

Lecture 10

Effort and time estimation

24.3.2014

Week	Lecture	Exercise
10.3	Quality in general; Quality management systems	Patterns
17.3	Dependable and safety-critical systems	ISO9001
24.3	Work planning; effort estimation	Code inspections
31.3	Version and configuration management	Effort estimation
7.4	Role of software architecture; product families; software evolution	?
14.4	Specifics of some domains, e.g. web system and/or embedded and real time systems	Break?
21.4	Easter	Break?
28.4	Software business, software start-ups	?
5.5	Last lecture; summary; recap for exam	?

Learning goals

- The factors of time and effort
- Estimation techniques
 - COCOMO II
 - FISMA
- Division of work
 - Distributed work
 - Global SW Development

Structure of the lecture

- **A few points about project planning**
- Effort estimation + budgeting
- Distributed and global development

Plan-driven vs Agile

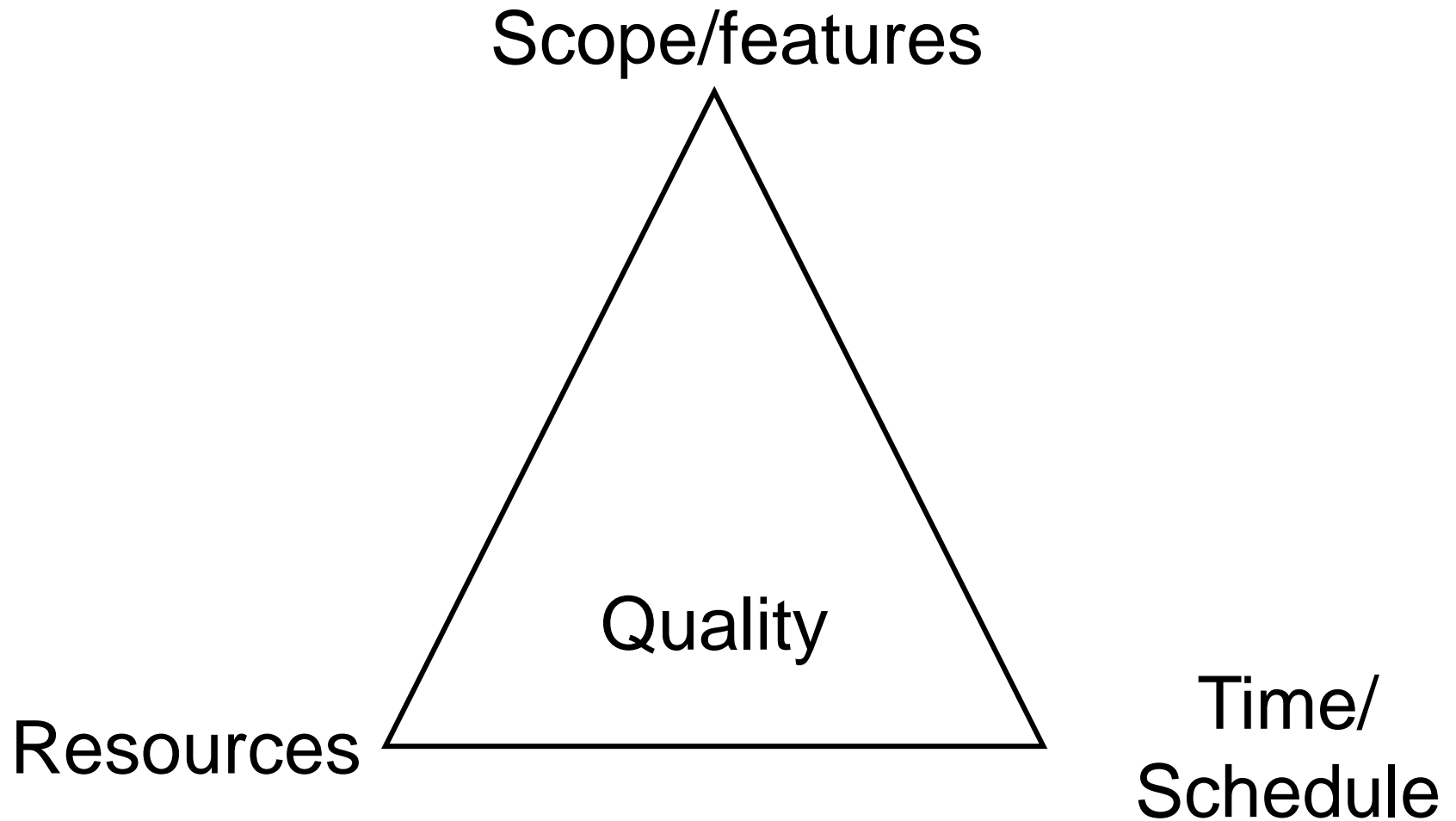
Plan driven

- The order and timing of all events is planned in advance.
- The plan is updated during the work.
- Accuracy improves during the project.
- It is very hard to estimate effort in the beginning

