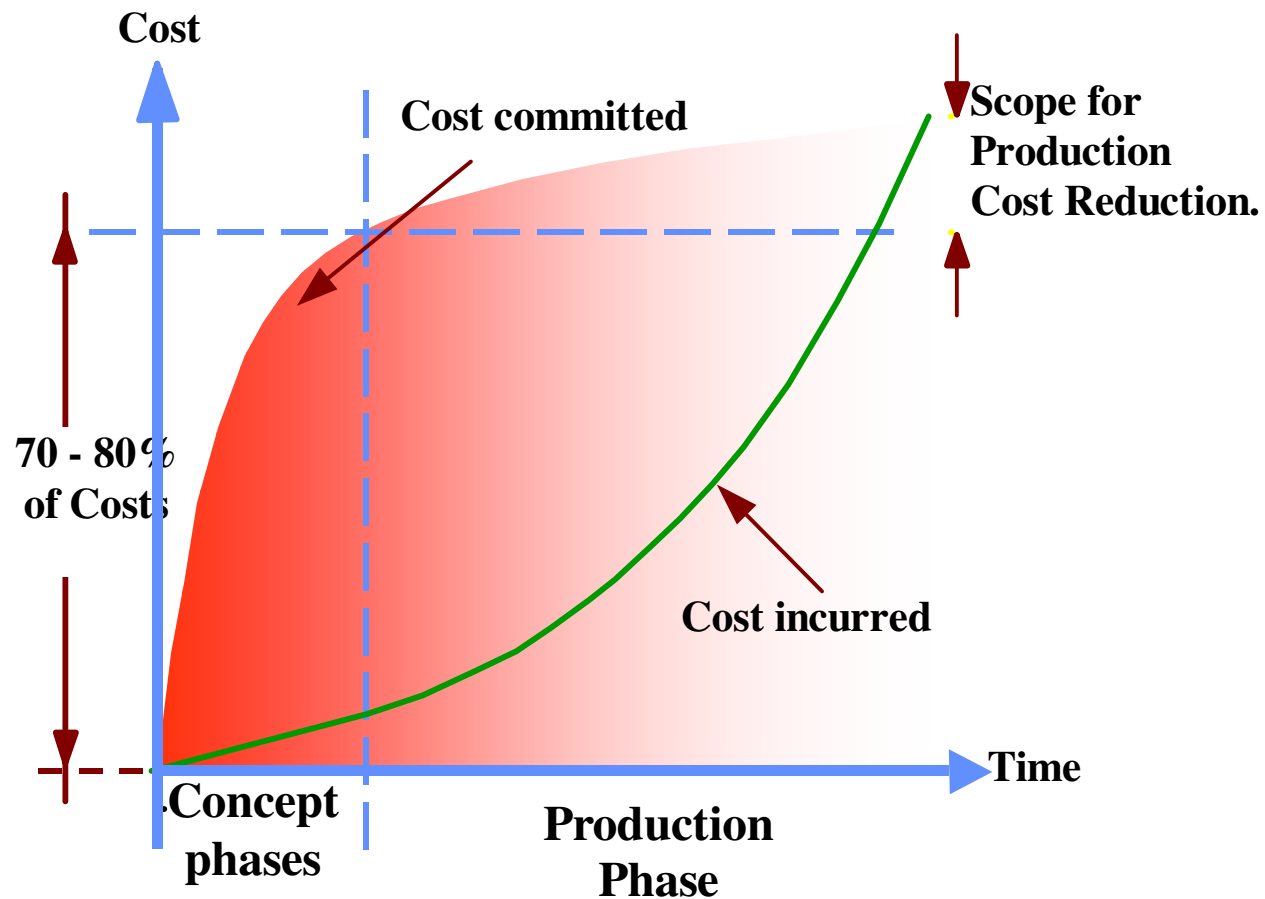
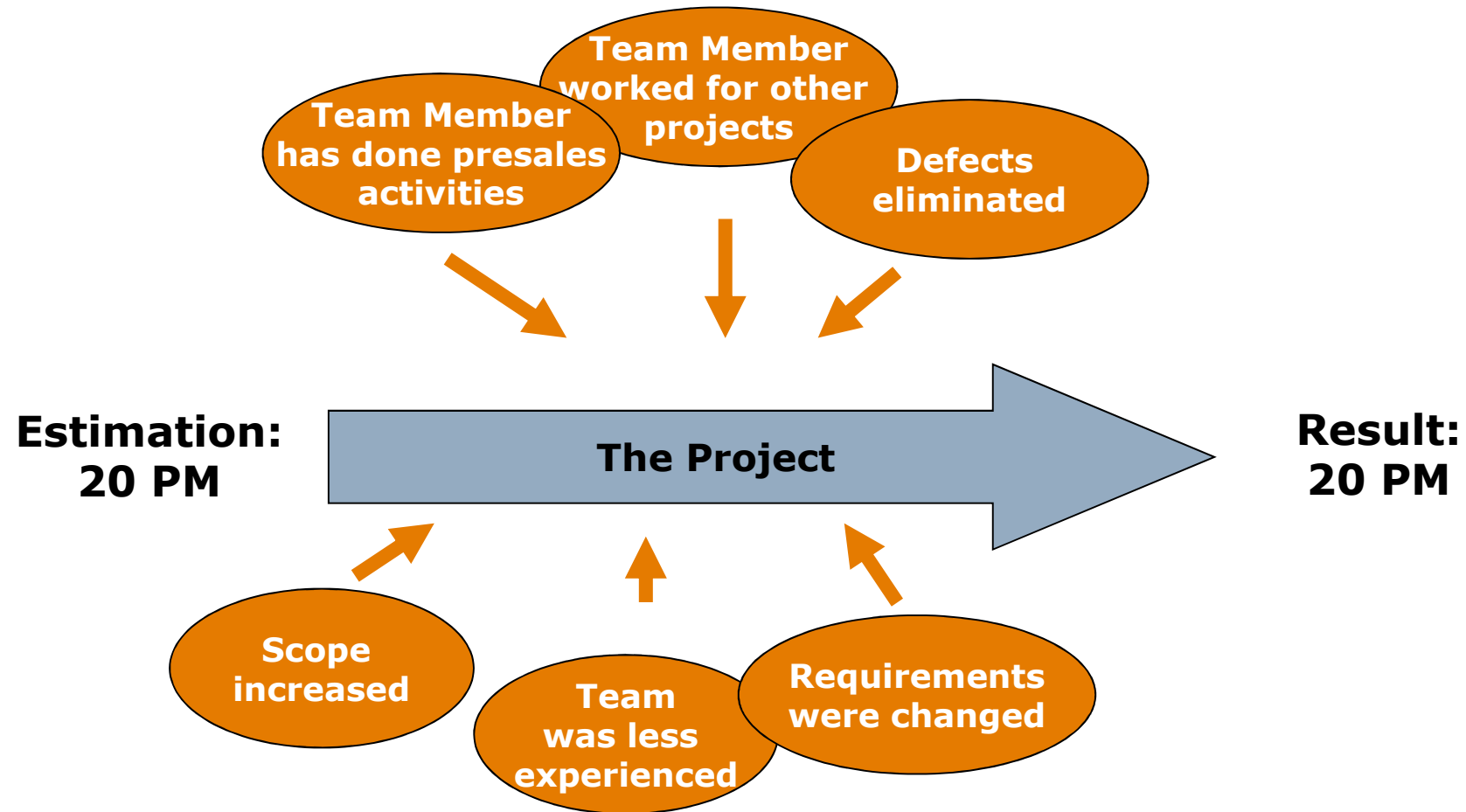
