to all levels of system descriptions (from overview models to code)

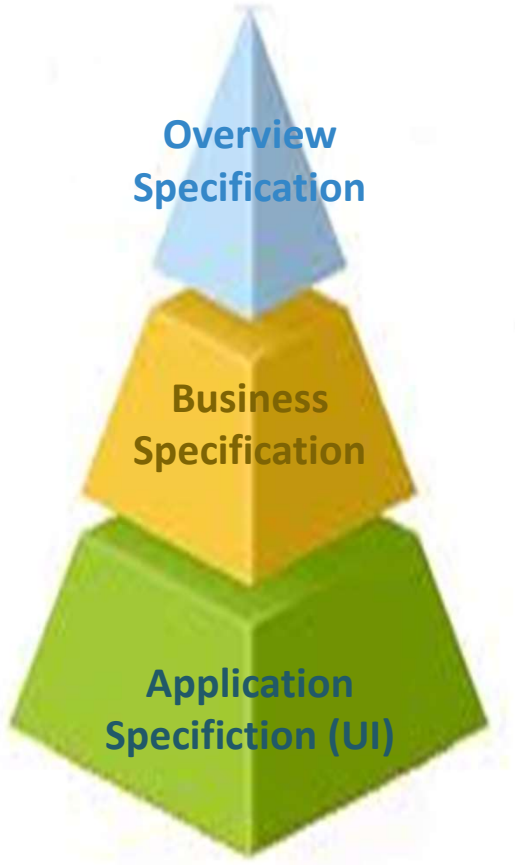
## Negative Aspects

- Needs a modelled specification / formal system description
- The ‘same’ system can be modelled differently by different persons
- Not well known

SolutionBoxX

# About Levels of System Specification Descriptions

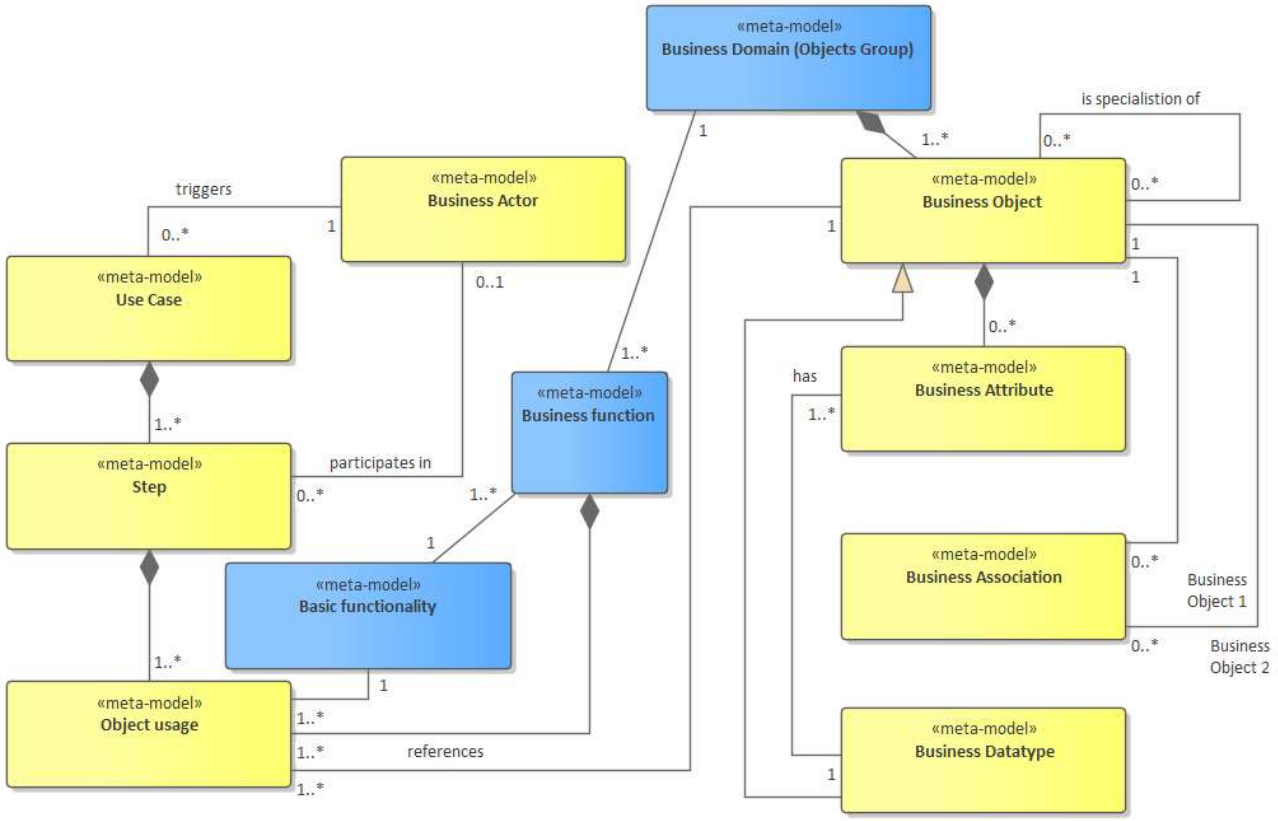


SolutionBoxX

# An Overview specification

**Dynamic Model** **+** **Structural Model**



SolutionBoxX

# An Overview specification Example (1/3)

| Business Domain | #Business Objects |
|---|---|
| Produkte | 5 |
| Kunden | 3 |
| Bestellungen | 4 |

| Basic Functionality | #Business Operations |
|---|---|
| erstellen | 1 |
| suchen/anzeigen | 1 |
| ändern | 1 |
| löschen | 1 |

| Business Domain | Basic Functionality |
|---|---|
| 01 - Produkte | suchen/anzeigen (R) |
| 02 - Kunden | suchen/anzeigen (R) |
| 03 - Bestellungen | erstellen (C) |
| 03 - Bestellungen | suchen/anzeigen (R) |
| 03 - Bestellungen | ändern (U) |

