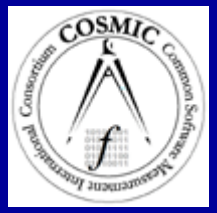


From requirements to project effort estimates – work in progress (still?)

Charles Symons

Founder & Past President, The Common Software Measurement International Consortium

Cigdem Gencel

Assistant professor and senior researcher at the Faculty of Computer Science of the Free University of Bolzano, Italy

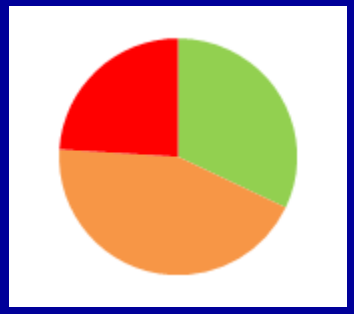
REFSQ Conference , Essen, Germany , April 2013

Agenda

- Estimating: the issues
- How can we improve the early estimation of project effort from requirements?
- What about Non-Functional Requirements?
- Conclusions for Requirements Engineers

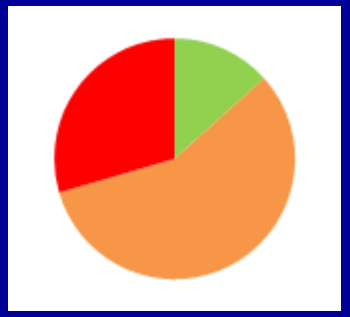
Software industry delivery to time and budget is notoriously bad

Standish CHAOS study 2009 [1]



European Union Study '98 – '05 [2]

UK Public Sector Study 2007 [3]



ISBSG Study 2013 [4]

Failed

>10% over budget

Successful

>10% over budget

'Successful'

The cost of these over-runs and failures (all to the Customers) is a scandal

Study Country	No. of Projects	Cost	Over-runs/ Write-offs
UK Public Sect. [4]	105	£ 29B	£ 9B (31%)
Mostly US [5]	1471	$ 246B	$ 66B (27%)

Annual cost of failures and over-runs:

- US market (Standish) ~100 Billion US$
- European market ~100 Billion €

The 'world-class' software suppliers' profit margins on the UK contracts[3]: 10 – 20+ %

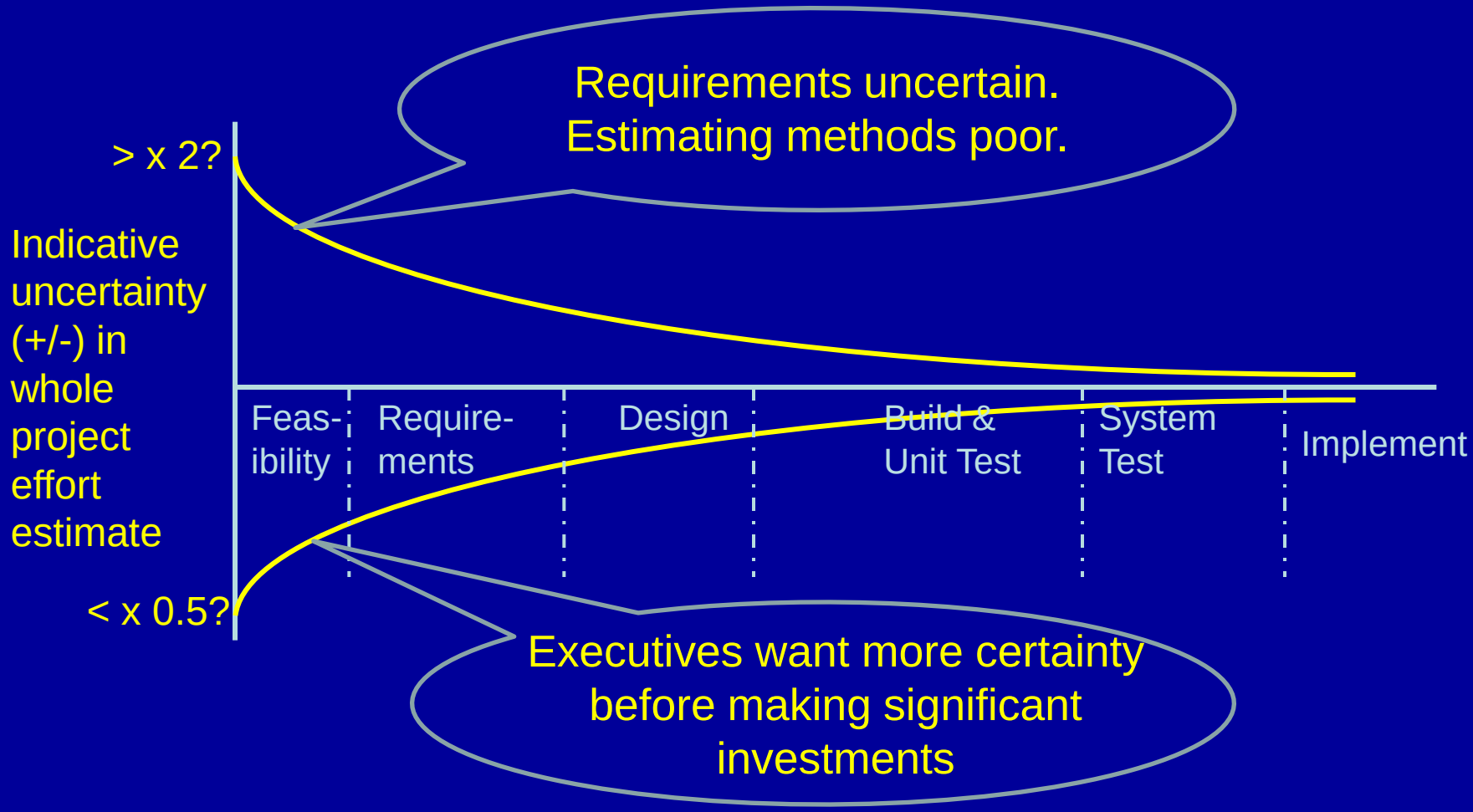
Why do we get this poor delivery to time and budget?

Studies repeatedly show:

- Incomplete and changing requirements
- Poor estimating
- Project management failures
- etc

..... in spite of >20 years of process improvement !