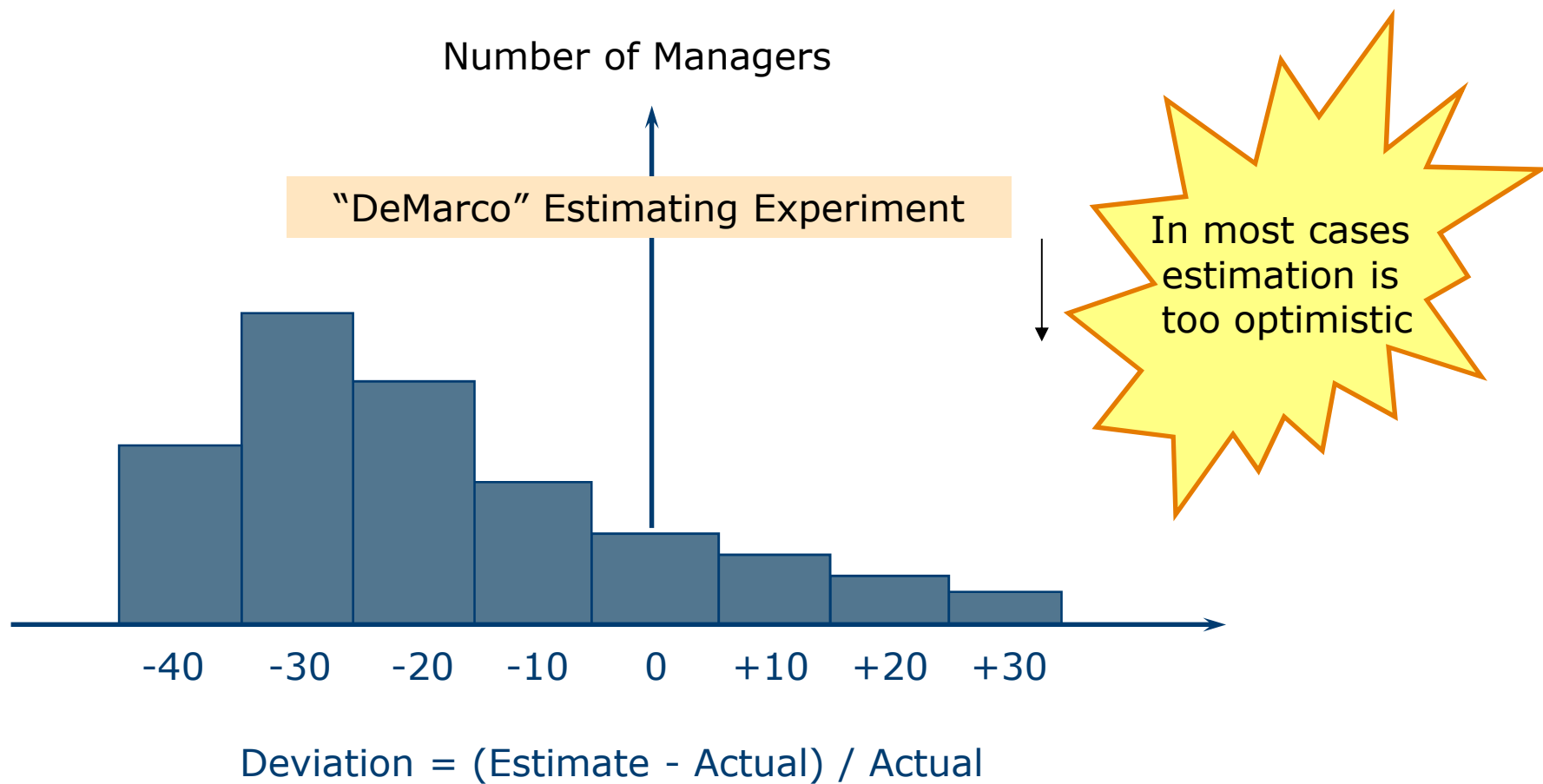