SolutionBoxX

# An Overview specification Example (2/3) – Reuse Analysis

| Business Domain | #Business Objects | |
|---|---|---|
| Produkte | 5 | reused |
| Kunden | 3 | reused |
| Bestellungen | 4 | new |

| Basic Functionality | #Business Operations | |
|---|---|---|
| erstellen | 1 | reused |
| suchen/anzeigen | 1 | reused |
| ändern | 1 | reused |
| löschen | 1 | reused |

| Business Domain | Basic Functionality | |
|---|---|---|
| 01 - Produkte | suchen/anzeigen (R) | reused |
| 02 - Kunden | suchen/anzeigen (R) | reused |
| 03 - Bestellungen | erstellen (C) | new |
| 03 - Bestellungen | suchen/anzeigen (R) | new |
| 03 - Bestellungen | ändern (U) | new |

SolutionBoxX

# An Overview specification Example (3/3) – PRE System Meters

| Business Domain | #Business Objects | | external SM | internal SM | PRE System Meter Total = 10 | Total = 27.8% |
|---|---|---|---|---|---|---|
| Produkte | 5 | reused | 1 | 6 | 1 | 2.78% |
| Kunden | 3 | reused | 1 | 4 | 1 | 2.78% |
| Bestellungen | 4 | new | 1 | 7 | 8 | 22.22% |

| Basic Functionality | #Business Operations | | external SM | internal SM | PRE System Meter Total = 5 | Total = 13.9% |
|---|---|---|---|---|---|---|
| erstellen | 1 | reused | 1 | 2 | 1 | 2.8% |
| suchen/anzeigen | 1 | reused | 2 | 5 | 2 | 5.6% |
| ändern | 1 | reused | 1 | 2 | 1 | 2.8% |
| löschen | 1 | reused | 1 | 1 | 1 | 2.8% |

Total
36 PRE System Meter

| Business Domain | Basic Functionality | | external SM | internal SM | PRE System Meter Total = 21 | Total = 58.3% |
|---|---|---|---|---|---|---|
| 01 - Produkte | suchen/anzeigen (R) | reused | 1 | 10 | 1 | 2.78% |
| 02 - Kunden | suchen/anzeigen (R) | reused | 1 | 6 | 1 | 2.78% |
| 03 - Bestellungen | erstellen (C) | new | 1 | 4 | 5 | 13.89% |
| 03 - Bestellungen | suchen/anzeigen (R) | new | 1 | 8 | 9 | 25.00% |
| 03 - Bestellungen | ändern (U) | new | 1 | 4 | 5 | 13.89% |

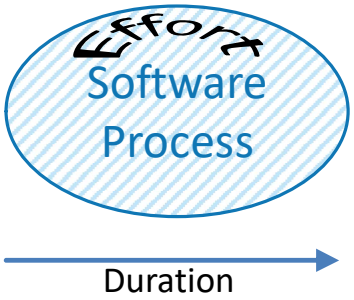
# Topics

SolutionBoxX

1. Why measure «things» in Software Engineering?
2. Software Metrics Basics
3. Software Product Metrics – Measuring Code
4. Software Product Metrics – Measuring Specifications
5. **Software Process Metrics**
6. Estimation (of Project Effort and Duration )