Agile

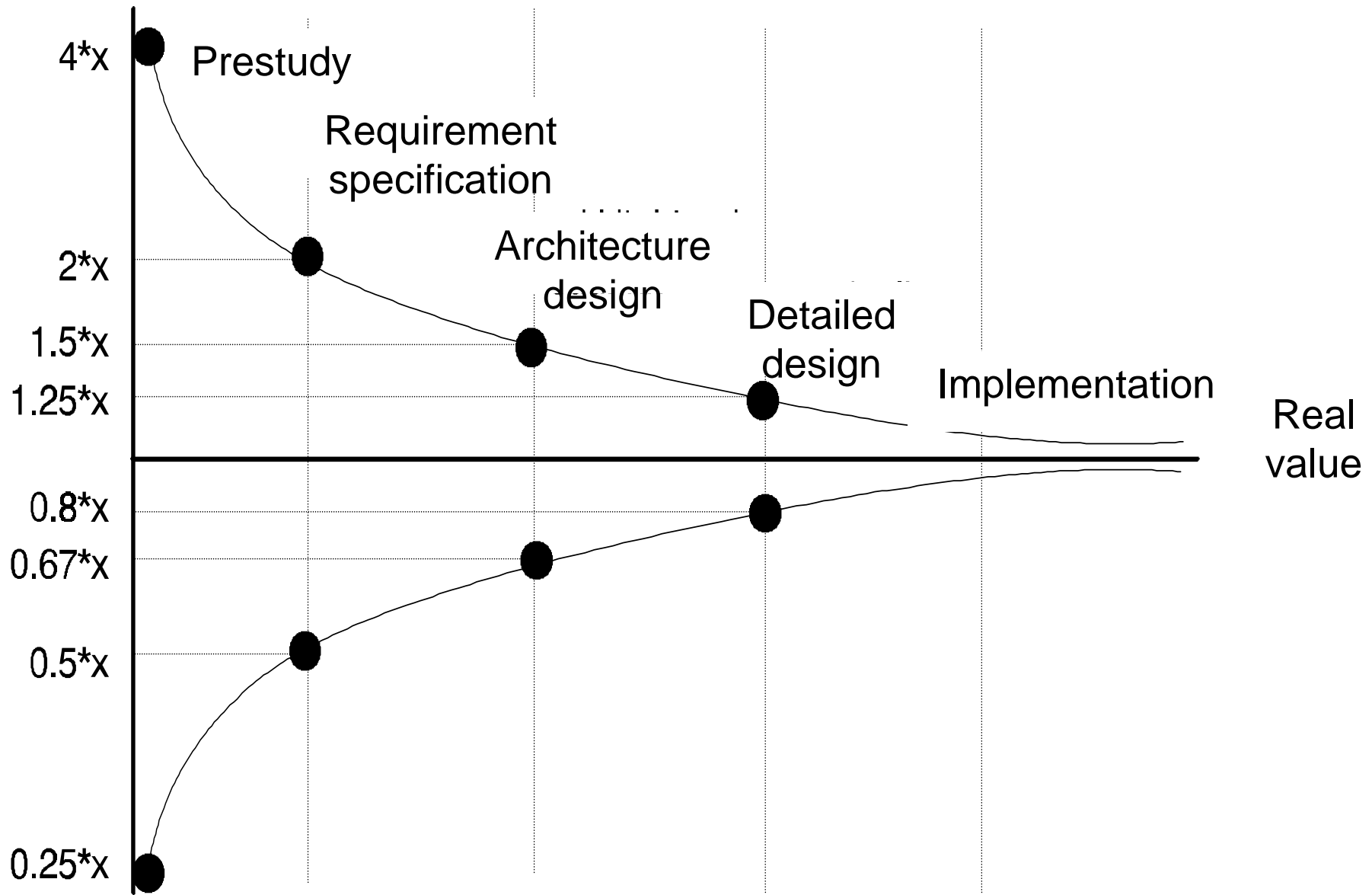
- Only major milestones (e.g. releases) planned in advance.
- Content of each sprint is planned just before the sprint according to current customer requirements
- In many Agile methods strict time-boxing drives the work
- Mini effort estimation in the beginning of every sprint

Remember the iron triangle



Estimates improve as the project progresses

Picture 12.8 in Haikala&Mikkonen, 23.9 in Sommerville



Plan-driven scheduling

- Start with constraints.
- Make work breakdown structure
 - Including dependencies
- Estimate efforts for tasks (e.g. days) – best to take input from several people
- Check availability of resources
- Put to calendar
 - Who, when
- Project planning tools help in mechanical work
 - The challenge is the "guessing" in advance



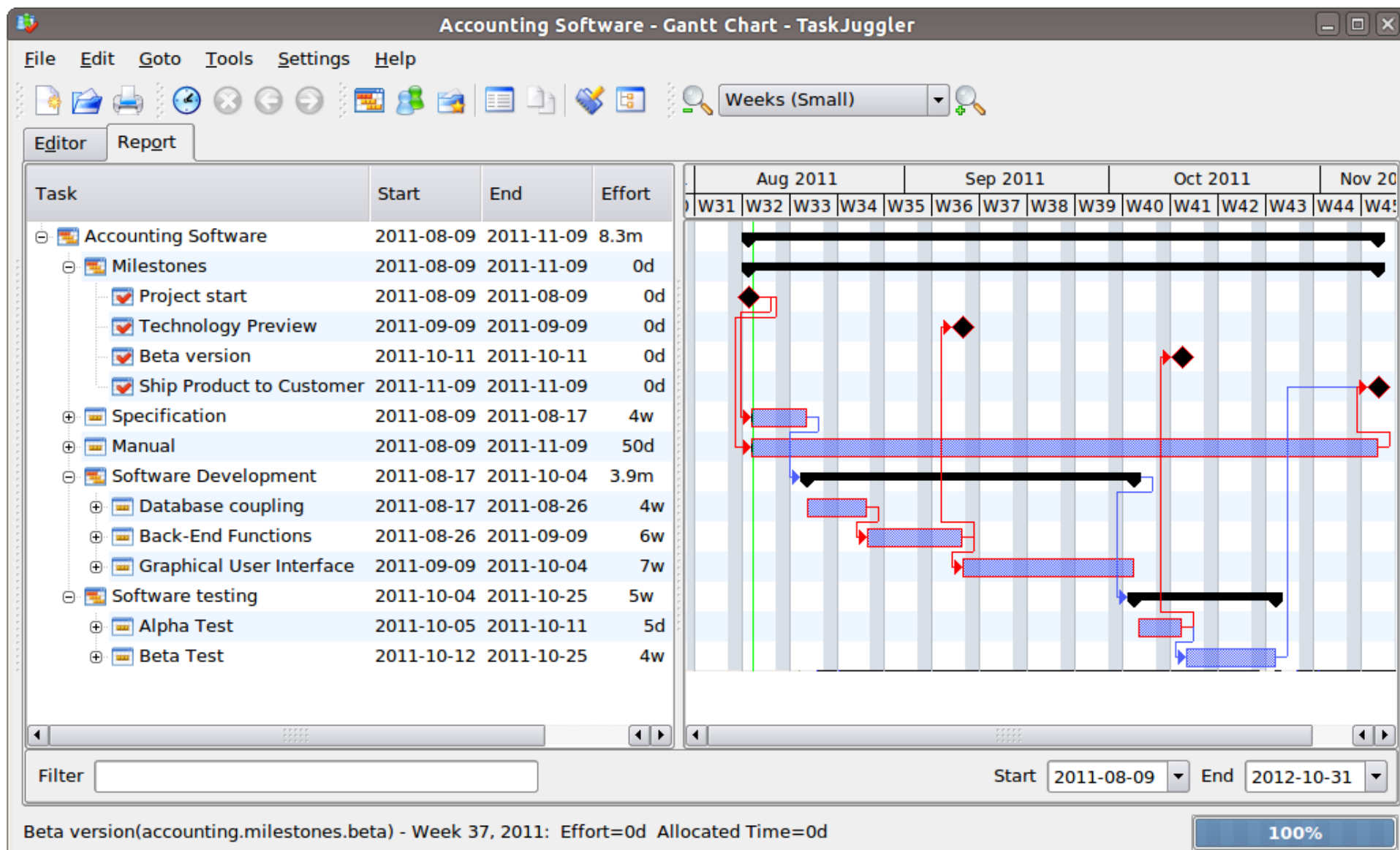
Examples of constraints

- Software must be in operational use 01.01.2016
- 3-man project
- Can use at least Ahto and half of Teemu's time
- New version of database software is not available earlier than May 2015.