The estimating ‘cone of uncertainty’ is too wide, especially early in a project life



(Agile methods do not solve the problem of whole project estimation)

Agenda

- Estimating: the issues
- How can we improve early estimation of project effort from requirements?
- What about Non-Functional Requirements?
- Conclusions for Requirements Engineers

Most project estimating still relies on informal methods [6]

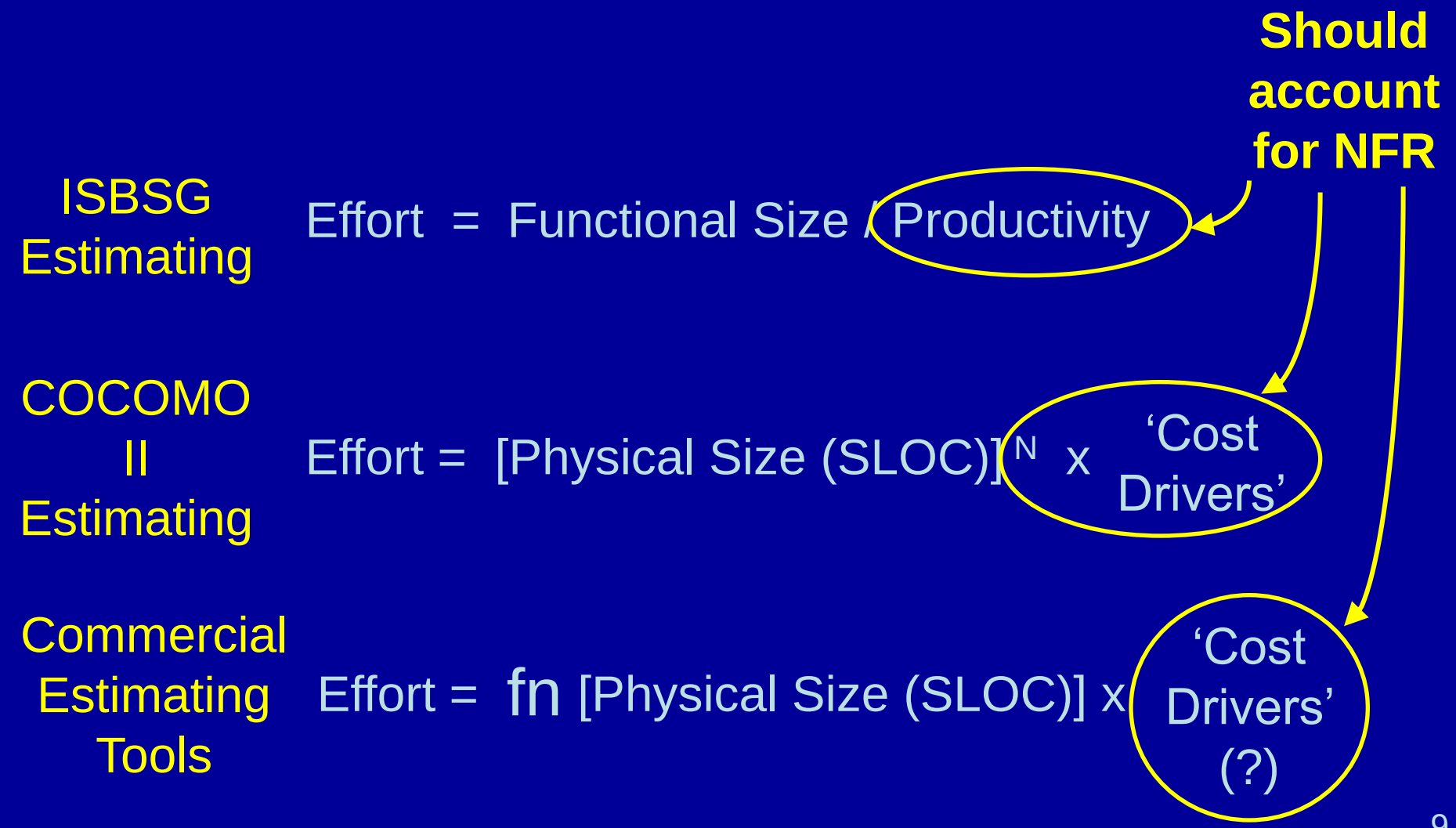
- Expert judgement, estimating by analogy
- Use of benchmark data (e.g. ISBSG [7])
- Open estimating method (e.g. COCOMO [8])
- Black box commercial estimating tools

