Seminar on Software Cost Estimation
WS 2002/2003

# Manual Techniques, Rules of Thumb

Pascal Ziegler

## Introduction

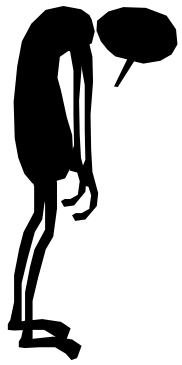
- good software measurement and estimation are important
- simple methods are widely used
- simple, but not very accurate
- can be calculated mentally or with a pocket calculator

**Where manual estimation techniques are useful:**

- Early estimates before requirements are known
- Small projects needing only one or more programmers
- Low-value projects with no critical business impacts

**Where manual estimation techniques are NOT useful :**

- Contract purpose for software development or maintenance
- Projects larger than 100 function points or 10'000 source code statements
- Projects with significant business impact



## Content

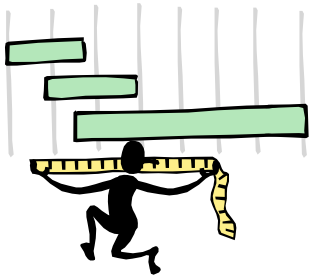
Function Point Sizing Rules of Thumb

[Boehm81] Boehm, B. (1981). *Software Engineering Economics*. Englewood Cliffs, N.J.: Prentice Hall.

Other manual techniques (B. Boehm)

[Jones98] Jones, T.C. (1998). *Estimating Software Costs*. New York : McGraw-Hill.

# Design Goals of FP



Productivity

Quality

Discussions with Clients

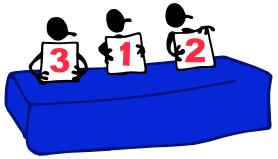
In Any Known
Programming Language

Software Contracts

In Any Combination
of Language

Large-scale Statistical Analysis

Value Analysis




All Class of Software

*see [Jones98], p182*

# Function Point Sizing Rules of Thumb



## Sizing function point totals prior completion of requirements

- FP cannot be calculated accurately until the requirements analysis is terminated
- Method for estimating a rough approximation of FP total
- Three kind of factors: Scope, Class, Type
- A rough sizing method:

Three Steps:

- Apply the numeric list values to the project to be sized in terms of the scope, class, and type factors.
- Sum the numeric values from the three lists.
- Raise the total to the 2.35 power.

# Function Point Sizing Rules of Thumb



## Examples:

**Client/server application:**

| **Step 1** | **Step 2** | **Step 3** |
|---|---|---|
| Scope = 6 (standalone program)<br>Class = 4 (internal-single site)<br>Type = 8 (client/server) | Sum = 18 | $18^{2.35} = 891$ |

**Personal application:**

| **Step 1** | **Step 2** | **Step 3** |
|---|---|---|
| Scope = 4 (disposable prototype)<br>Class = 1 (individual software)<br>Type = 1 (nonprocedural) | Sum = 6 | $6^{2.35} = 67$ |

# Function Point Sizing Rules of Thumb

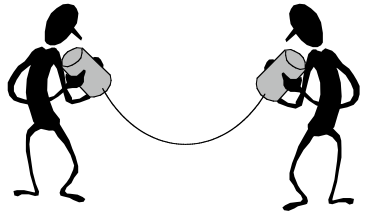


## Estimation Methods derived from Function Points

- different metrics based on function points
- Capers Jones describes 12 rules

# Function Point Sizing Rules of Thumb



LOC  FP

**Rule 1 - Sizing source code volumes:**
One function point = 320 statements for basic assembly language
One function point = 213 statements for macro assembly language
One function point = 128 statements for the C programming language
One function point = 107 statements for the COBOL language
One function point = 107 statements for the FORTRAN language
One function point = 80 statements for the PL/I language
One function point = 71 statements for the ADA 83 language
One function point = 53 statements for the C++ language
One function point = 15 statements for the Smalltalk language

*Programming style and programming language can vary the results significantly!*

# Function Point Sizing Rules of Thumb



- Software development is very paper intensive.
- For large systems: The documentation costs more than the coding.

**Rule 2 - Sizing Software Plans, Specifications, and Manuals:**
Function points raised to the 1.15 power predict approximate page counts for paper documents associated with software projects.