SolutionBoxX

# Effort and Duration



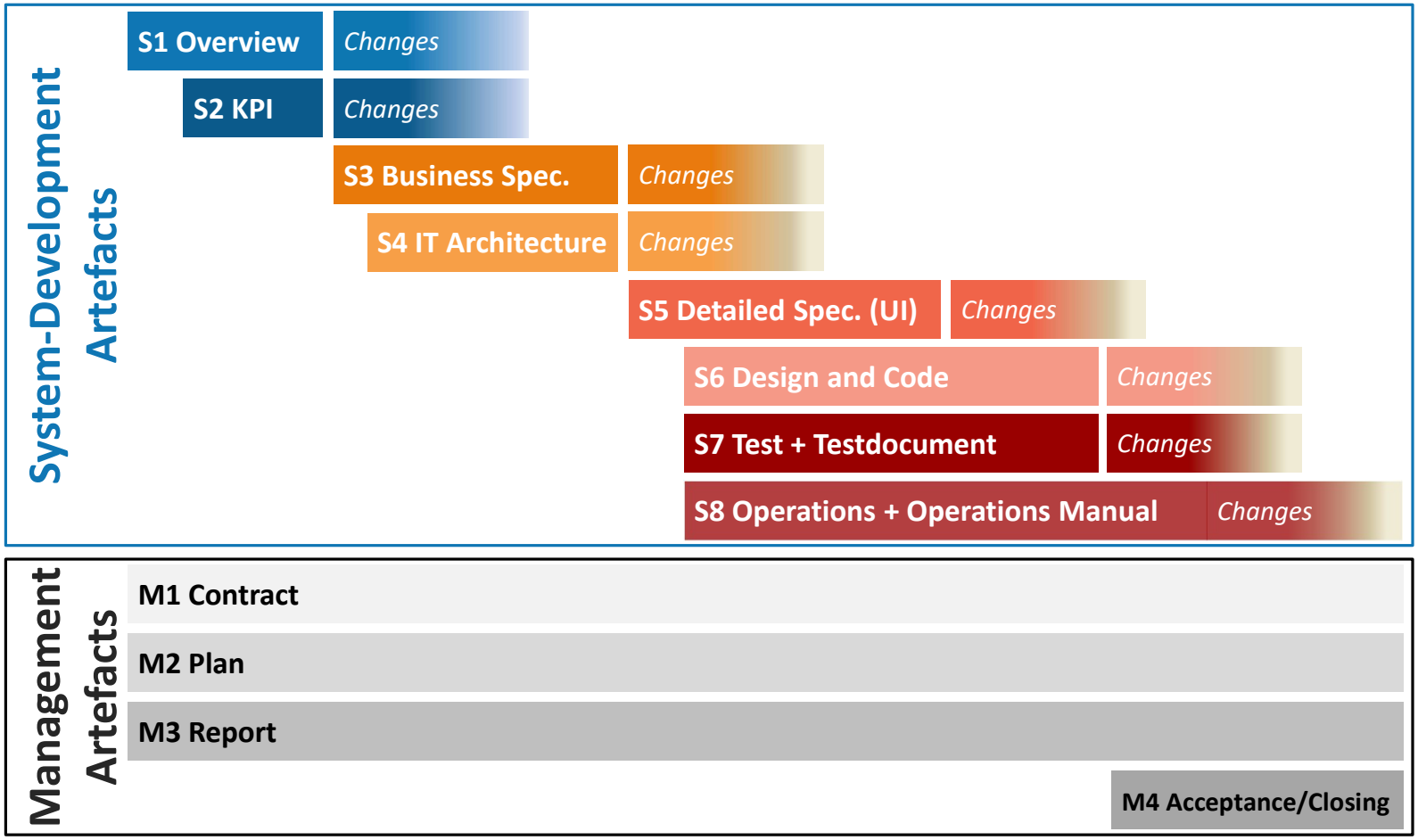

:= (Project Records, Q, count hours/elapsed days)

- **1 Person Day (PD)** = 8 Person Hours
  1 Person Month = 20 Person Days
  1 Person Year = 10 Person Months = 200 PD
- **1 Calendar Day (CD)**
  1 Calendar Month = 20 Calendar (Work) Days
  1 Calendar Year = 10 Calendar Months = 200 CD

But of what work?

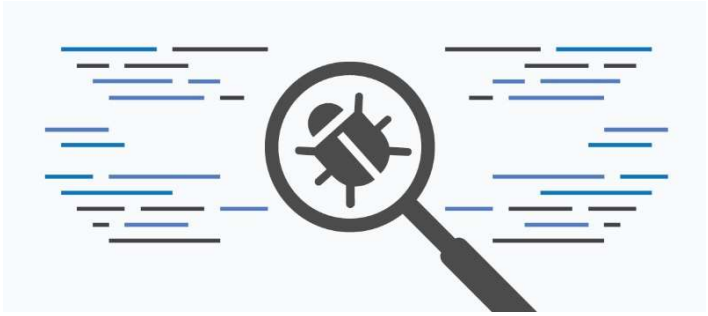
# Normed Software Work Breakdown Structure – Effort to create …

SolutionBoxX



SolutionBoxX

# Defects, Defect Density, Defect Fixing Effort Ratio



:= (Project Records, Q , count production defects/size)

Naturally, bigger systems have a higher probability for more defects, therefore:

- Defect density := #Defects/LOC ??
- **Defect density** := #Defects/FP (or SM) ??

- **Defect Fixing Effort Ratio** :=

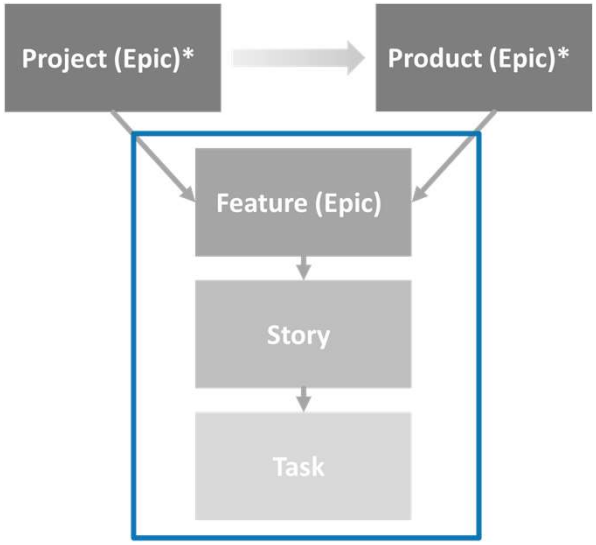
  (Production) Defect Fixing Effort (of a period, e.g. 1 year)

  /

  Development Effort

SolutionBoxX

# Story Points




:= (Story (and Tasks), $\mathbb{N}$, team-expected difficulty/complexity)

- **1 Story Point** equals n (team-defined) ***estimated*** **Person Hours**
- Assigned at task level, then summed up.
- This assigned difficulty/complexity of a story remains
  - If the effective person hours are more
  - If the effective person hours are less
  - Or the team hit it exactly
- For a **set of stories** for a team in a sprint, it is a goal that the sum of estimated person hours equals the effective person hours! Over und under estimates shall balance out.
- Often rated/estimated at **Fibonacci** sequence.