Most common,
Least data

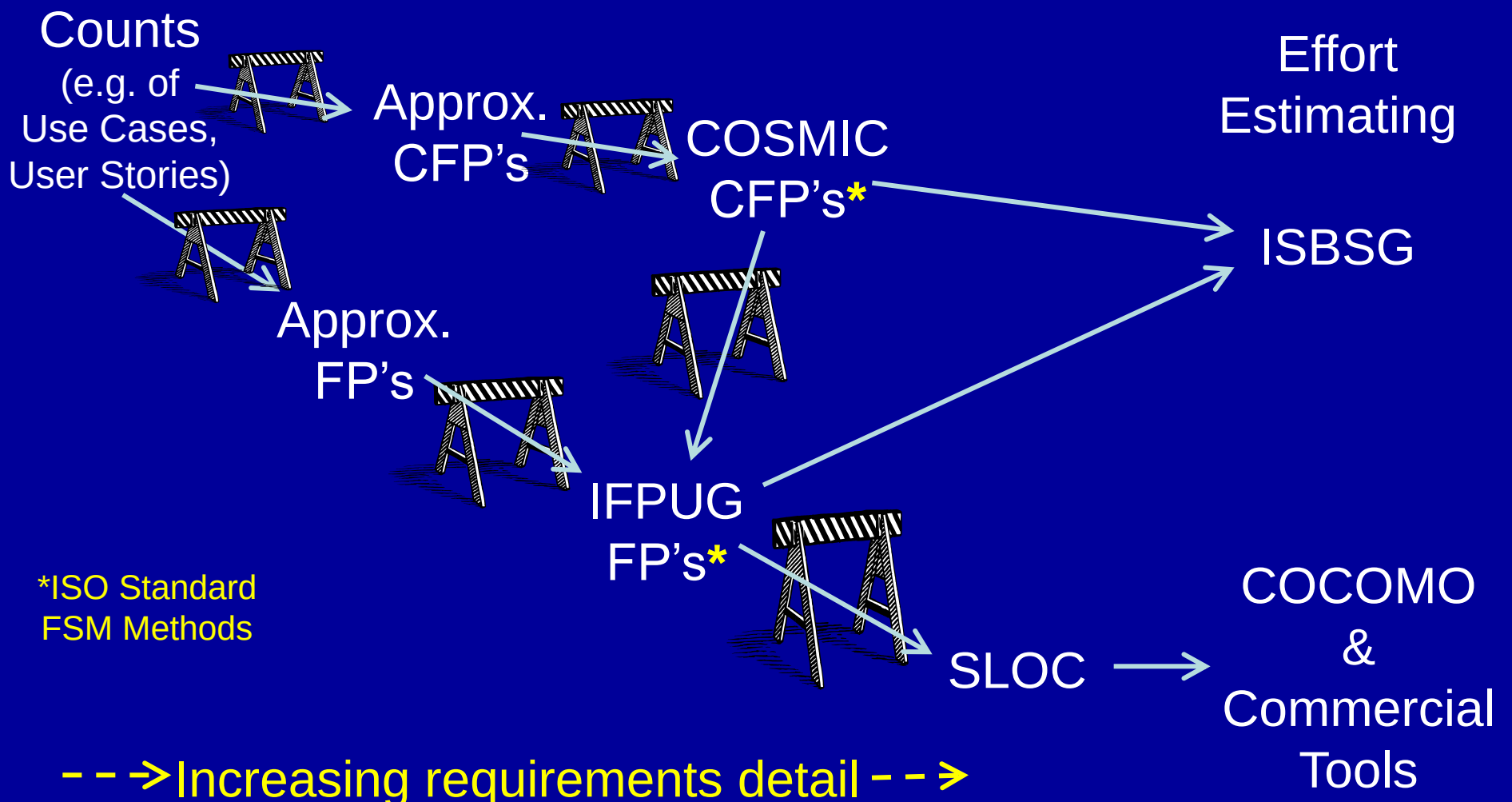
Least common,
Most data

(Ideas in this paper are relevant to all types of estimating methods)

Software ‘product size’ is the biggest driver of project effort



Many methods and 'routes' are used to estimate size, then effort, early in a project



Every conversion from one size-type to another, or to effort, adds to uncertainty

'Error propagation' arises due to [9]:

- Intrinsic differences between the input and output variables (e.g. two size-types) so they do not correlate well
- Errors in measurement of the input(s) to the conversion
- Increasing the number of variables and algorithmic complexity of the conversion

(In principle, the more variables we add to the conversion formula, the greater the accuracy, but the more sources of input measurement errors, resulting in greater error propagation.)

FSM methods can measure a functional size of requirements as they evolve

IFPUG (Albrecht)

- 1970's empirical model
- Business applications
- Counts 'elementary processes' and 'files'
- Limited size range of processes and files
- Most widely used

COSMIC

- Fundamental SE Principles
- Business, real-time & infrastructure software
- Measures 'functional processes'
- No size limit on functional processes
- Rapidly increasing usage

For now, think of IFPUG sizing as a less well-defined and approximate version of COSMIC sizing.

Use Case sizes vary enormously. Conversion to a functional size is only possible locally

Company A – Project Type: I

UC No	# of Trans.	FP (Trans. Size)	CFP	CFP / FP
UC1	1	6	27	4.5
UC2	1	7	25	3.6
UC3	1	6	29	4.8
UC4	3	16	46	2.9
UC5	1	6	30	5.0
UC6	1	6	28	4.7
UC7	9	44	112	2.5
UC8	9	59	122	2.1
UC9	2	12	52	4.3
UC10	2	9	25	2.8
UC11	1	6	30	5.0
UC12	15	88	267	3.0
UC13	10	51	113	2.2
UC14	5	17	24	1.4
UC15	1	6	10	1.7

Different project types may have different:
- # of Transactions / UC
- Average size / UC

Company A - Project Type: II

UC No	# Trans.	FP (Trans.Size)	CFP	CFP/FP
UC1	1	7	22	3.1
UC2	1	7	13	1.9
UC3	1	7	15	2.1
UC4	1	7	25	3.6
UC5	1	7	17	2.4
UC6	1	7	14	2.0
UC10	1	7	13	1.9
UC11	1	7	18	2.6
UC12	1	7	14	2.0
UC13	1	7	20	2.9
UC14	1	6	17	2.8
UC15	1	7	10	1.4
UC16	1	7	17	2.4
UC17	1	7	15	2.1
UC25	4	24	32	1.3
UC26	4	13	16	1.2
UC27	1	6	8	1.3
UC28	4	12	17	1.4

Conclusions for users of UML and for 'Agilistas' wanting to estimate total size

- Learn how FSM methods define a 'transaction'*
- When analyzing requirements, determine the number of transactions in each Use Case or User Story
- Then either use an average functional size of each transaction for:

Total size = (No. transactions) x (Av. size per transaction)

- or use a more sophisticated approximate FSM method e.g. with an average transaction size per 'size band',

'transaction' = IFPUG 'elementary process' or COSMIC 'functional process'

Several factors should be considered to get a smooth functional size/effort relationship

Functional Profile

Hi → Functional similarity → Low

Read/List → DB accesses → Cr/Update/Del

Gather data → Hardware accesses → Control

Low → Maths/logic processing → High

None → Interfaces → Many

Low effort → Relative project effort → High effort

(Not well understood)