*"For a few really large systems in the 100'000-function point range, the specifications can actually exceed the lifetime reading speed of a single person, and could not be finished even by reading 8 hours a day for a entire career!"* *[Jones98], p192*

# Function Point Sizing Rules of Thumb

**Creeping User Requirements:**

- serious problem
- additional expense

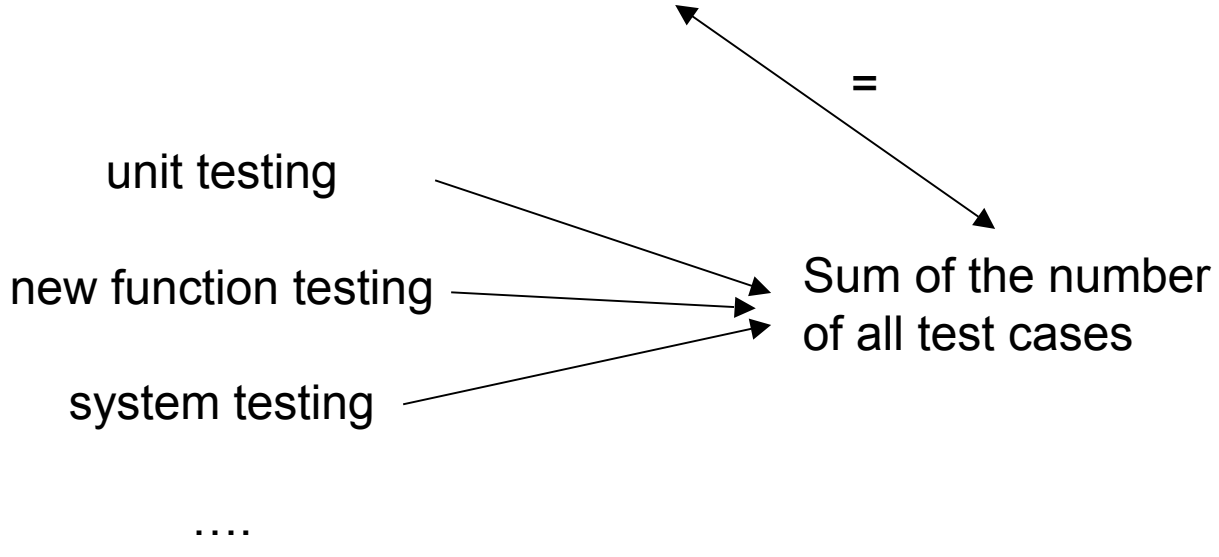
***Rule 3 - Sizing Creeping User Requirements:***
*Creeping user requirements will grow at an average rate of 2 percent per month from the design through coding phases.*

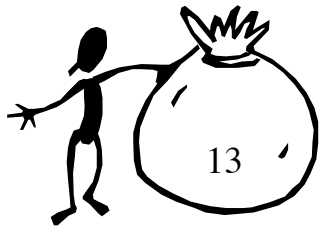
- to avoid disagreement => specify in contract
- time-dependent => the later the changes, the bigger the costs

# Function Point Sizing Rules of Thumb



***Rule 4 - Sizing Test-Case Volumes:***
*Function points raised to the 1.2 power predict the approximate number of test cases created.*



# Function Point Sizing Rules of Thumb



**Major kinds of error:**

1. Requirements errors
2. Design errors
3. Coding errors
4. User documentation errors
5. Bad fixes, or secondary errors introduced in the act of fixing a prior error

**Rule 5 - Sizing Software Defect Potentials:**
Function points raised to the 1.25 power predict the approximate defect potential for new software projects.

***Example:***

- *personal application: 70 FP*
- *$70^{1.25}$ = about 200 bugs*

# Function Point Sizing Rules of Thumb



***Rule 6 - Sizing Testing Defect-Removal Efficiency:***
*Each software test step will find and remove 30 percent of the bugs that are present.*

***Example:***

| Step | Bugs | 30% of the Bugs |
|---|---|---|
| 1 | 200 | 60 |
| 2 | 140 | 42 |
| 3 | 98 | 29 |
| 4 | 69 | 21 |
| 5 | 48 | 14 |
| 6 | 34 | 10 |
| 7 | 24 | 7 |
| 8 | 16 | 5 |
| 9 | 12 | 3 |
| 10 | 8 | 2 |