Top 10 Project Impaired Factors (Standish Group)

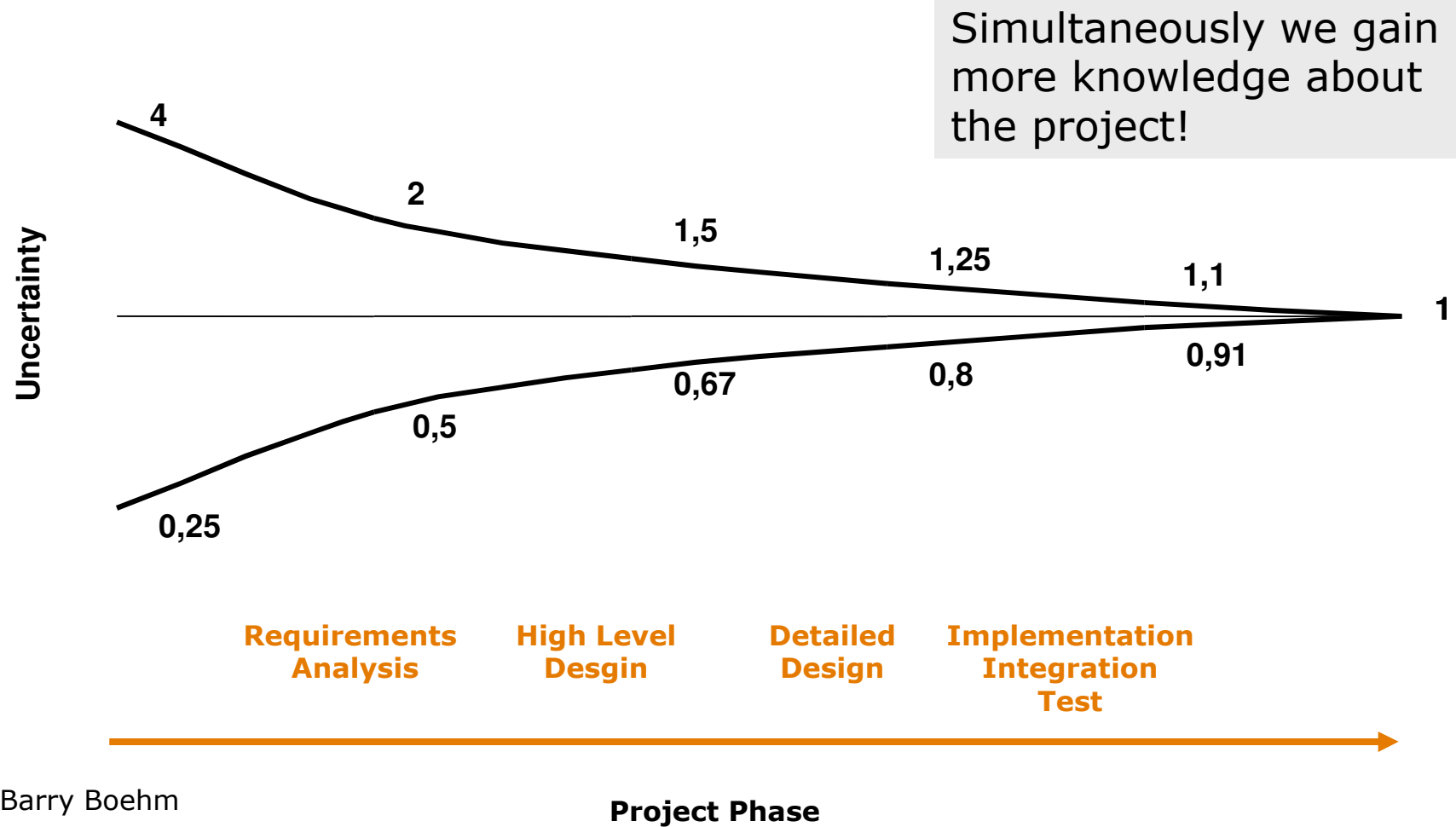
- Incomplete Requirements 13.1%
- Lack of User Involvement 12.4%
- Lack of Resources 10.6%
- Unrealistic Expectations 9.9%
- Lack of Executive Support 9.3%
- Changing Requirements & Specifications 8.7%
- Lack of Planning 8.1%
- Didn't Need It Any Longer 7.5%
- Lack of IT Management 6.2%
- Technology Illiteracy 4.3%