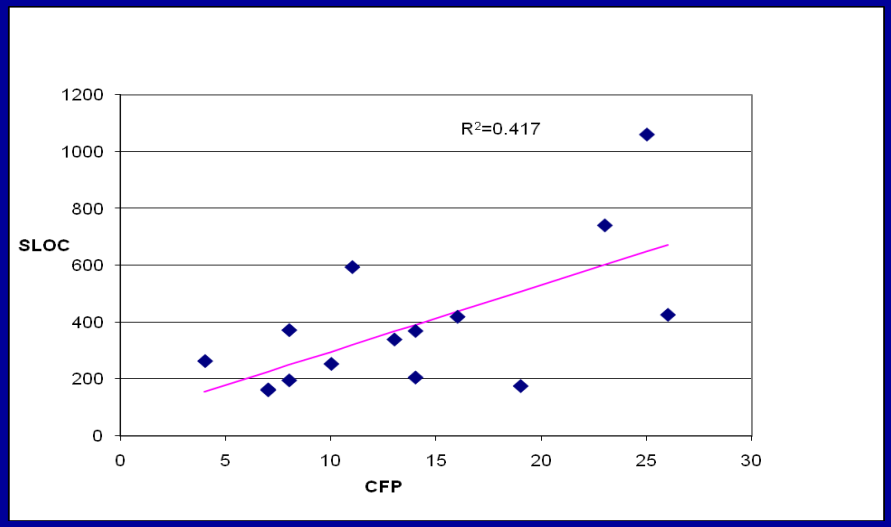
All we know about FS:SLOC conversion tells us that it is very unreliable

Functional Sizes	SLOC Sizes
• A measure of functional requirements; ignores NFR	• A designer's view of size; implements NFR
✓ International standards	x No reliable standards
✓ Technology-independent	x Technology-dependent
• Measurement results show an economy of scale with increasing software size (up to ~2000 FP)	• Measurement results show a dis-economy of scale with increasing software size (up to 1M SLOC)

There is little published evidence to support claimed FP/SLOC conversion ratios for various programming languages

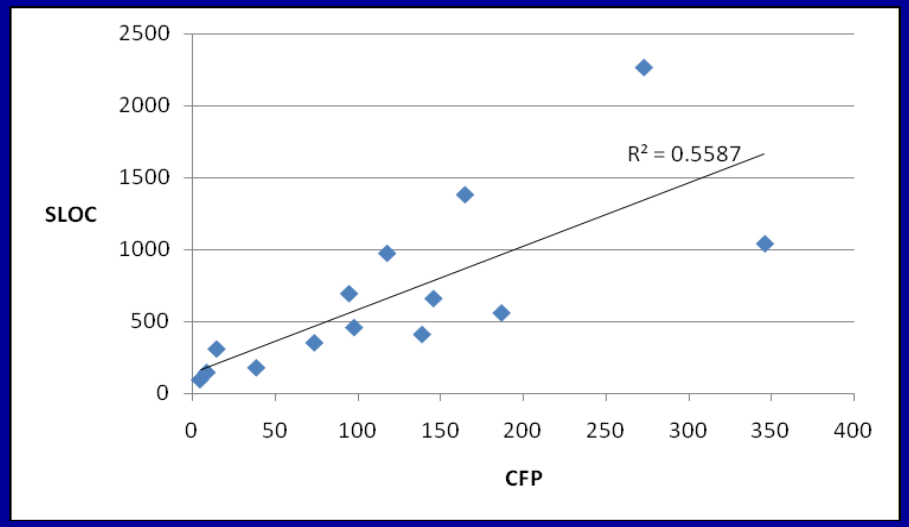
Results confirm that CFP and SLOC sizes do not correlate well (neither do FP and SLOC)

Software components – one automotive company [10]



# of Comp.	SLOC / CFP			
	Min	Med	Max	Std. Dev.
15	9.2	26.1	65.8	15.8

Software projects (new) – ISBSG Dataset



Prog. Lang.	# of Projects	SLOC / CFP			
		Min	Med	Max	Std. Dev.
C++	14	2.95	6.03	20.6	6.04