The tools often use Gantt Charts

(Source: <http://orgmode.org>)



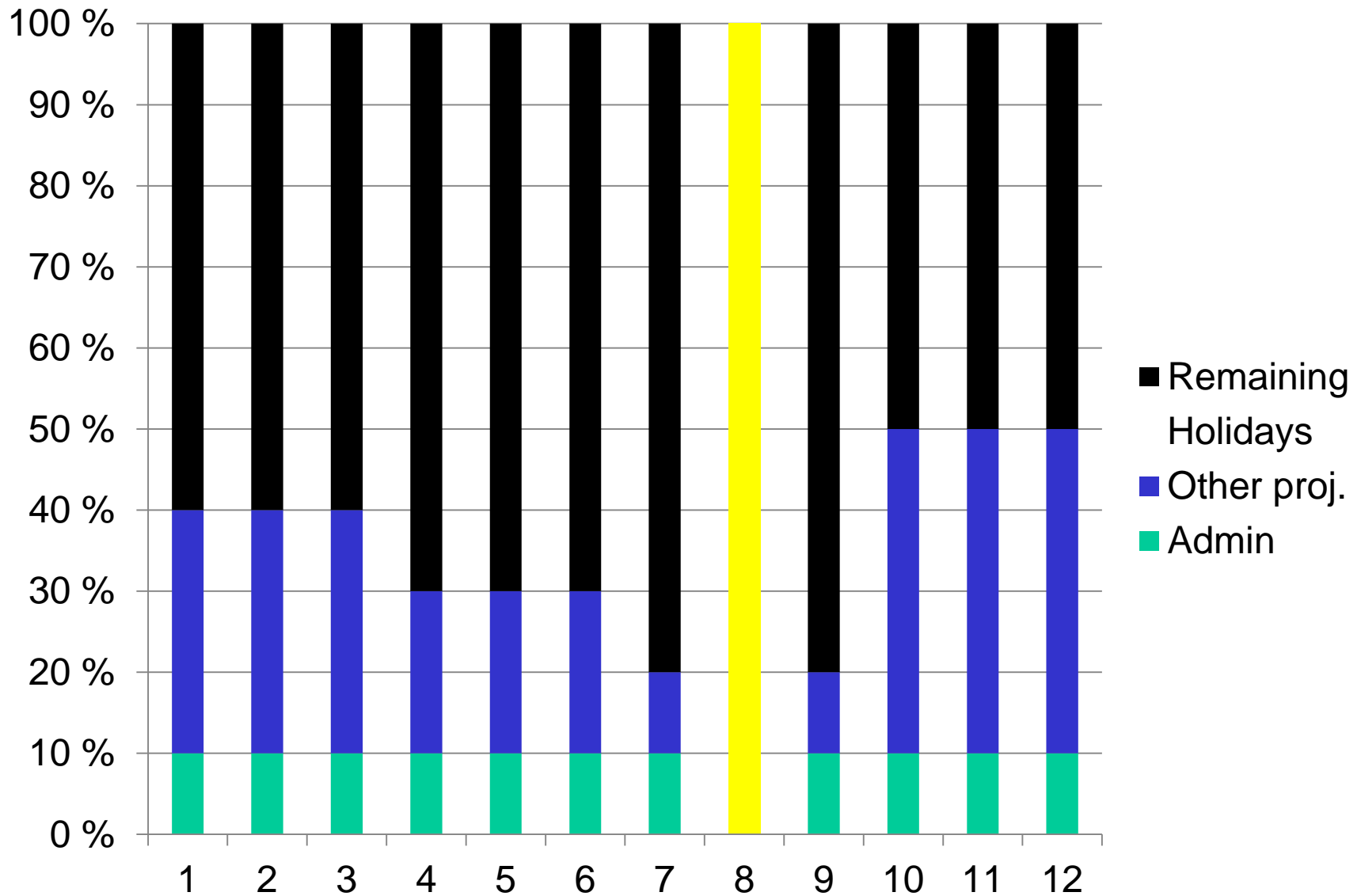
Availability and contribution

(NOTE: assume detailed planning, but these things should always be remembered)

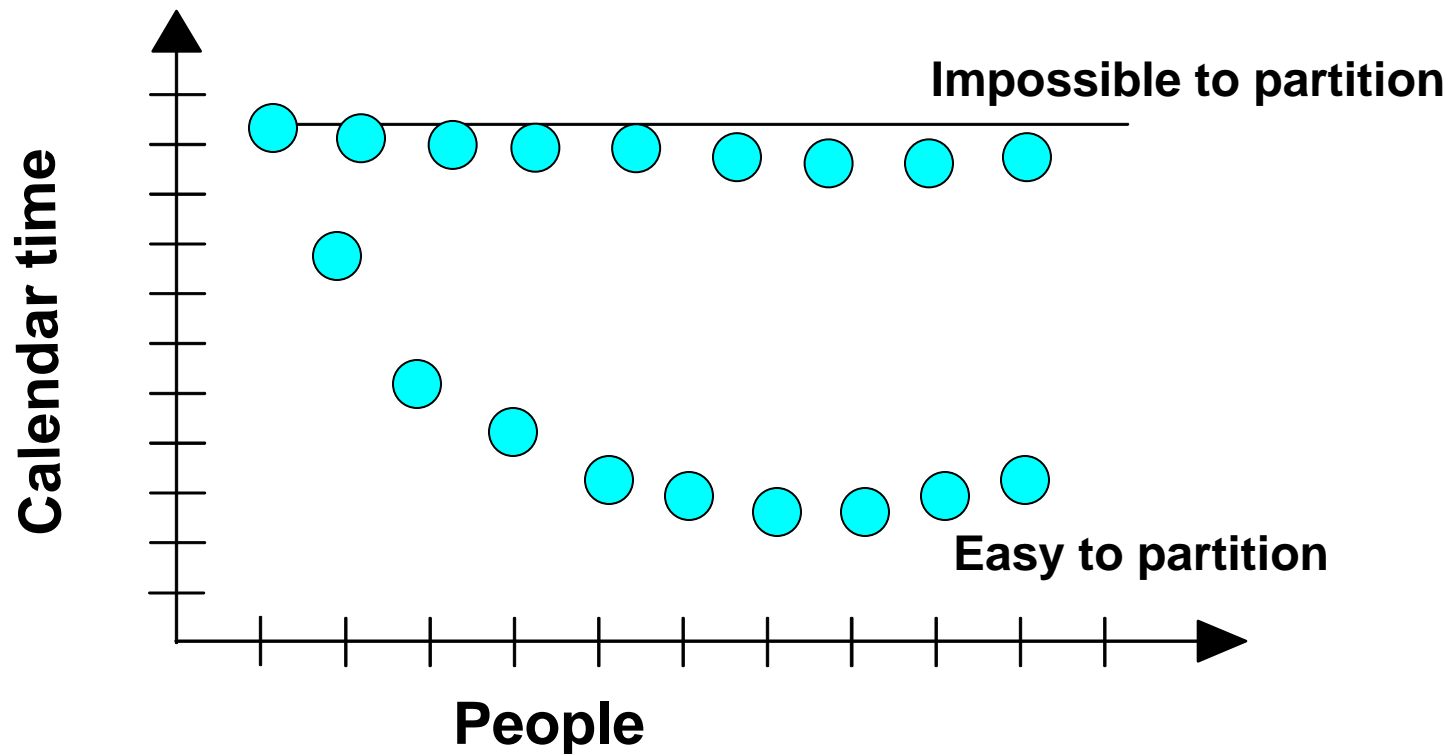
- We need to know contribution of all members (typically assume 5 days a week)
- Reduce vacations, training, responsibilities on other projects
- Reserve time for surprising tasks
- Reduce non-productive work (admin, meetings, travel)
- The remaining “productive work” should be used in remaining planning