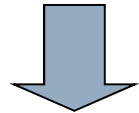


Uncertainty in Estimation

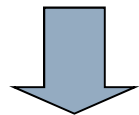


Reducing Risks

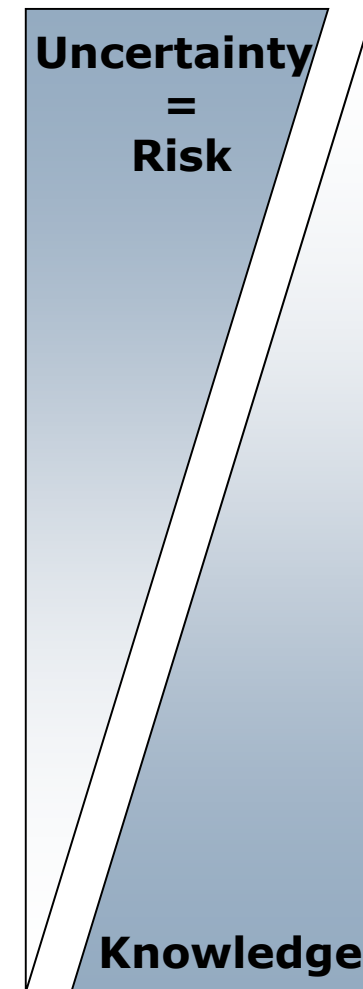
- Fixed Price versus Cost Based Projects
- Learning about requirements, design, etc.



Reducing Uncertainty Conveying Information



Supporting better Decision Making Establishing Trust



Before Project Start

- → Quote
- → Profitability
- → Milestone Planning
- → Resource Planning

Project Start

- → Milestone Planning
- → Detailed Project Plan
- → Resource Planning

During Project Lifecycle

- → Re-Estimation / re-planning for the next iteration(s)
- → Risk Management

Project Closure

- → Final Costing
- → Evaluation of Estimation Process



Estimation means dealing with Uncertainty.

But without any information you can't estimate.

An important goal must be
reducing Uncertainty to reduce risks.

Estimating means also learning

- **Learning about the project**
- **Measuring to improve further estimates**

Estimation is a **recurrent task**.

- It's done during the whole lifecycle, but with different focus.

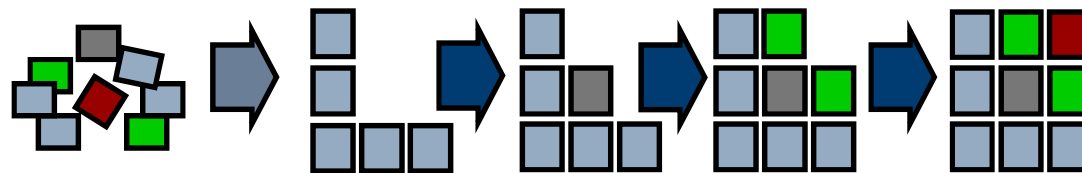
Estimation does not mean Negotiation nor Commitment!



- Intuition, Guessing
- Analogy, Experience
- Company Standards
- Work Breakdown Structures based
- Estimation by Experts & Delphi-Method
- Three-Point-Estimation
- Proxy-based Estimation
- Advanced Methods like Function Point, Cocomo, etc.

Estimation Scope must be defined

- Project Objectives
- Requirements
- Deliverables
- Architectural Diagram
- External Interfaces



What are the project's phases and milestones?

What are the activities which must be performed during a project phase?

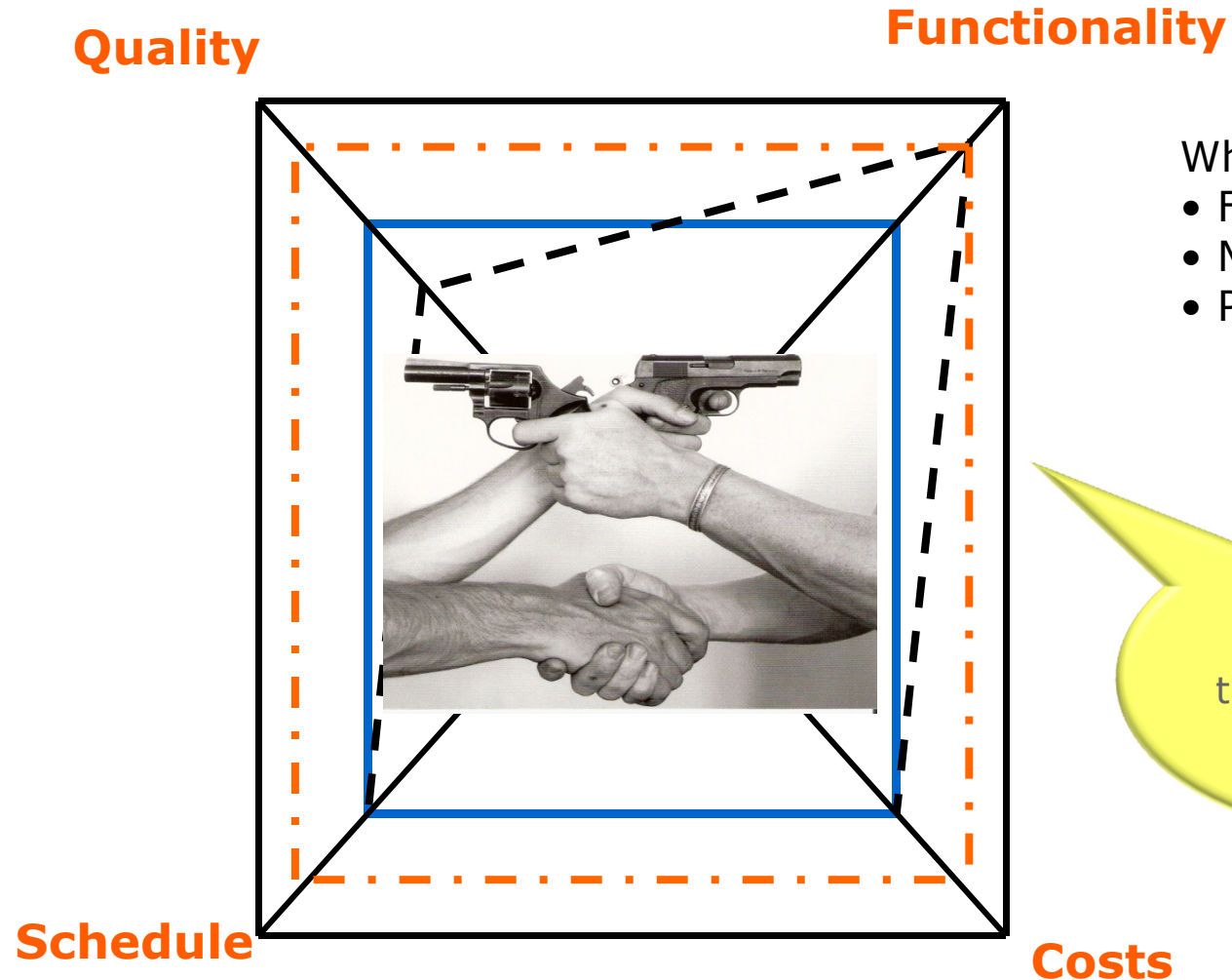
What kind of supporting activities must be performed?