SolutionBoxX

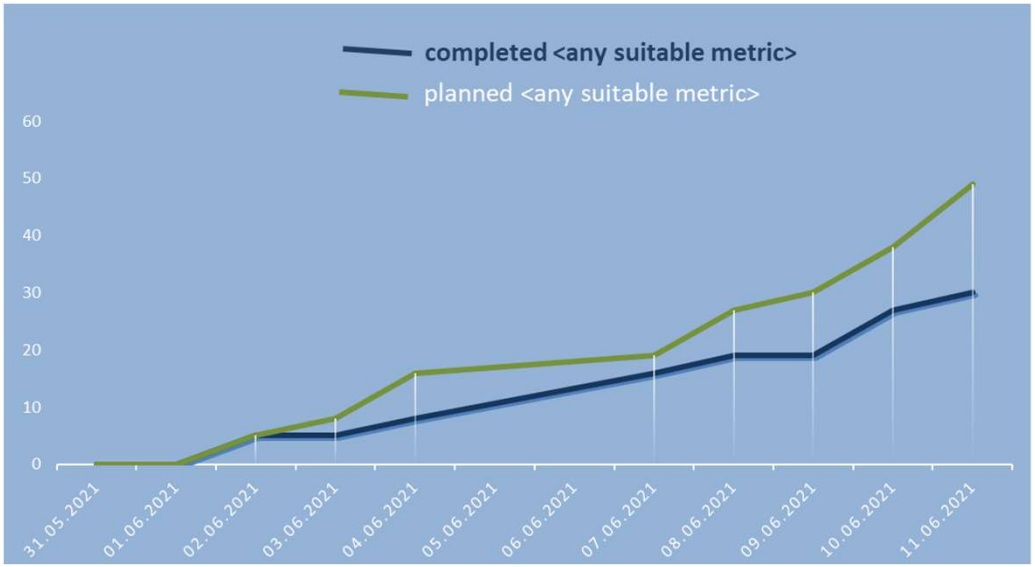
# Business Value (and risk reduction/opportunities, time criticality)

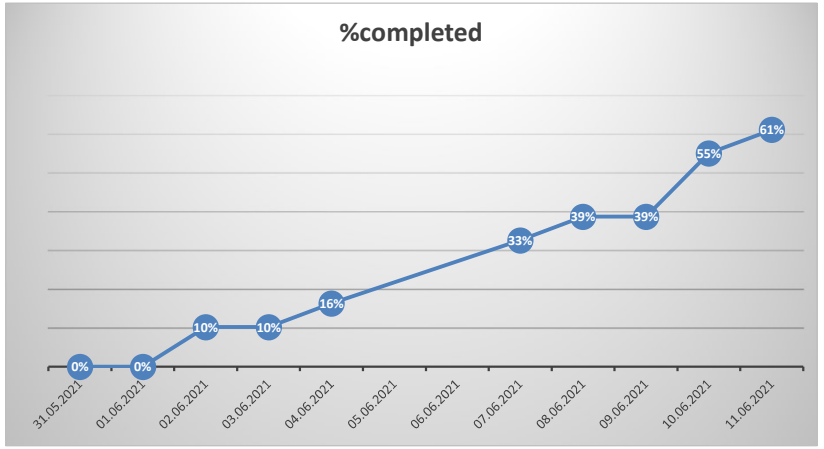
:= (Story/KPI forecasts, ℚ, sum of monetary benefits)

- The net benefit a business receives per period from *operating* a software system
- Apart from monetary benefits, also
  - Regulatory obligations
  - Reputation or customer satisfaction benefits
  - Etc.
- May increase/decrease over time ➔ **Time Criticality**
- Story may also **reduce risks** and/or open **opportunities** (‘enabler’)

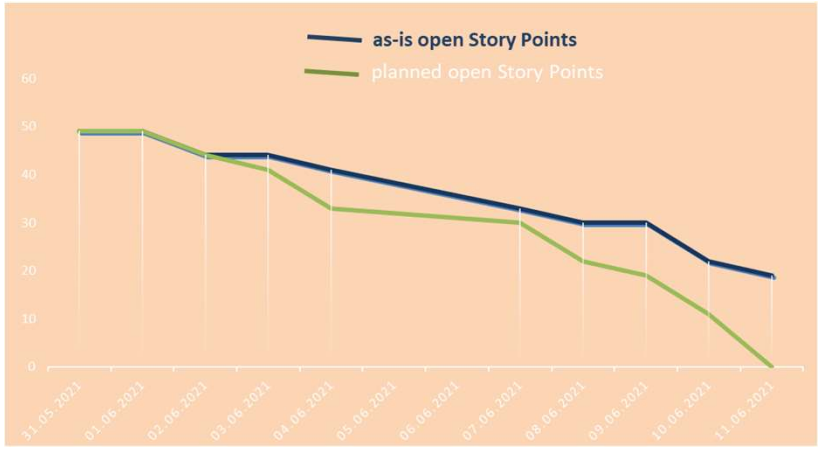
SolutionBoxX

# %completed (or Burndown chart)



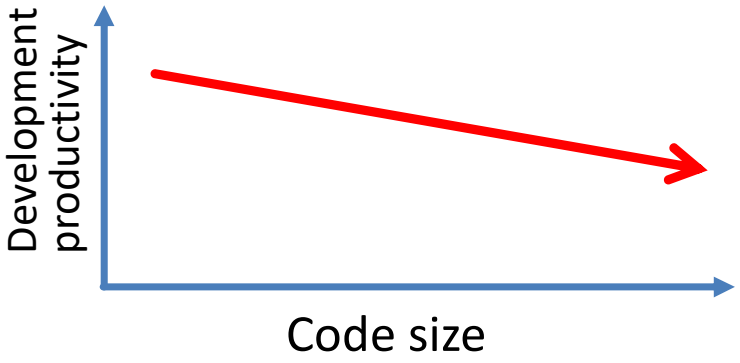




Same displayed as Burndown Chart



SolutionBoxX

# Development Productivity




NOT: «Code size» per Effort unit

BUT: «Functionality» per Effort unit

Functionality = size of bug-free implemented specifications

Observation: Increasing code size means decreasing development productivity

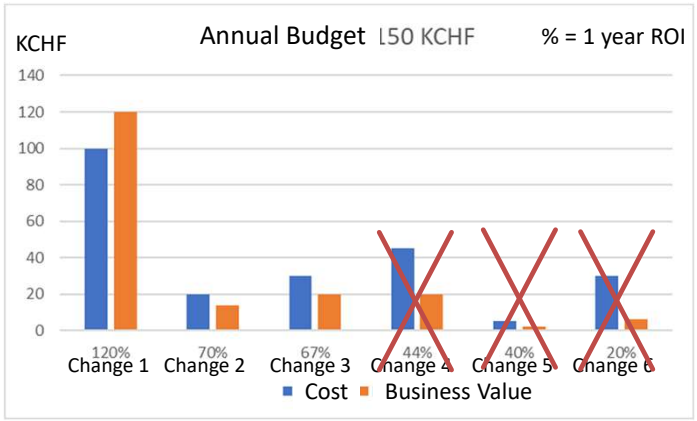
# Topics

SolutionBoxX

1. Why measure «things» in Software Engineering?
2. Software Metrics Basics
3. Software Product Metrics – Measuring Code
4. Software Product Metrics – Measuring Specifications
5. Software Process Metrics
6. **Estimation (of Project Effort and Duration )**