Recall WIP and
fast lane in Kanban

Estimation of available effort



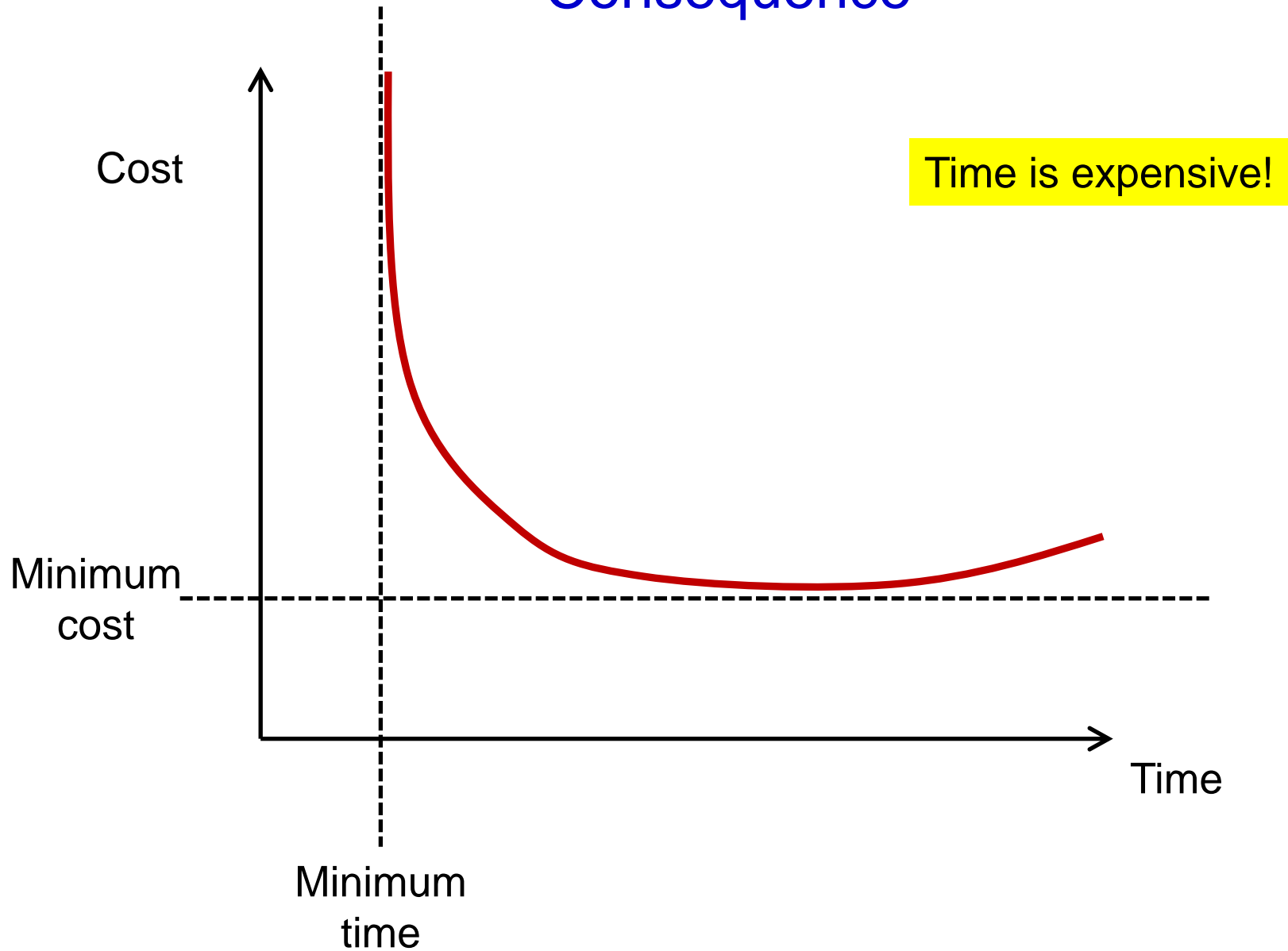
Partitioning of the project



What one programmer can do in a day,
two programmers can also do in day.

A man's got to know his limitations. -- Dirty Harry.