**=> low efficiency**

# Function Point Sizing Rules of Thumb



**Formal Inspection:**

- higher efficiency
- not cheap
- best ROI

***Rule 7 - Sizing Formal Inspection Defect Removal Efficiency:***
*Each formal design inspection will find and remove 65 percent of the bugs present.*
*Each formal code inspection will find and remove 60 percent of the bugs present.*

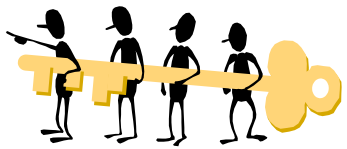
# Function Point Sizing Rules of Thumb



***Rule 8 - Postrelease Defect-Repair Rates:***
*Maintenance programmers can repair 8 bugs per staff month.*

**Maintenance repair rate:**

- has been around the software industry for more than 30 years
- Good defined process and tools => improve this value



# Function Point Sizing Rules of Thumb



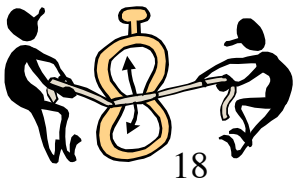
## Rules of Thumb for Schedules, Resources, and Costs

- important topic for clients, project managers, software executives
- just rough approximations!

***Rule 9 - Estimating Software Schedules:***
*Function points raised to the 0.4 power predict the approximate development schedule in calendar months.*

***Example:***
*MS Word = about 5000 FP*

*Rule 9: 5000 FP $^{0.4}$ = about 30 calendar months*



# Function Point Sizing Rules of Thumb

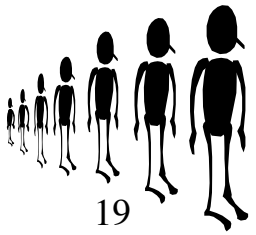


***Rule 10 - Estimating Software Development Staffing Levels:***
*Function points divided by 150 predict the approximate number of personnel required for the application.*

***Example:***
*MS Word = about 5000 FP*

*Rule 9: 5000 FP $^{0.4}$ = about 30 calendar months*
*Rule 10: 5000 FP / 150 = 33,3 full-time personnel*

***Rule 11 - Estimating Software Maintenance Staffing Levels:***
*Function points divided by 750 predict the approximate number of maintenance personnel required to keep the application updated.*



# Function Point Sizing Rules of Thumb



***Rule 12 - Estimating Software Effort:***
*Multiply software development schedules by number of personnel to predict the approximate number of staff months of effort.*

***Example:***
*MS Word = about 5000 FP*

*Rule 9: 5000 FP $^{0.4}$ = about 30 calendar months*
*Rule 10: 5000 FP / 150 = 33,3 full-time personnel*
*Rule 12: 30 months * 33,3 personnel = about 999 staff months*

## Further Manual Software Cost-Estimation Methods

- Expert Judgment: Delphi Technique
- Parkinsonian Estimation
- Price-to-win Estimation
- Top-Down Estimation
- Bottom-Up Estimation