However, CFP and memory size (bytes) correlate very well

Using COCOMO II to estimating effort from SLOC also has intrinsic difficulties

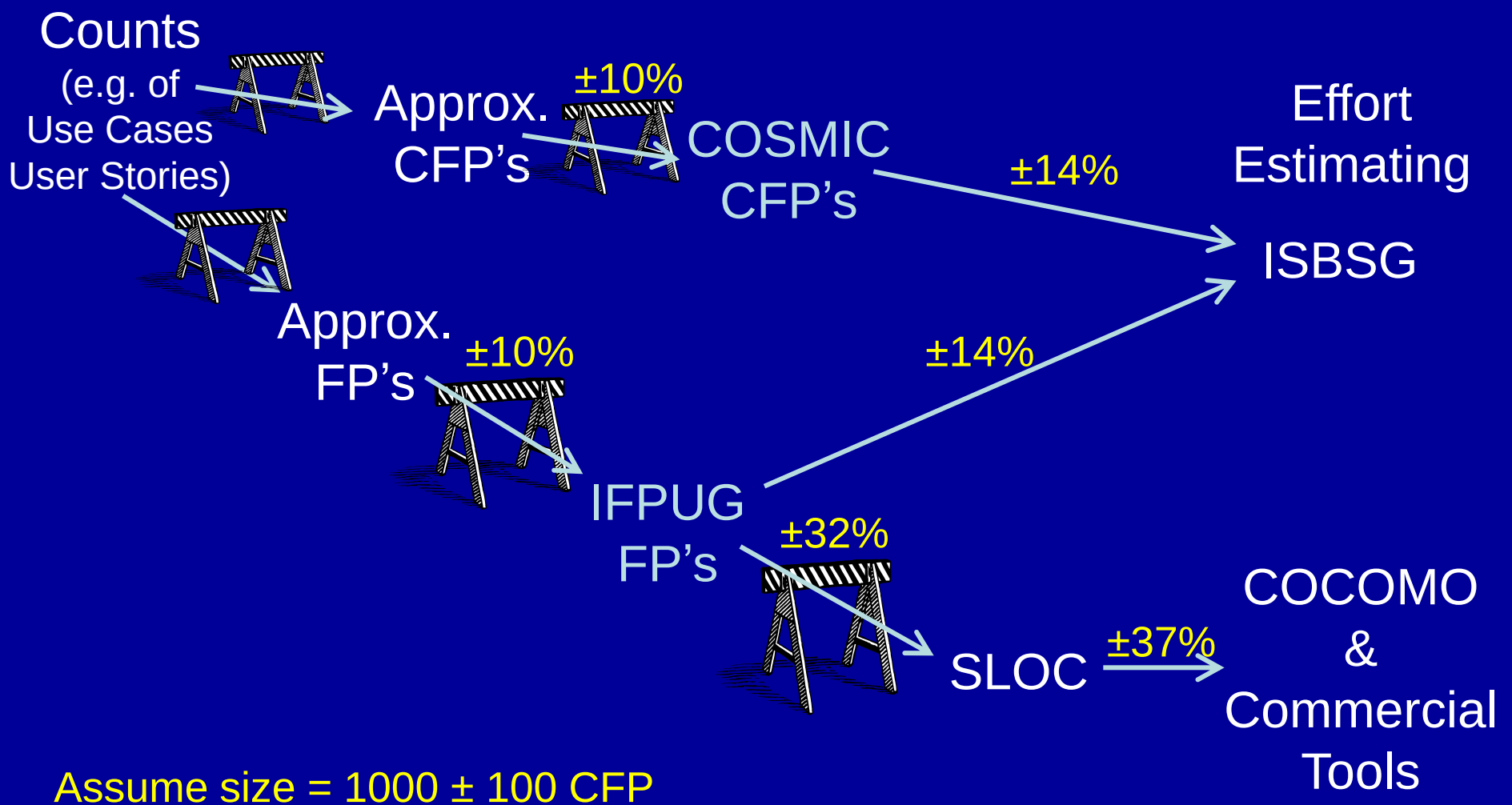
SLOC Sizes

- x Difficult to estimate from requirements
- x No reliable standards
- x Technology-dependent
- x A designer's view of size
- ✓ Implement all requirements, incl. NFR

COCOMO II Effort

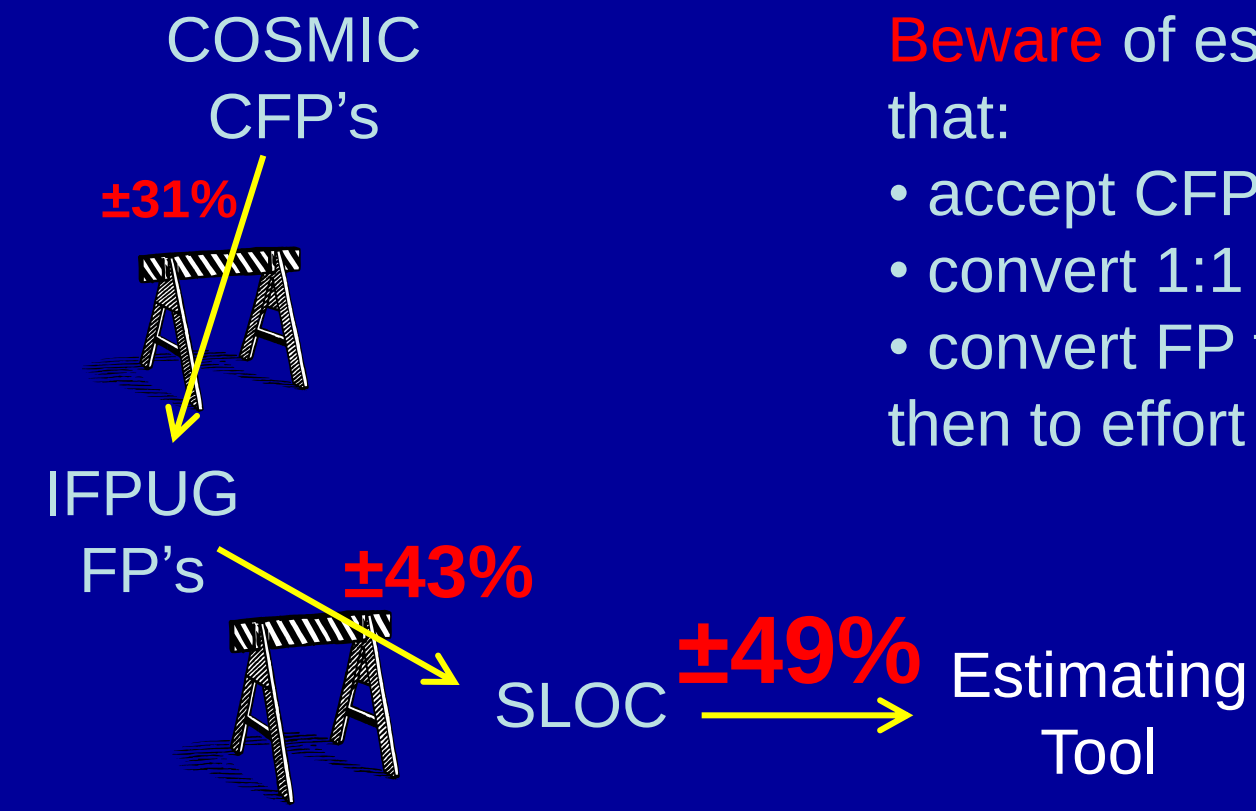
- x A complex model with up to 17 variables
- x Calibrated by expert judgment from a limited range of projects
- ✓ But 'open' and widely used with local calibration