Consequence



But time is also money

(From an imaginary example from a keynote by professor Jan Bosch)

In company X

- R&D is 10% of revenue, e.g. 100M€ for a 1B\$ product
- New product development cycle: 12 months
- Alternative 1: improve efficiency of development with 10%
=> 10 M€ reduction in development cost
- Alternative 2: reduce development cycle with 10%
=> 100M€ add to top line revenue
(product starts to sell 1.2 months earlier)

Structure of the lecture

- A few points about project planning
 - Effort, resources and time don't have linear dependencies
- **Effort estimation + budgeting**
- Distributed and global development

Effort estimation

- Necessary for planning: timetable and budget
- Factors
 - Complexity of the software – often hard to know in advance
 - Productivity of the team
 - Skills
 - Experience
 - Spirit
 - Required quality
 - Timetable (recall previous slides)

Why so difficult

- Productivity of individuals vary (10-times is not rare)
 - And team dynamics make situation event more complex
- The exact requirements are seldom known
 - Plan driven: customer changes mind => plan is changed
 - Agile: replanning for each sprint
- Required implementation effort for each requirement
 - Especially if nothing similar has been done earlier
 - Surprising technical problems often appear

...why so difficult

- Too many unknowns
- Wishful thinking
- Too much self-confidence
- People have their own agendas
 - For example sales want to win the deal
(Too often the worst estimate wins!)
- Too complex and big system under estimation
- Lack of historical data

Methods for effort estimation

- Planning poker
- Cocomo (constructive cost model) – will be described today
- FPA (function point analysis)
- Use your experience and historical data
 - Collect information for next projects



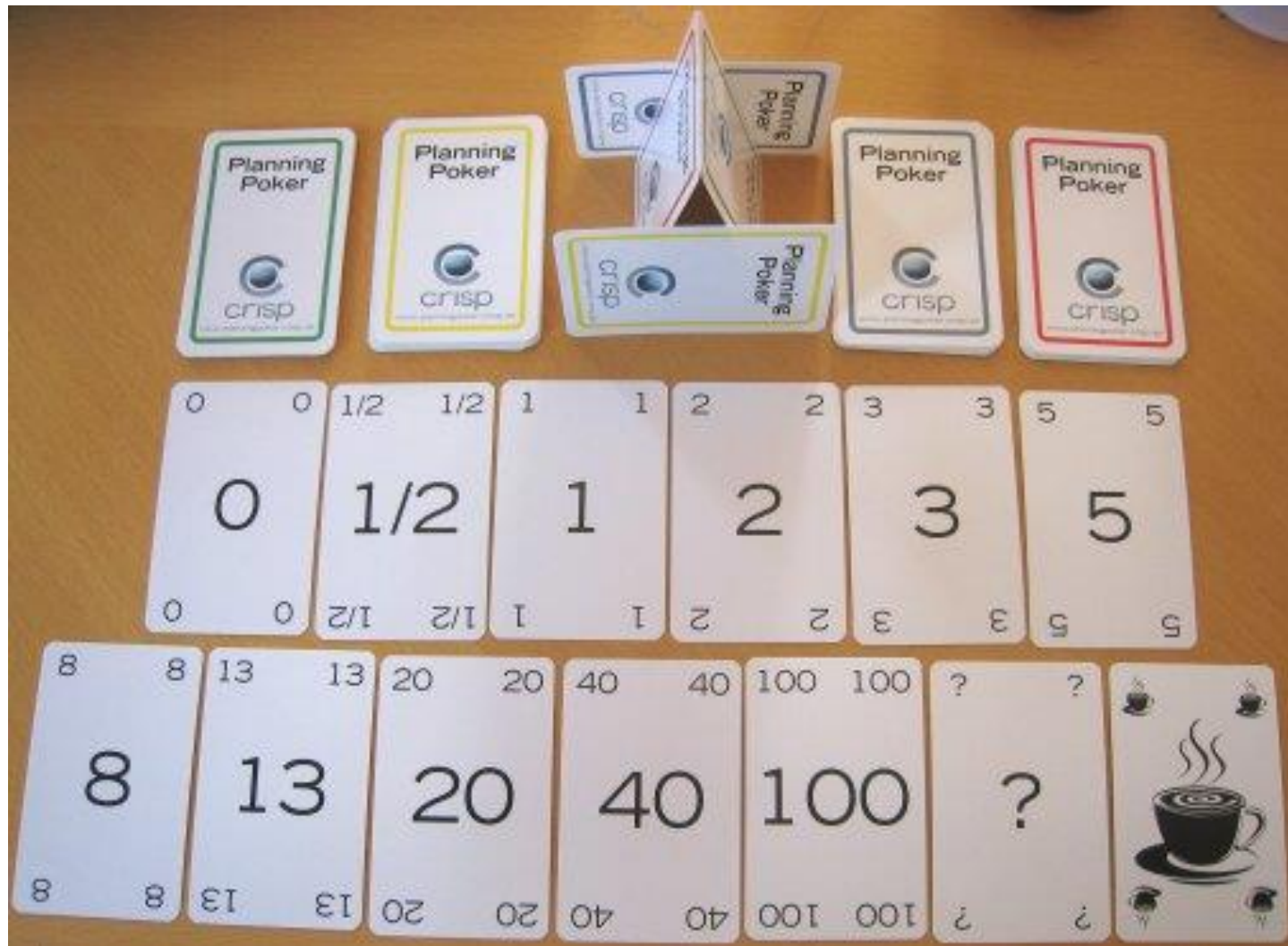
Planning poker - what

- Often used in agile projects
- Collects and combines understanding from several participant.
 - Developers are very involved
- [K. Molokken-Ostvold, N.C. Haugen]
 - More accurate and less optimistic than mechanical methods to combine individual estimates
 - Competes with expert estimation