- Configuration Management
- Change Management
- Process Quality Assurance
- Project Management
- Supplier Management

Depending on the Estimation Scope, consider ...

- Project Lifecycle
- Product Lifecycle





What's the input?

- Functionality
- Maximum Budget
- Project End Date

Also represented
as a „conflict
triangle“ of project
management

“Work Breakdown Structure”

- A work breakdown structure (WBS) is a tool used to define and group a project's discrete work elements (in relation to the deliverables) in a way that helps organize and define the **total work** scope of the project.
- The WBS is a **tree structure**, so each lower level contains a more detailed description of a work element.



Typically based upon

- Deliverables
- Lifecycle Model
- Architectural Diagram

The 100% Rule...states that the WBS includes **100% of the work defined by the project scope** and captures all deliverables – internal, external, interim – in terms of the work to be completed, including project management.

The 100% rule is **one of the most important principles** guiding the development, decomposition and evaluation of the WBS.

No overlap in scope definition between two elements of a Work Breakdown Structure.

This ambiguity could result in **duplicated work or mis-communications** about responsibility and authority.

Likewise, such overlap is likely to cause confusion regarding project **cost accounting**.

If the WBS element names are ambiguous, a **WBS dictionary** can help clarify the distinctions between WBS elements.

The WBS Dictionary describes each component of the WBS with

- milestones, deliverables, activities, scope, and sometimes dates, resources, costs, quality.

„Balanced Structuring“ means ...

For each Work Package (lowest level of the WBS)

- **Manageable size**, i.e.
 - Not too small
 - → Effort for managing the WBS!
 - Not too big
 - → Low trust in estimation results
 - → Difficult for Tracking
 - Guidance:
 - Rough Granularity:
should not exceed 4-6 weeks duration or 160 hours effort
 - Detailed Granularity:
few days, but should not exceed 10 days effort

For each Work Package (lowest level of the WBS)

- Must be possible to assign **responsible** persons
- **Independent** to other Work Packages
(Minimizing interfaces)
- Must be possible to **estimate**
(size, effort, duration, cost, etc.)
- **Measurable** (estimating and tracking affected)

During Estimation you will ...

- make assumptions
- find risks

Examples for assumptions

- Unclear requirements, features, etc.
- Availability of resources
- Usage of a certain technology
- Dependencies on suppliers or customer
- And other important influence factors on estimation

Documentation AND Communication necessary!

Validate the Assumptions!

Mitigate the Risks!



Lessons Learned and Improvement is only possible
if the **Estimation Baseline** is known



- Documentation of
 - All input documents for the Estimation Process (Requirements Baseline, etc.)
 - The estimation approach itself
 - Participants of the estimation
 - All output documents of the estimation (e.g. Work Breakdown Structure, Size/Effort/Cost Estimations, Risks, Assumptions, etc.)
 - ... and the actual values!

- Don't rely on only one expert.
(→ review of estimation results)
- Usage of several estimation methods / approaches to verify the estimation results.
- The estimation result should not be a number but an interval.
- The bandwidth of the interval depends on the knowledge of the object to be estimated.
- Round the estimation result. Positions after decimal points simulate an accuracy which does not exist!



Uncertainty in requirements:

- How much does a house cost?
 - → it depends 😊
- Development is a process of a stepwise improvement
 1. Vision of the product
 2. Project objectives
 3. Project Strategy
 4. Product Requirements
 5. Architecture
 6. Low Level Design
 7. Etc.



Chaotic Development Process:

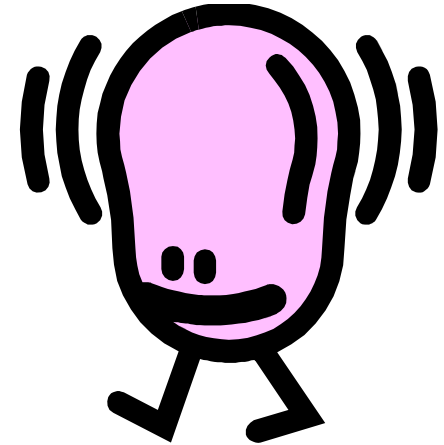
- Unclear requirements
- Missing validation of functional scope with important stakeholders
- “Bad” Design leading to tons of errors in software code
- “Bad” coding style → large effort for bug fixing
- Unexperienced team members
- Incomplete Project Planning
- Underestimation / Overestimation
- “Bad” Configuration Management
- Etc.