## Expert Judgment

= *one ore more experts hand in an estimation*

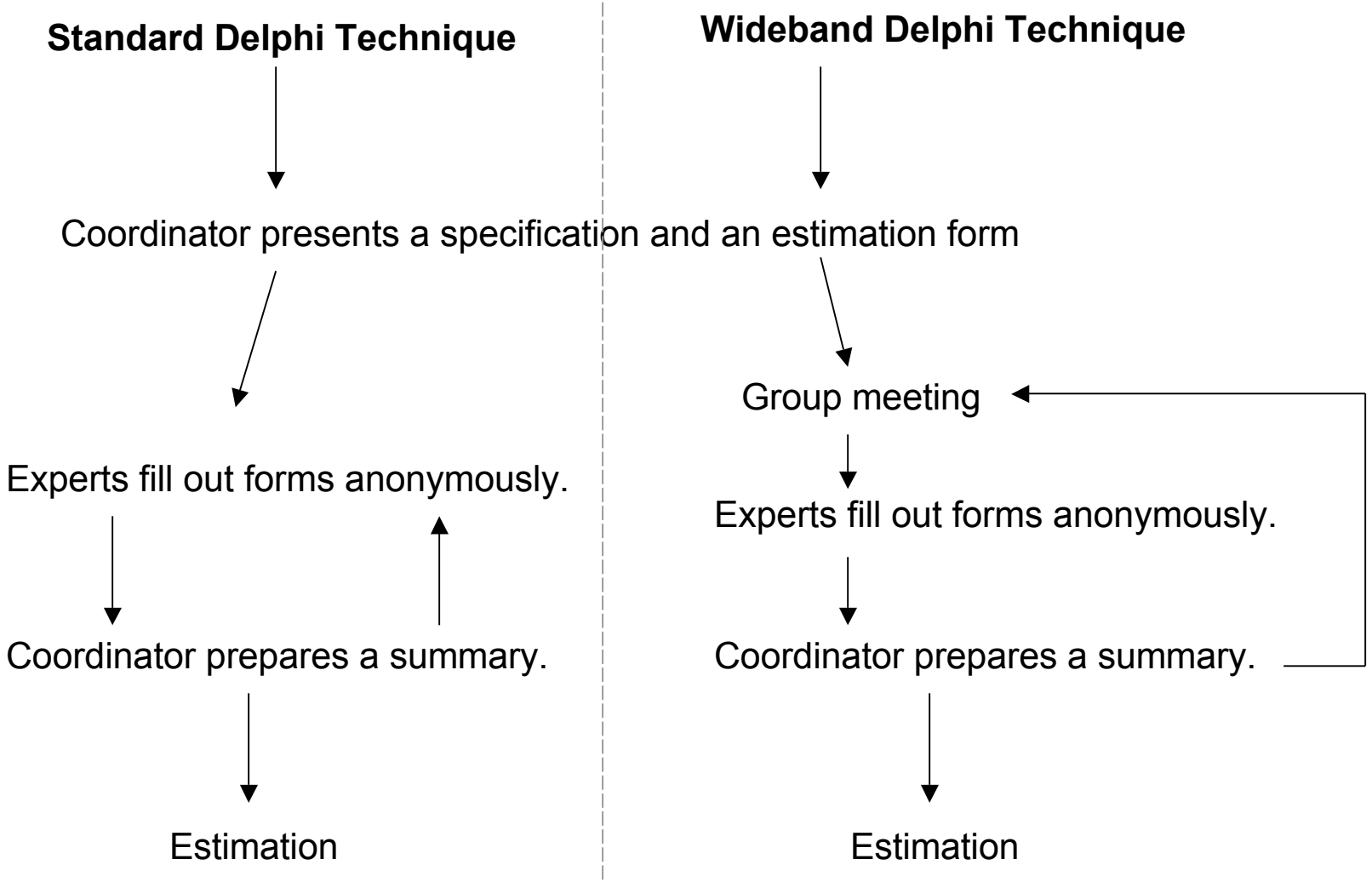
+ Expert is able to factor in the difference between past and future projects.

+ Personal characteristics and interactions

- Depends on the objectivity

# Further Manual Software Cost-Estimation Methods





## Estimation by Analogy

- Compare with other similar projects
- Example: 10000 CHF + 2000 CHF - 1000 CHF = 11000 CHF