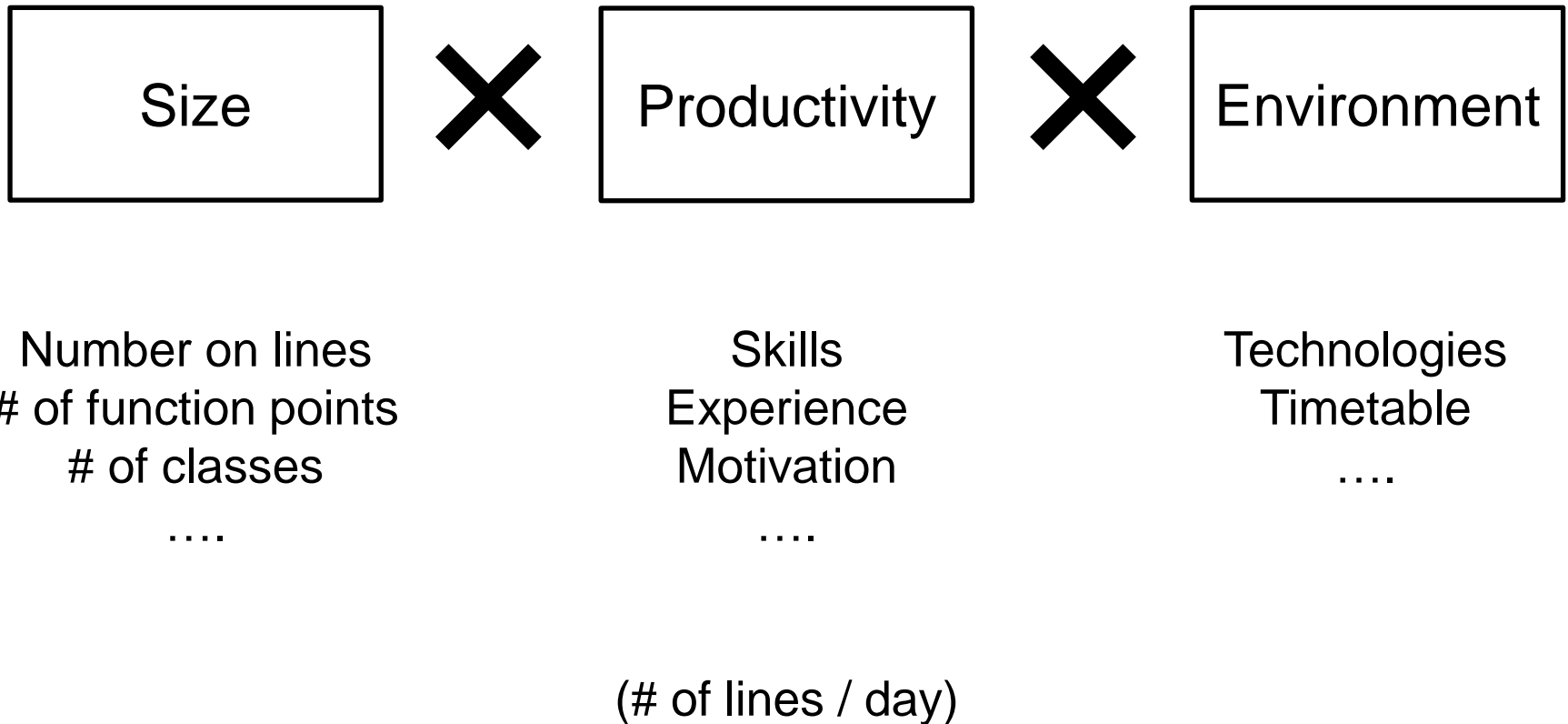
Planning poker - how

- Participants get a deck of cards with numbers
 - Often Fibonacci series 0, 1, 2, 3, 5, 8, 13...
- Somebody present the task to be estimated
- Everybody shows a card that describes his or her opinion about the effort
 - The cards are shown synchronously (at the same time)
- Those who are different from common opinion defend their view
- As long as there are different opinions repeat

Cards for planning poker



General model



Simple formula from Sommerville

$$\mathbf{Effort} = \mathbf{A} \times \mathbf{Size}^{\mathbf{B}} \times \mathbf{M}$$

- A is a constant that represent local practices and type of the software
- Size is size, e.g., in function point
- B is in between 1 and 1.5 and reflects the fact that size do not affect linearly the effort
- M is multiplier combining process, product and organizational aspects

COCOMO

- First version about n. 1980.
- Our material is based on COCOMO-II
- Input:
 - Size of the product (SLOC, source lines of code)
 - Scaling factors of the whole product
 - Cost drivers for different parts of the project, about properties of
 - Product
 - Development methods, tools etc
 - Developers
 - Project
- Outcome
 - Effort
 - Calendar time
- Four submodels: application composition, early design, reuse, and post-architecture

COCOMO submodels

- Application composition
 - Can be used when SW is mainly composed from existing components with scripting-like approach
- Early design
 - As the name says this approach is used when only draft design of the system exists
 - Based on 7 multipliers
- Reuse
 - Used when reusable components are used
- Post-architecture
 - After architecture design has finalized, a more accurate estimation can be done

Effort, PM = person months

Nominal effort

$$PM_{NS} = A * Size^E * EM_1 * EM_2 \dots * EM_n$$

A = effort coefficient, that can be calibrated according to situation

Based on large data set Boehm proposes 2.94

Could be interpreted as months per KSLOC

Size is KSLOC

EM_i are coefficients (7 in early design and 17 in post design)

E is a scaling exponent between 1.1 and 1.24 – depending on novelty

$$E = 0.91 + 0.01 * (SF_1 + SF_2 + SF_3 + SF_4 + SF_5)$$

(E is in between 1.1 and 1.24 – depending on novelty)

Note, Sommerville's book uses 'B' instead of 'E'

Calendar time

$$\text{TDEV}_{\text{NS}} = C * (\text{PM}_{\text{NS}})^F$$

C calibration factor, some material gives 3.67,
Sommerville uses 3

F is a scaling exponent,

$$\begin{aligned} F &= D + 0.2 * 0.01 * \text{SF}_1 * \text{SF}_2 * \text{SF}_3 * \text{SF}_4 * \text{SF}_5 \\ &= D + 0.2 * (E - 0.91) \end{aligned}$$

Where

D is a calibration term, some material propose 0.28,
Sommerville 0.33

How to estimate with COMOMO?

1. Estimate scaling factor SF_i ($i=1..5$) and calculate scaling exponent E .
2. Split product to independent parts to be estimate..
3. Estimate size of each part, considering reuse and automatic code generation
4. Estimate cost factors EM_i and calculate nominal person months PM_{ns}
5. Calculate development time $TDEV_{NS}$.
6. If one want to deliver in shorter time. Choose SCED value “low” or “very low”, when $TDEV$ drops max 25%. Correspondingly PM increases due to bigger SCED-factor.

Scaling factors SF_i

- Each factor estimated in 6 level scale:
 - very low, low, nominal, high, very high, extra high

	VL	L	N	H	VH	EH
Precedentedness	6.20	4.96	3.72	2.48	1.24	0.00
Development Flexibility	5.07	4.05	3.04	2.03	1.01	0.00
Architecture/Risk Resolution	7.07	5.65	4.24	2.83	1.41	0.00
Team Cohesion	5.48	4.38	3.29	2.19	1.10	0.00
Process Maturity	7.80	6.24	4.68	3.12	1.56	0.00

From those:

$$E = 0.91 + 0.01 * (SF_1 + SF_2 + SF_3 + SF_4 + SF_5)$$

$$F = 0.28 + 0.2 * (E - 0.91)$$

https://files.ifi.uzh.ch/rerg/arvo/courses/seminar_ws02/reports/Seminar_4.pdf

	Early Design cost drivers	Post-Architecture cost drivers (Counterpart combined)
Product reliability and complexity	RCPX	RELY, DATA, CPLX, DOCU
Required reuse	RUSE	RUSE
Platform difficulty	PDIF	TIME, STOR, PVOL
Personnel capability	PERS	ACAP, PCAP, PCON
Personnel experience	PREX	AEXP, PEXP, LTEX
Facilities	FCIL	TOOL, SITE
Required Development Schedule	SCED	SCED