Even best case assumptions show significant uncertainty on any one conversion



If your process involves multiple conversions, expect major error propagation

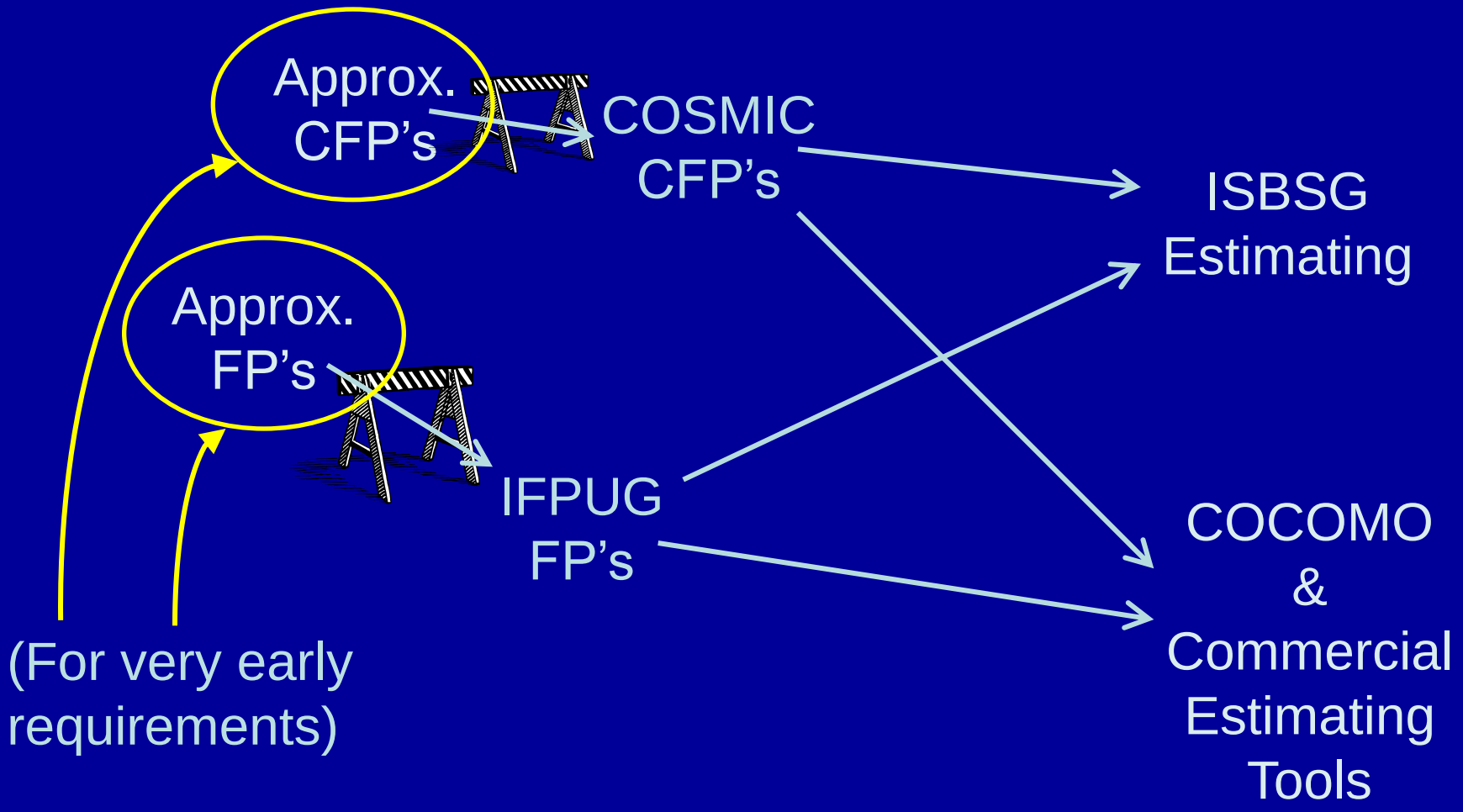
Assume 1000 ± 100 CFP



Beware of estimating tools that:

- accept CFP sizes as input
- convert 1:1 to FP
- convert FP to SLOC and then to effort

Conclusion: estimating methods should be calibrated directly with functional sizes



This is how the best results are obtained

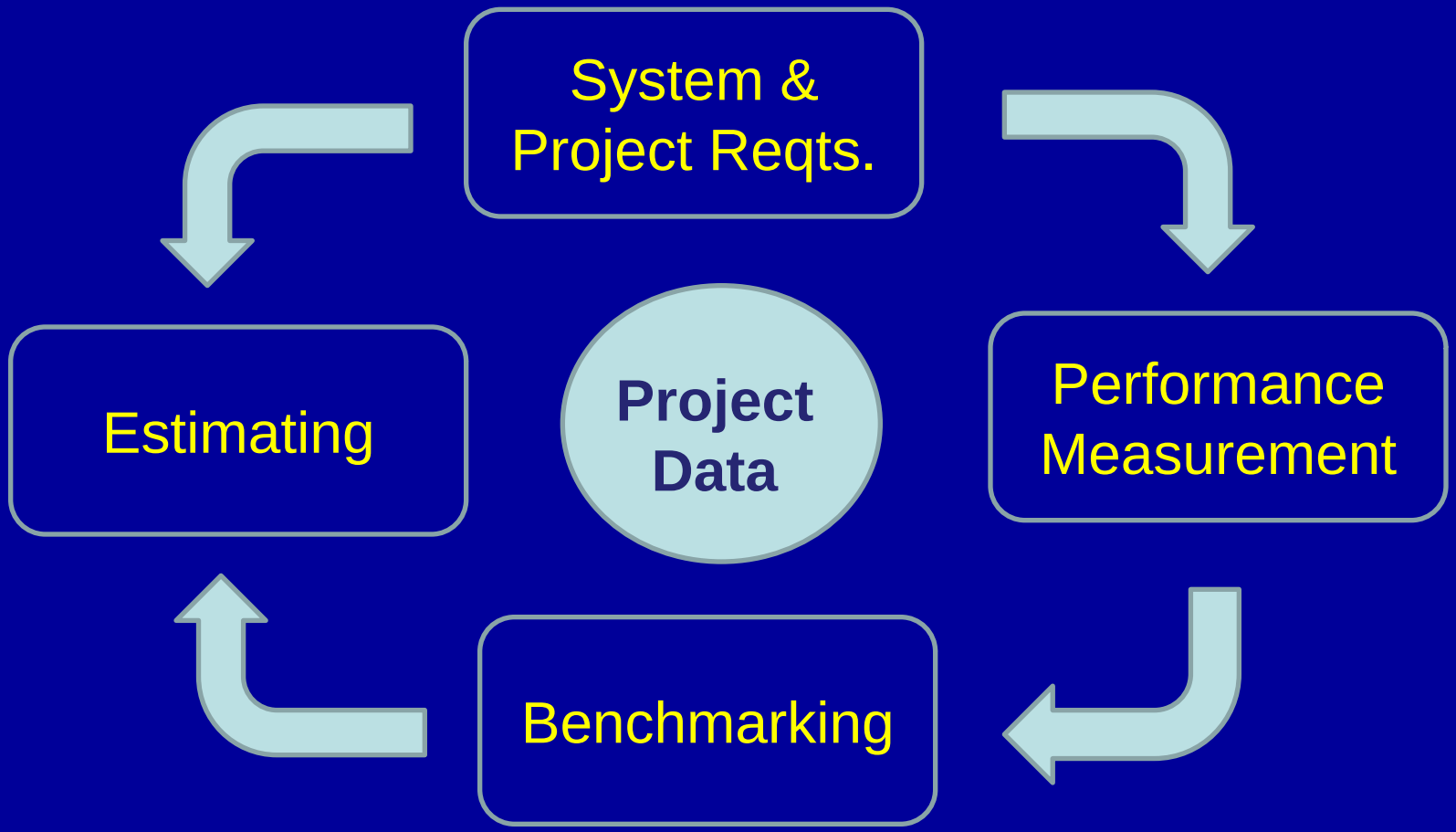
Conclusions from this section

- Do not rely on CFP:FP or on FP:SLOC conversion for estimating
- Segment the project types that recur frequently in your organization. Then, for each project type:
- Adapt your own very early sizing methods, e.g.
  - ‘counts’ of Use Cases (UCP) or User Stories (USP)
  - and/or approximate FSM methods
- Calibrate your estimating method or tool:
  - with your own data on software functional size, functional profile and other project attributes
  - using as few variables as possible to achieve the desired accuracy