- **Don't expect that you can improve your estimation in a chaotic project by using better estimation methods!**
- **Getting rid of the chaos is more important than to improve the estimation!**

Forgotten Activities:

- Most common source of estimation errors!
- 3 categories:
 - Missing requirements
 - Missing activities during development
 - Missing activities outside development
- Examples:
 - Setup / Installation software & Data Migration
 - Help System
 - Interfaces to other systems
 - Training & Coaching for new team members
 - Meetings with Management, department / company wide meetings
 - Build-Process
 - Maintenance of automatic tests (scripts, test cases, etc.)
 - Coordination with suppliers
 - Vacation, public holidays



Unfounded Optimism:

- Don't reduce the estimations done by developers!
In most cases the estimation results are already optimistic!
- Studies proof: Developer often underestimate 20-30% (same is true for Management 😊; see Tom Demarco)
- One reason for this: Nobody wants to be the doomsayer
→ Culture for Risk Management!
- Examples:
 - "Within the current project we are more productive than before."
 - "We had a lot of problems within the last project. This time this will not happen!"



Estimation should be based upon

- Product Structure
- Lifecycle Model
- → Work Breakdown Structure

Documentation of the whole estimation process is necessary

- Traceability of estimation
- Possibility to improve

Document your assumptions and risks identified!

Verify and Validate your estimations!



If possible, count!

If you can't count, try to calculate!

Use your expert opinion only as last chance!

- Try to find a countable indicator for the work to be done.
- The indicator must be correlated with the effort.
- This indicator must be countable early in the lifecycle, not late!
- It should be easy to count! (If you need a high effort to count then it is maybe the wrong indicator.)
- If you want to calculate a statistical average you should have at least 20 representative numbers.
- Understand what you are counting
 - If you compare historical with current data, this data must be comparable!
 - Are the assumptions / constraints the same?



Unit to be counted	Historical data necessary to convert the indicator into an effort estimation
Customer Requirements	<ul style="list-style-type: none">• average effort (hours) to derive system requirements from a given customer requirement• average effort (hours) to implement a customer requirement• average effort (hours) to test a customer requirement• average effort (hours) to document a customer requirement
Features	<ul style="list-style-type: none">• average effort (hours) to implement / test a feature
Use Cases/ Stories	<ul style="list-style-type: none">• average effort (hours) to implement / test a use case• average number of use cases which can be developed within a given time period (e.g. per week)
System Requirements	<ul style="list-style-type: none">• average effort (hours) per system requirement to implement / test / document the system requirement• average number of system requirements which can be reviewed within one hour

Unit to be counted	Historical data necessary to convert the indicator into an effort estimation
Function Points	<ul style="list-style-type: none">• average effort (hours) for development / testing per function point• average number of LOC per function point
Change Requests	<ul style="list-style-type: none">• average effort (hours) to implement / test / document a change request (typically this is categorized in small/ medium/ large change requests)
Defects	<ul style="list-style-type: none">• similar to "Change Requests"
Webpages	<ul style="list-style-type: none">• average effort (hours) per web page to develop/ test
GUI Dialogues	<ul style="list-style-type: none">• similar to "Webpages"
Database Tables	<ul style="list-style-type: none">• average effort (hours) per database table to define / implement sql queries
LOC	<ul style="list-style-type: none">• average number of defects per LOC• average review effort (hours) per LOC
Reports	<ul style="list-style-type: none">• average effort (hours) per report for project reporting

Example:

“If you know that

- there exist 400 defects which must be fixed and
 - your team needed up to now an average of 2 hours to fix one defect
- then you know also that you will need maybe 800 hours for this task”.

The important point is

- No example shown used the expert's intuition.
- The effort was calculated using indicators which were counted.

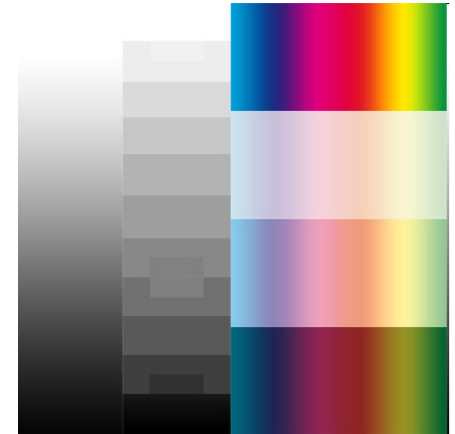
Estimations are more accurate if they are related to “something concrete”.

Compared to the “Expert's Intuition”, estimations based upon historical data are typically more accurate.

Don't adjust an estimation which was counted / calculated only because of an “expert thinks it can be done faster”!

Used to convert counted indicators into estimations:

- LOC → Effort
- # User Stories → # Calendar Weeks
- # Requirements → # Test Cases
- Etc.



Three types of Calibration:

- Industry Data: Data from other organizations/ companies developing similar software products
- Historical Data: Data from the same organization, but from older projects
- Project Data: Data from the same project, but from an earlier phase

- “Historical Data” considers also organizational influence factors.
- Typically organizational influence factors are more important in medium sized / large projects.
- Within very small projects the individual factors have the biggest influence.