SolutionBoxX

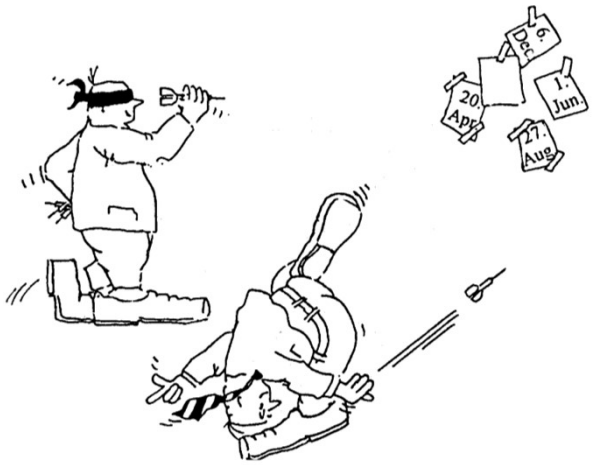
# Why is the Estimation of Effort and Duration important?




- Drives Project Expectations
  - of the customers
  - of the team and management
- Defines Success
- Needed for Planning (Team Sizing)
- Needed for Prioritization
- Shapes the Reputation of IT Professionals

SolutionBoxX

# Estimation Challenges



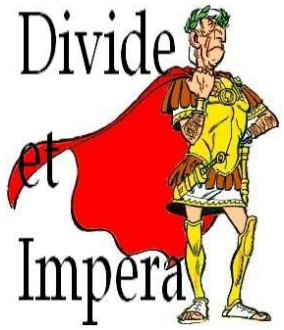

- Not relevant – the customer pays anyway.
- Managers expect low estimates.
- No time to analyse/model.
- No past records.
- No estimation expertise.

„Forecasts are difficult, especially when they concern the future“

attributed to Niels Bohr

SolutionBoxX

# The pure expert ‘divide and impera’ approach




Is the task to estimate „complex“?

YES:

1. **Divide** the task into sub-tasks
2. Apply this approach for the sub-tasks
3. Estimate = **Sum of the estimates of the sub-tasks**

NEIN: Directly rate/estimate the task.

Aka **Bottom-Up Approach**

**Advantges:**

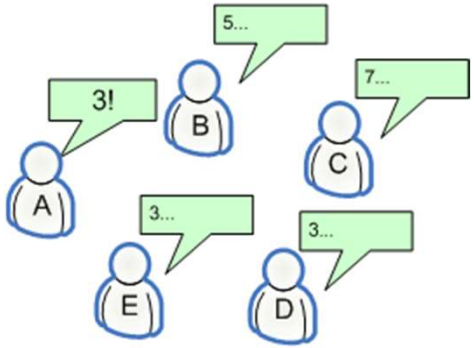
- efficient

**Drawbacks:**

- random quality
- depending on the expert
- exposed to the risk of fulfilling low estimates expectations

SolutionBoxX

# The Delphi approach




Many experts:

(A) Discuss/understand the task to estimate

(B) Do simultaneous estimation

(C) On divergence:
- challene/defend the estimates
- then go back to (B) or (A)

(D) Else: Consensus on 1 estimate (or 'cannot estimate')

Aka **Wideband Delphi**

Original Delphi: without step (A)

**Advantges:**
- promotes discussion and understanding
- only agreed values are used

**Drawbacks:**
- needs n equal experts
- difficult to reproduce
- democracy on 'scientific' issues can fail badly
- 'the team agreed'

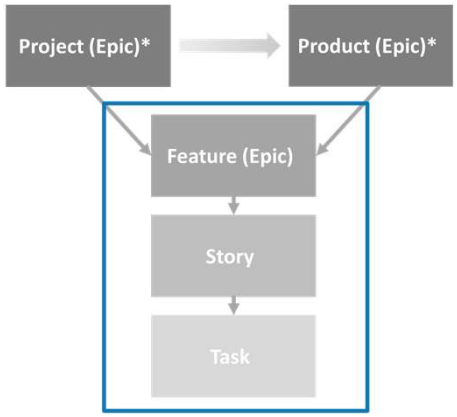
SolutionBoxX

# The Delphi approach – WSJF Prioritization as an example

**WSJF := Weighted Shortest Job First**

**:= (BV + RO + TC) / Complexity**

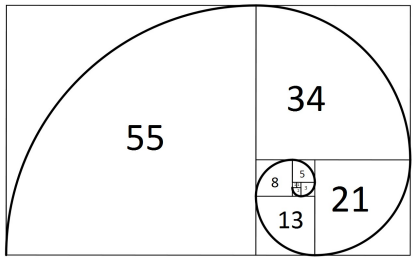
**The resulting WSJF values are used to prioritize business features.**




**4 Parameters of a Feature are estimated using 'Planning Poker' on Fibonacci scale:**

1. **BV = Business Value**
2. **RO = Risk Reduction and/or Opportunities**
3. **TC = Time Criticality**
4. **Complexity (Story Points)**