Factors EM – post design

	Product	VL	L	N	H	VH	EH
RELY	Required Software Reliability	0,82	0,92	1,00	1,10	1,26	
DATA	Database Size		0,90	1,00	1,14	1,28	
DOCU	Documentation Match to Lifecycle Needs	0,81	0,91	1,00	1,11	1,23	
CPLX	Product Complexity	0,73	0,87	1,00	1,17	1,34	1,74
RUSE	Required Reusability		0,95	1,00	1,07	1,15	1,24
	Platform			1,00			
TIME	Execution Time Constraint			1,00	1,11	1,29	1,63
STOR	Main Storage Constraint			1,00	1,05	1,17	1,46
PVOL	Platform Volatility		0,87	1,00	1,15	1,30	
	Personnel			1,00			
ACAP	Analyst Capability	1,42	1,19	1,00	0,85	0,71	
APEX	Applications Experience	1,22	1,10	1,00	0,88	0,81	
PCAP	Programmer Capability	1,34	1,15	1,00	0,88	0,76	
PLEX	Platform Experience	1,19	1,09	1,00	0,91	0,85	
LTEX	Language and Tool Experience	1,20	1,09	1,00	0,91	0,84	
PCON	Personnel Continuity	1,29	1,12	1,00	0,90	0,81	
	Project			1,00			
TOOL	Use of Software Tools	1,17	1,09	1,00	0,90	0,78	
SITE	Multisite Development	1,22	1,09	1,00	0,93	0,86	0,80
SCED	Required Development Schedule	1,43	1,14	1,00	1,00	1,00	
	Schedule Compression Percentage	0,75	0,85	1,00	1,30	1,60	

Simple example

- Compiler for a Pascal-language (Language from history books)
 - 50000 source code lines
 - Skillful develops
 - New HW platform
- Assume normal scaling factors
 - $E = 0.91 + 0.01 * (3.72 + 3.04 + 4.24 + 3.29 + 4.68) = 1.1$
 - $F = 0.28 + 0.2 * (E - 0.91) = 0.32$
- Cost factor personnel & project-tekiä are VH, other N
 - $PM_{ns} = 2.94 * 50^{1.1} * 0.71 * 0.81 * 0.76 * 0.85 * 0.84 * 0.81 * 0.78 * 0.86$
 $= \sim 37 \text{ person months}$
 - $TDEV_{NS} = 3.67 * 37^{0.32} = \sim 12 \text{ months}, \Rightarrow 3 \text{ person in one year}$
- If schedule is tighter, SCED increases effort. For example if SCED=VL
 TDEV decreases 25% (9 months), but effort raises to $\sim 53 \text{ PM}$
 - I.e. 6 people for 9 months!

Another example

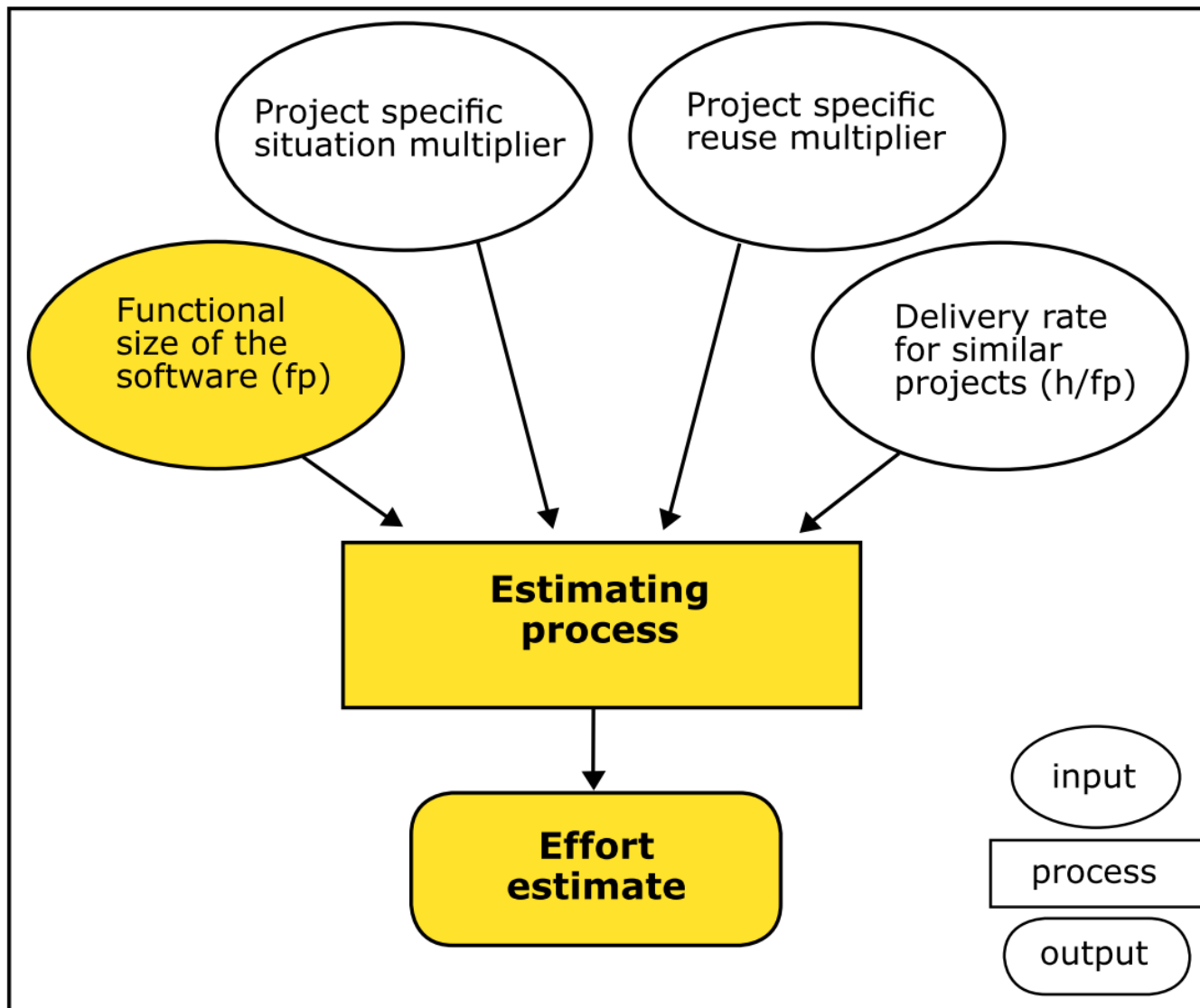
- <http://www.codeproject.com/Articles/9266/Software-Project-Cost-Estimates-Using-COCOMO-II-Mo>