Examples for “Organizational Influence Factors”:

- Stable requirements?
- How much documentation?
- Safety relevant environment? Which (A)SIL?
- Culture of (Resource) Management
- Maturity of Development Processes
- Etc.

Size:

- The whole source code or only the source code which is part of the Release (e.g. what about unit-test-code)?
- How do you count re-used components?
- How do you count components from third parties?
- What about empty LOC and comments?

There doesn't exist an Industry Standard.

But no matter what kind of size you will count:

- Define criteria within your organization!
- Build baselines of historic data based upon the criteria!

Otherwise you can't compare the data!



Effort:

- The unit? Hours, Days or something different?
- How many hours per day are you counting?
(8 hours or maybe only the hours you were working for the project)
- How about unpaid overtime?
- Days for vacation, training, etc.?
- Meetings on department level?
- Which kind of effort are you counting? Requirements, Design, Testing, Documentation, Sales Support?
- What about effort which is used for several projects?
- Effort for support of already released products?
- Effort for travelling?

But no matter how you are measuring effort:

→ Define criteria within your organization!

→ Build baselines of historic data based upon the criteria!

Otherwise you can't compare the data!

Duration:

- When does the project start?
(The day of the first project idea? The day of the approved budget?)

→ Capers Jones:
Less than 1% of the projects have a clear project start date.
- When is the project's end?
(Acceptance by the customer? The day the last release candidate is delivered to the test team?)

→ Capers Jones:
15% of the projects have an unclear project end date.

But no matter how you are measuring "duration":

- Define criteria within your organization!
- Build baselines of historic data based upon the criteria!

Otherwise you can't compare the data!

Defects:

- All change requests or only defects? (Who classifies?)
- What about several reports on the same defect?
- Only defects discovered by the customer? Or by the test team? Or by the whole team?
- When do you start counting defects? Before alpha-release? Before beta-release?

But no matter how you are counting defects:

- Define criteria within your organization!
- Build baselines of historic data based upon the criteria!

Otherwise you can't compare the data!

Collect the data as soon as possible after

- the project's end
- phase end

More useful:

Collect the data regularly (e.g. every 2 weeks)

- Re-use of the data within the same project
- Analysis of trends possible
- You learn more about the lifecycle

Hint:

- Use the data of the current project to adjust your estimation for the rest of the project's lifecycle.



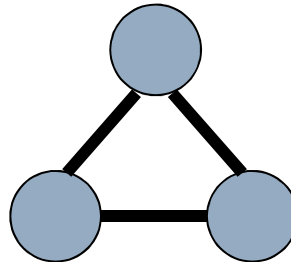
Disadvantage of “Size”

People:

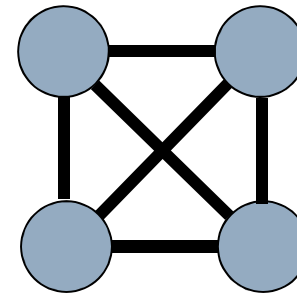
2



3



4



Communication: 1

3

6

$$\# \text{ Communication Pathes} = (n-1)*n / 2$$

→ Exponential Growth of Communication Effort?

Don't just re-use the old measures! There is no linear relationship between project effort and project size.

Experience (Mc Connell):

- If the project to be estimated is not so much different to the projects you have already done, then you can re-use the historical data.
- Typically this means that the smallest and the largest project should not differ in size more than factor 3!

Work to be done should be estimated by the people who will do the work.

Estimations by experts who will not do the work are less accurate.
They tend to under-estimate!

Typically a WBS based approach gives the best results.

“Law of large numbers”:

- Divide large estimations in several smaller units.
- Mistakes about under-estimation and over-estimation will compensate (partially).

Estimation done by a group of experts
Workshop needs a moderator

Steps:

1. Moderator explains the scope, procedure, etc. documents are distributed
2. Group discussion about the scope
3. Experts are estimating separately (using forms)
4. Moderator summarizes the estimated values; deviations are calculated
5. Group discussion about the results and the deviations
6. Each expert updates separately the estimation
7. Repeat until the various estimations are all within a band-width

Result = Average of the various estimations



Often one-point-estimations are best-cases

Uncertainty of Estimation is considered

Asking experts for three estimations:

- *Optimistic*: Best Guess, Lowest Value
- *Realistic*: Largest Probability
- *Pessimistic*: Worst Case, Highest Value

$$\text{Expected Value} = \frac{\text{Optimistic} + 4 * (\text{Realistic}) + \text{Pessimistic}}{6}$$

Assumption: Linear relationship between „size“ and „estimation result“ (development effort, test effort, cost, etc.)

Example:

$$\text{Development Effort (PM)} = C \times \text{„Size of Module“ (in loc)}$$

Factor C must include influence factors like complexity, programming language, etc.

Issue: Recording of similar project data over a long time period necessary (difficult!)

Assumption: Estimates are more accurate if you know the measures from similar projects.



Steps:

1. Determine detailed size, effort and cost measures of the old project. If possible use a WBS based approach (or something similar).
2. Compare the size of the new project bit by bit with the size of the old project.
3. Determine the estimate for the new project as a percentage of the old project.
4. Calculate the effort estimation based upon the comparison of the old and the new project's sizes.
5. Check if the assumptions made in the old and the new project are consistent!

That's not really an estimation method on its own

Align values from one phase to another phase by applying factors

Factors must be determined per project category/ product/ etc.