\+ based on experience

\- correlation to older projects not clear

# Further Manual Software Cost-Estimation Methods



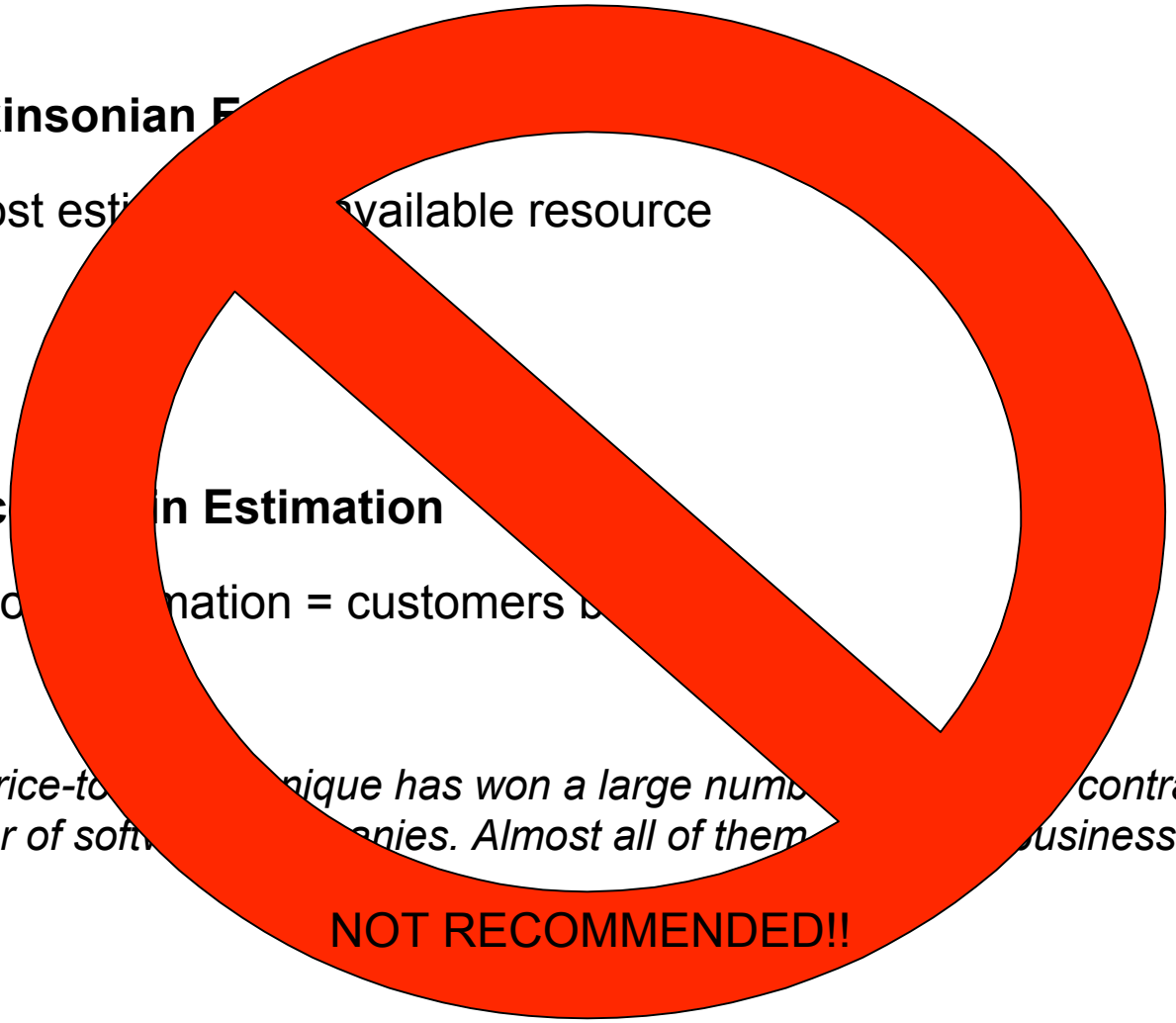
**Parkinsonian E**

Cost esti vailable resource

**Pric in Estimation**

Co mation = customers b

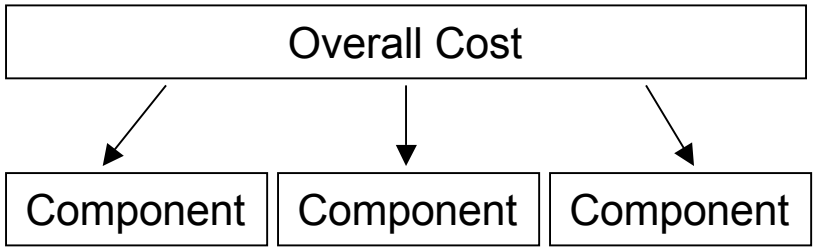
*"The price-to nique has won a large numb contracts for a large number of soft nies. Almost all of them usiness today."*



# Further Manual Software Cost-Estimation Methods



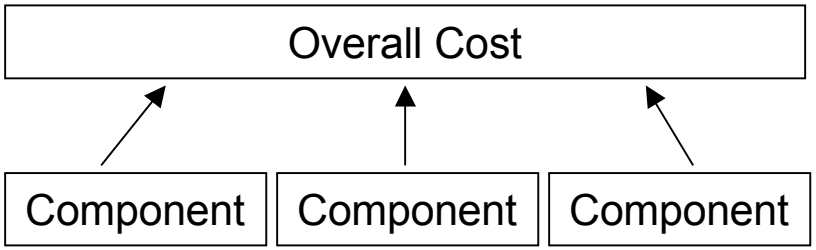
## Top-Down Estimation




+ focus on system level

- difficult to recognize low-level technical problems

## Bottom-Up Estimation




+ component is estimated by the responsible person

- missing system level focus

## Conclusion

- Rules of Thumb are not accurate!
- We have seen different simple tools.

**It is important to question the result of such estimations and to compare it with other values!**