Function points

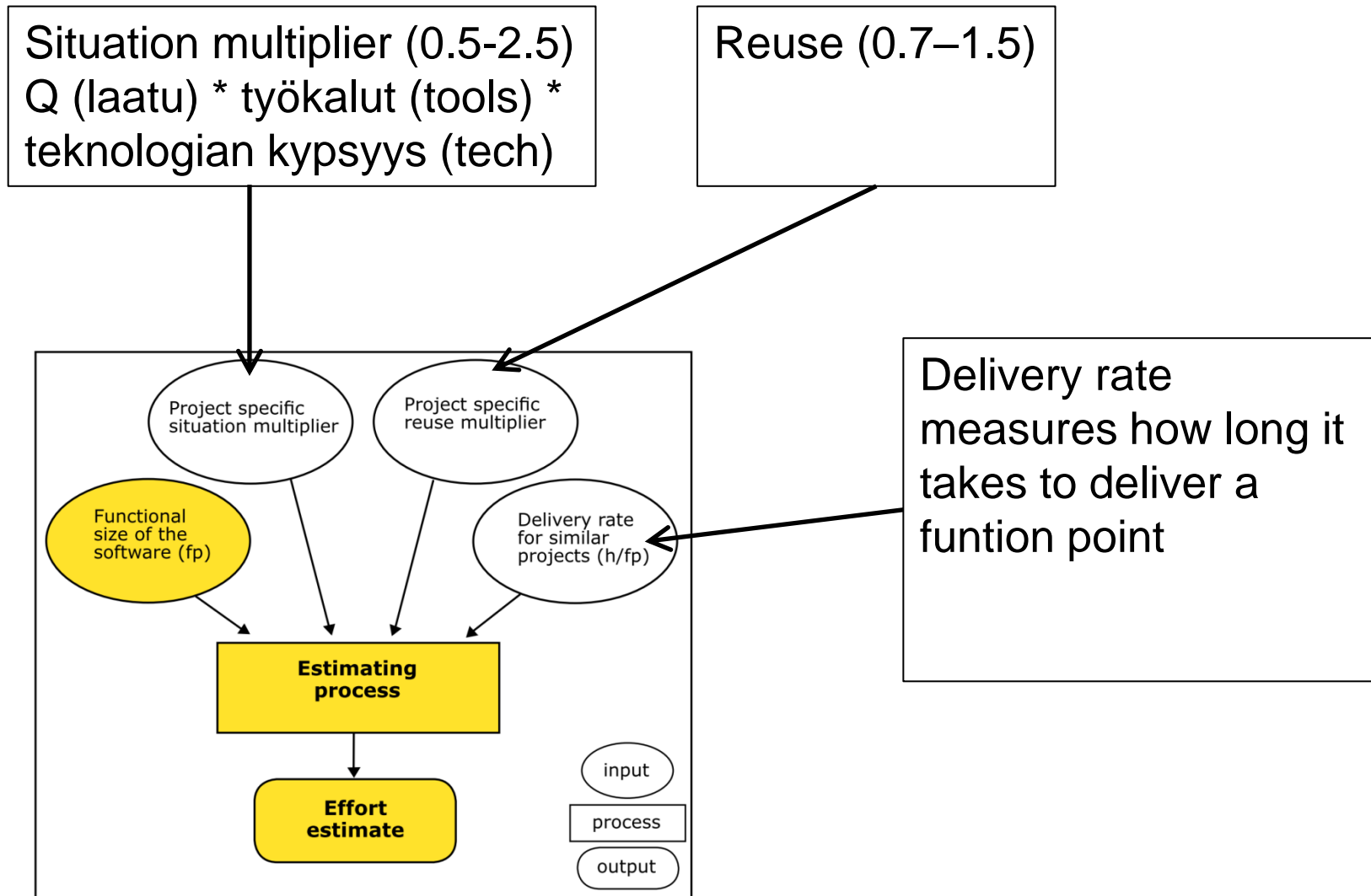
- Wikipedia definition: A function point is a unit of measurement to express the amount of business functionality an information system (as a product) provides to a user.
- Should be (but result debated) independent from programming language
- Number of function points can be automatically measured from source code
- **Number of function points can be estimated from (detailed) functional specification of the system**

Estimation based of Function points (FiSMA)

(Source: Pekka Forselius: Toimintopistelaskenta tuottavuuden todentajana)



Estimation based of Function points (FiSMA)



About budgeting with an arbitrary example

- A small software company of 10 employee
- Fulltime manager; no admin personel
- Other employee do billable work 75% of the time
- Reasonable rotation: on average 1 person leaves and joins
- Salaries + compulsory side costs
 $1.6 * 3.5 \text{ k€} * 12 \text{ kk} = \sim 67 \text{ k€}$
- Office, rents, equipments, supplies ... 50%
 $1.5 * 67 \text{ k€} = 100 \text{ k€}$
- 10 employees => costs are $\sim 1000 \text{ k€}$

...example

- Since managers work cannot be invoiced, 9 people generate the revenue.
- Due to rotation, sick leaves, etc, we use estimate 8. Thus the company can invoice:
$$8 * 1700 * 0.75 = \sim 10000 \text{ working hours}$$
- minimum price for a working hour is
$$1000\text{k€} / 10000 = 100\text{€}$$
- Working day costs
$$7.5 * 100 = 750\text{€}$$
- And year
$$1700 * 100 = 170\text{k€}$$



Structure of the lecture

- A few points about project planning
 - Effort, resources and time don't have linear dependencies
- Effort estimation
 - Art, magic ... and deep understanding
- **Distributed and global development**

Reasons for distributed development

- Size of the project
 - One team cannot finish in time
 - Not enough resources available otherwise
- Need for additional competences
 - HW, databases, UI
 - Domain knowledge
- Economic
 - In some places development is cheaper

Global SW development

- SW developed by a geographically distributed organization
 - Reasons like above
- Often main reason is lower salary costs
 - Also called off-shore development
 - Real savings debated
- Issues
 - Communication distance: (timezone, language cultural..).
Serious issue since communication is major challenge in SW
 - Spirit, motivation, commitment
 - "They and us" attitude
 - No feeling of ownership

Guidelines for planning distributed projects

- Design for partitioning
 - Or it will happen anyways
 - Conway's law: " organizations which design systems ... are constrained to produce designs which are copies of the communication structures of these organizations"
- Help communication
- Add extra effort for
 - Communication
 - Solving problems due to bad communicationin your plan