Agenda

- Estimating: the issues
- How can we improve early estimation of project effort from requirements?
- What about Non-Functional Requirements?
- Conclusions for Requirements Engineers

Whether we use our own or public data, we need consistent measures across four fields

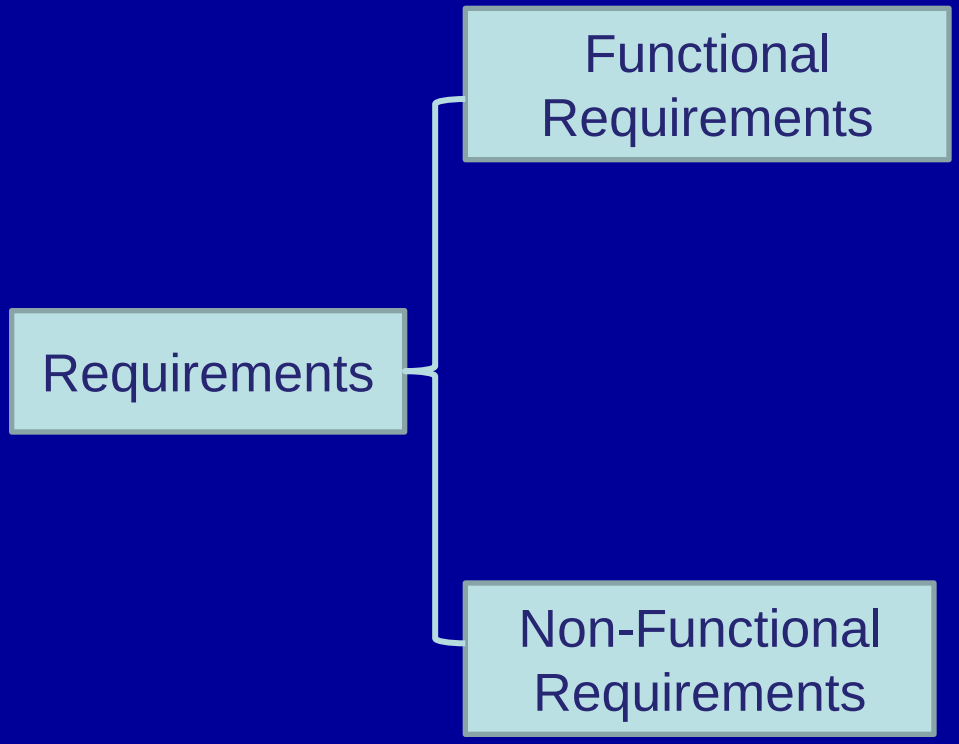


We can measure software size consistently across our four fields ....

.... but what about all the other data we could gather (> 100 possible variables)?

- Effort and schedule data (not easy!)
- NFR – whatever they are?

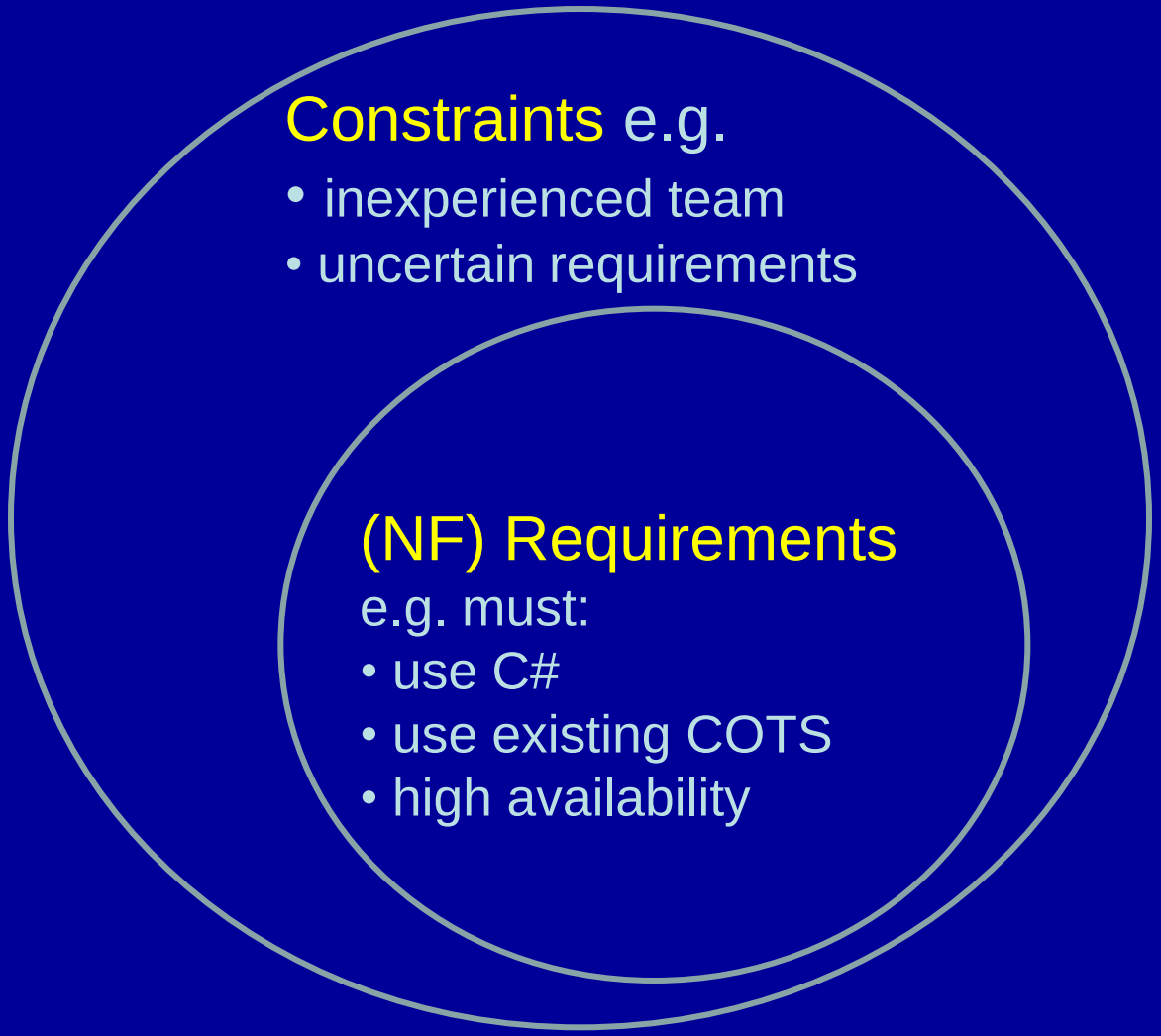
Let's start with a simple distinction between Functional (FR) and Non-Functional Requirements (NFR)