Usage:

- Calculate the effort / duration/ etc. of the whole project if you know the data from the first phase.
- Consistency checks
- This method is also applied for „activities“ and not only for „phases“
e.g. „Effort for Review“ = X % „from the development effort“

Idea:

- Often experts are not able to estimate that ...
 - a specific features is implemented in exactly 253 LOC
 - you have to implement 400 test cases in your project
 - etc.
- Determine a “Proxy” for the value to be estimated
- It should be easier to estimate the proxy than the parameter you really want to estimate, or
 - the value of proxy is available earlier in the project than the parameter you really want to estimate
 - But of course:
Proxy must be correlated to the parameter you really want to estimate



Use historical data to derive these values.

Estimation within the new project:
Same criteria for “small”, “medium”, “large”, etc.
must be used!



You can achieve this ...

- Using the same persons for estimation
- Training of the people who are estimating
- Define Criteria for the categories

“Law of large numbers”:

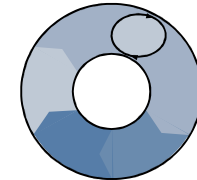
- Fuzzy logic cannot be used to estimate a single feature.
- You need a large number (at least 20)
→ Some estimates could be totally wrong, but in average the estimation will be good!

If you don't have historical data:

1. Classify your features within your project
 2. Use some reasonable data (experts intuition)
 3. Start implementation
 4. After the first iteration:
 - Calculate the "team velocity"
 - Calibrate the estimation with project data!
- See also "Story Points" and "Planning Poker" in agile projects (→ Cohn, 2006)



Method



- Card deck with the following cards 0, ½, 1, 2, 3, 5, 8, 13, 20, 40, 100, ∞, coffee break
- Calibration item
- The requirement (user story) is explained
- The team asks questions
- When all questions are answered, everyone in the team selects one card.
- All cards are placed on the table simultaneously

Agreement

- ideal: everybody estimated the same value
- if not, the person with the
 - lowest value explains his estimation
 - highest value explains his estimation
 - Everyone estimates again→ Repeat maximum two times
- If no agreement is reached a different way for getting an agreement should be found

First comes counting, then calculating. Very last option is expert intuition!

Use historical data. If you don't have historical data, start small, but immediately.

Use the 3-point-estimation approach to deal with uncertainty and to get better results!

Estimation by analogy is often the appropriate approach in industry.

Delphi-Methods help to minimize uncertainty.

Planning Poker is a kind of "Delphi Method" for agile projects.

T-Shirt Sizing is used in an early project stage to prioritize user stories.



Examples of Metric based estimating methods

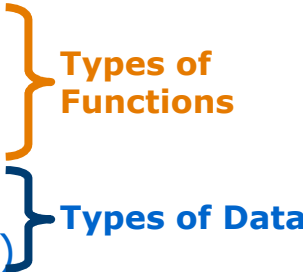
Function Point Analysis (FPA)

- Evaluation of functionalities ("Functions") considering their Type, their Complexity and also additional influencing factors
- The result (so called Function Points) is then used to calculate the development effort, based on experience-based productivity data

Constructive Cost Model (CoCoMo) [Barry W. Boehm]

- Estimate of the number of statements in a software application
- Effort calculated under consideration of: Quality objectives, Type of System, Size of the Project and Productivity parameters

Procedure:

1. Functions (on user level) are evaluated considering type and complexity Typ → unadjusted function points
 - Inputs („external input – EI“)
 - Outputs („external output – EO“)
 - Queries („external inquiry – EQ“)
 - Internal Data („internal logical file – ILF“)
 - External Reference Data („external interface file – EIF“)
2. Adjusting with impact factors
→ adjusted function points
3. Effort calculation using productivity data

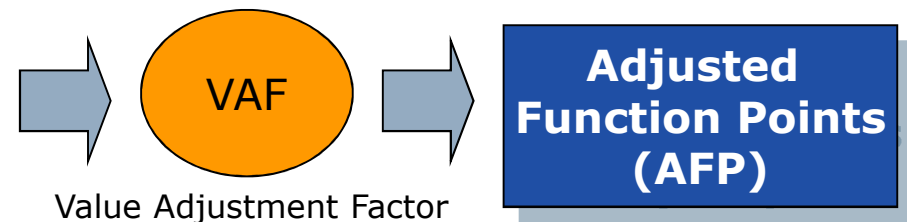
Today:

- Step 2 and 3 are not done, instead use Cocomo

Analysis and evaluation of the influencing factors

14 "General System Characteristics" (GSC) to be scored

- Data Communication
- Distributed data processing
- Performance
- Heavily used configuration
- Transaction rate
- Online data entry
- End user efficiency
- Online update
- Complex processing
- Reusability
- Installation ease
- Operational ease
- Multiple sites
- Facilitate change



$\text{TDI} = \text{Sum of the 14 „GSC“}$

$\text{VAF} = 0,65 + (\text{TDI}/100)$

$\text{AFP} = \text{UFP} * \text{VAF}$

- Constructive Cost Model: Empirical model based upon statistical analysis
- Introduced first by Barry Boehm 1981 (COCOMO 81), Center of SW Engineering, University South California
- Today: COCOMO II
- Industry data was collected over more than 25 years and used to derive this empirical model
- Usage of cost drivers and scaling drivers
- Basic Idea:
$$\text{Effort} = C * (\text{Size})^S$$

- Function Point Analysis is used for size estimation. COCOMO used to calculate the effort based upon the size.
- FPA and COCOMO contain industry experience collected over more than 25 years.
- Even you do not want to use COCOMO as your estimation method, you can consider the values (cost drivers / scaling drivers) within your own estimation method.