Leonardo Fibonacci (1170 – ca. 1240) in Pisa



SolutionBoxX

# The Delphi approach – WSJF as an example

**WSJF := Weighted Shortest Job First**

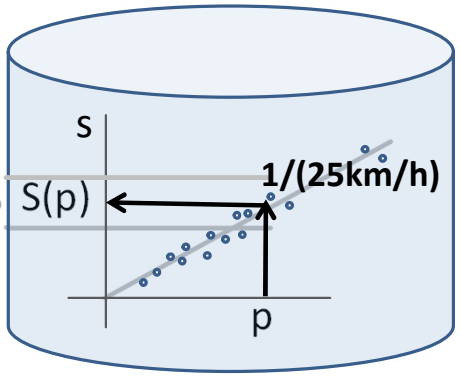
**:= (BV + RO + TC) / Complexity**

Sample Feature:
“An association has online registration forms for new members and event participations that send mail to the secretary (a board member). The secretary then manually enters the data into lists (Excel).
A script shall be programmed that catches those mails (by subject), reads the data and adds it to the Excels.”

LINK to Voting Poker

SolutionBoxX

# The parametric approach




**S(p) = 1/25 x p**
(linear)
dS = +/-20% (+/-2σ)

**A Preparation: Get the relevant facts**

E.g. buy a map of Switzerland
and draw the planned bicycle tour


**B** measure **the predictor metric p**

E.g. the tour distance: **150 km**

**C** The **estimated metric** is then caclulated using an estimation function **S(p)** with uncertainty **dS**

E.g. know from experience (**Empirical Database**)
**the velocity (and uncertainty)**
and apply the estimation function

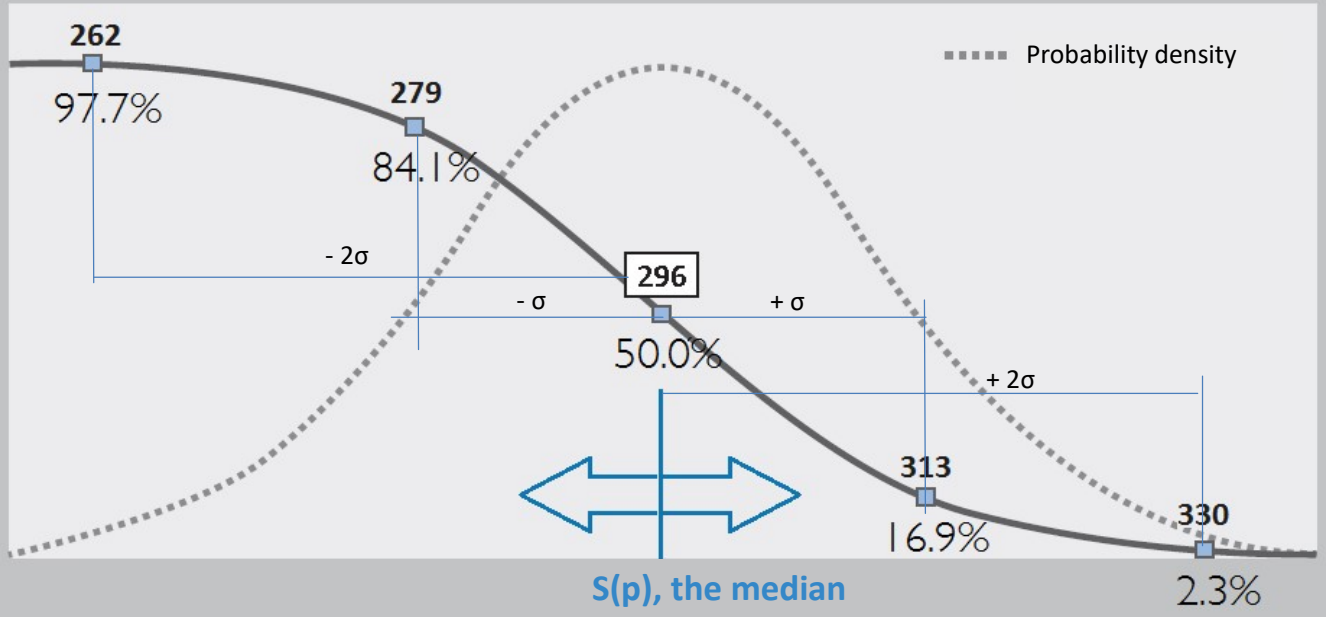
SolutionBoxX

# Interpreting the uncertainty dS (parametric approach)

Example (without units):

**S(p) = 296**

**$dS_{relative}$ = +/-12%** ➔ **$dS_{absolute}$ = +/-34**




% = risk of underestimation when using value for planning

SolutionBoxX

# **COCOMO** (a parametric approach)

## **Co**nstructive **Co**st **Mo**del

Developed by: Barry Boehm (Boeing), 1979, https://en.wikipedia.org/wiki/COCOMO

**Estimation Function (COBOL):**

**$E = 48.0 \times KLOC^{1.05}$**

**dS = +/-??%**

**Example:**

**KLOC = 20**

**$E_{D/C}$ = 48.0 x 23.23 = 1'115 PD**

*Fact:* **The Lines of Code to be programmed are *estimated (!!)***

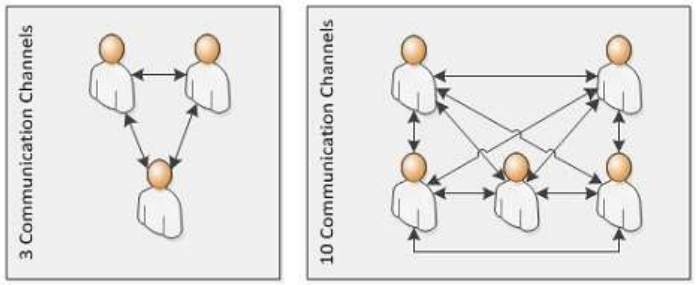
*predictor metric:* **KLOC**

*estimated metric:* **Effort (in Person Days) $E_{D/C}$ to create Design and Code**

notes:
- the complete model is more complex
- if used on measured KLOC for estimating technology porting effort, then divide **$E_{D/C}$** by 4.
- there exist conversion factors for KLOC of other programming languages
- **not simply linear !!**

SolutionBoxX

# Brooke's Law – Mythical Man Month




"Adding manpower to a late software project makes it later."

attributed to Fred Brooks, 1975 (book The Mythical Man-Month)

SolutionBoxX

# **Function Point Method** (a parametric approach)

**Estimation Function:**

**E = 0.655 x FP + 0.02799 x $FP^2$**

**dS = +/-20.3%**

**Example:**

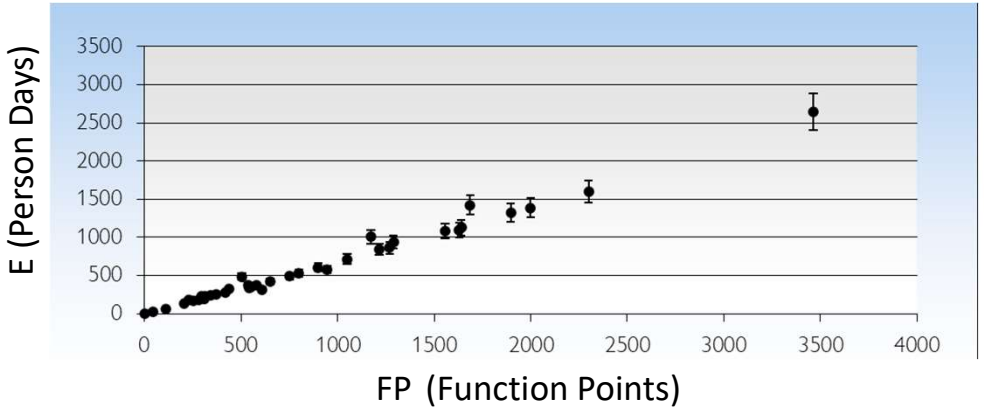
**FP = 200**

**E = 132 Person Days +/- 27**

*Fact:* **A specified system according to FP modelling**

*predictor metric:* **FP (Function Points)**

*estimated metric:* **Effort E (in Person Days) to create the normed deliverables**



SolutionBoxX

# **System Meter Method** (a parametric approach)

**Estimation Function:**

**E = 1.67 x PRE-SM + 0.0008 x PRE-SM²**

**dS = +/-34.0%**

**E = 0.3818 x DOME-SM + 0.00009 x DOME-SM²**

**dS = +/-10.2%**

**Example:**

**PRE-SM = 36**

**E = 61 Person Days +/- 21**

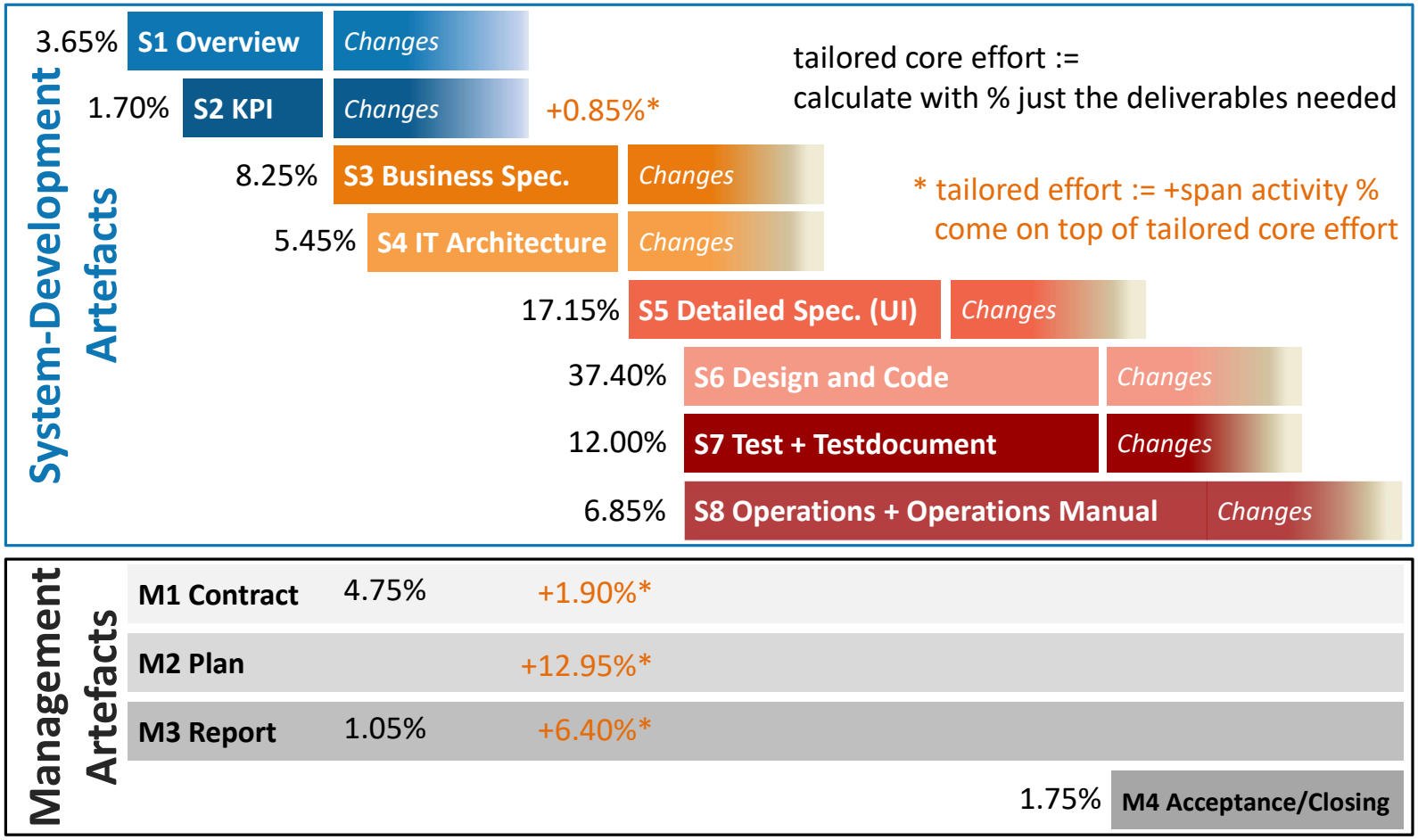
*Fact:* **A specified system according to Overview modelling**
**or to Business Specification**

*predictor metric:* **PRE-SM (PRE System Meter)**
**or DOME System Meter**

*estimated metric:* **Effort E (in Person Days) to create the normed deliverables**

# Effort per Deliverable (a parametric approach, linear % of total effort)

SolutionBoxX



SolutionBoxX

# Estimating Duration (a parametric approach)

**Estimation Function:**

$D = -185 + \sqrt{(34'339 + E_t \times 231)}$

**dS = +/-40%**

**Example:**

**$E_t$ = 600 Person Days**

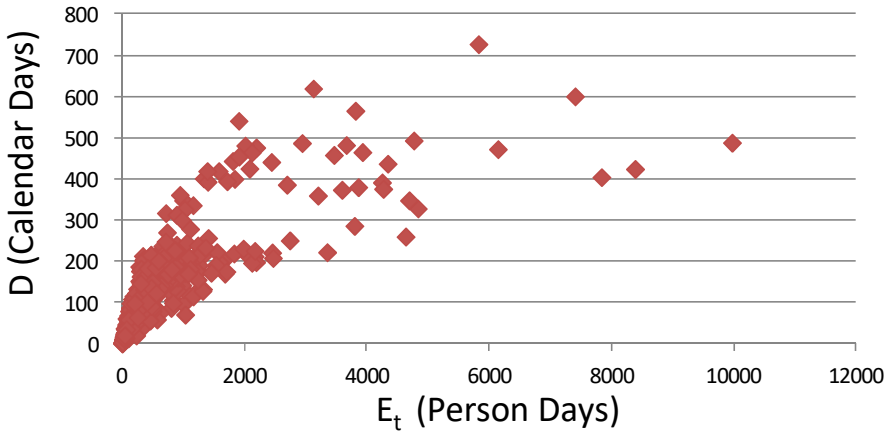
**D = 231 Calendar Days +/- 92**

consequence:
For a given project effort, there is an **ideal team size**.
**Ca. 600 / 200 = 3 Persons** in the example.

*Fact:* **The tailored total effort $E_t$ is estimated**

*predictor metric:* **$E_t$ (in Person Days)**

*estimated metric:* **Duration D (in Calendar Days)**



SolutionBoxX

# The Effect of Project Acceleration (a parametric approach)

**Estimation function:**

$m = 0.21 \times a + 0.792 \times a^2$

dS = +/-52%

**Example:**

a = 1.5 (3 years / 2 years)

m ≈ 2.1 (5'500 PD / 2'500 PD)

*Fact:* **A project acceleration is given**
**acceleration a := estimated duration / required duration**

*predictor metric:* **acceleration a**

*estimated metric:* **effort mutliplicator m**
**m := effort of accelerated project / estimated effort**

**Rule of thumb 1:**
**The estimated duration shall be reduced by -20% from the parametrically estimated duration because it has a large uncertainty (of +/-40%).**

**Rule of thumb 2:**
**Accelerations above 1.6 must be refused as impossible (so called 'death march' projects)!**

# What You should know

SolutionBoxX

1. Why measure «things» in Software Engineering?
   a. Assess (forecast) quality
   b. Estimate project effort and duration
2. Software Metrics Basics
   a. A classification scheme
   b. Basics about measurements and scales
3. Software Product Metrics – Measuring Code
   a. Lines of Code (LOC)
   b. Halstead's Volume Metric
   c. McCabe's Cyclomatic Complexity
   d. Class Coupling
   e. LCOM – Lack of Cohesion of Methods
4. Software Product Metrics – Measuring Specifications
   a. Function Points (FP)
   b. System Meter
5. Software Process Metrics
   a. Effort (Cost) and Duration of «What?»
   b. Number of Defects – Defect Density – Effort to remove defects
   c. Business Value
   d. Story Points
   e. Earned Value – Completion% - Burndown
   f. Software Development Productivity
6. Estimation (of Project Effort and Duration )
   a. Why is it important?
   b. Challenges
   c. The pure expert 'divide et impera' approach
   d. The Delphi approach – Example WSJF/Voting Poker
   e. The parametric approach
      - Example: COCOMO
      - Example: Function Point Method
      - Example: System Meter Method
      - Example: Phase% calculations
      - Example: The Mythical Man Month - project acceleration

# Recommended Reading (optional) …

SolutionBoxX

1986. Controlling Software Projects: Management, Measurement, and Estimates. Tom **DeMarco**. Prentice Hall, ISBN 0-13-171711-1

2004. Aufwandschätzung von IT-Projekten. Manfred **Bundschuh**, Axel **Fabry**. Verlag MITP-Verlag ISBN 382660864X

1977. Elements of Software Science. **Halstead**, Maurice H. Amsterdam: Elsevier North-Holland, Inc. ISBN 0-444-00205-7.