"What the software must do"

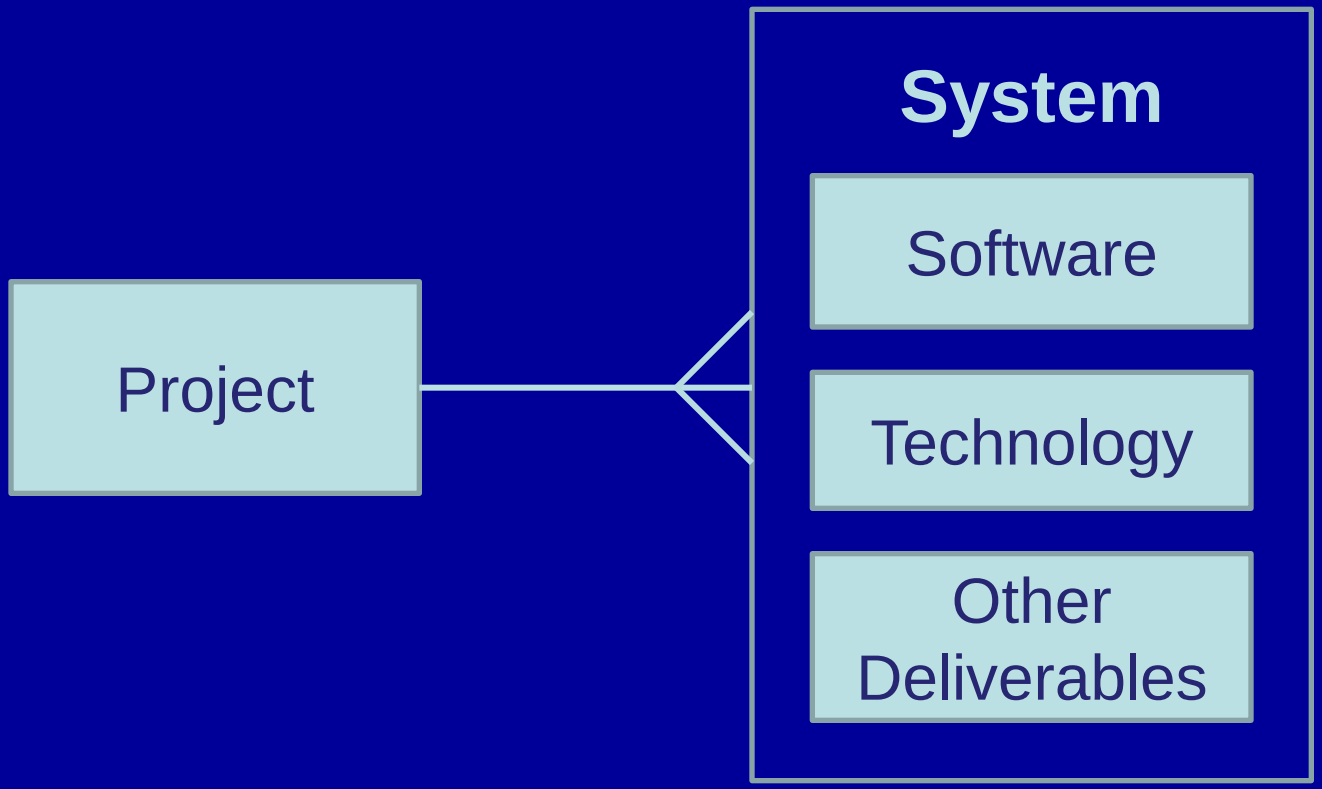
All other requirements on the system (and the project?)

We must consider NF ‘constraints’ as well as ‘requirements’



We must record NFR and constraints to help interpret performance measures & benchmark data, and for estimation purposes

Also, we need to distinguish what the ‘requirements’ (incl. constraints) apply to



Requirements can apply to any of these things

NFR vary enormously in importance between different types of systems

Typical Business Application

- NFR do not vary much across many systems
- Many may cause the same % effort overhead for similar projects(?)

ISBSG Estimating OK?

vs.

Mission Critical System (e.g. air traffic control, trading systems, etc)

- "NFR can account for half the pages of a statement of requirements" [11]

Segment project types on NFR for more accurate estimates (cf COCOMO)

So what do we really mean by 'NFR'? ISO/IEC definitions[12] are really bad

Functional Requirement: *"A requirement that specifies a function that a system or system component must be able to perform"*

Function:: *"A task, action, or activity that must be accomplished to achieve a desired outcome"*

Non-functional requirement*: "A software requirement that describes not what the software will do but how the software will do it."*

Example: software performance requirements, software external interface requirements, software design constraints, and software quality attributes. Note: Non-functional requirements are sometimes difficult to test, so they are usually evaluated subjectively.

Wikipedia definitions are totally useless

"*Functional requirements define what a system is supposed to do whereas non-functional requirements define how a system is supposed to be*"

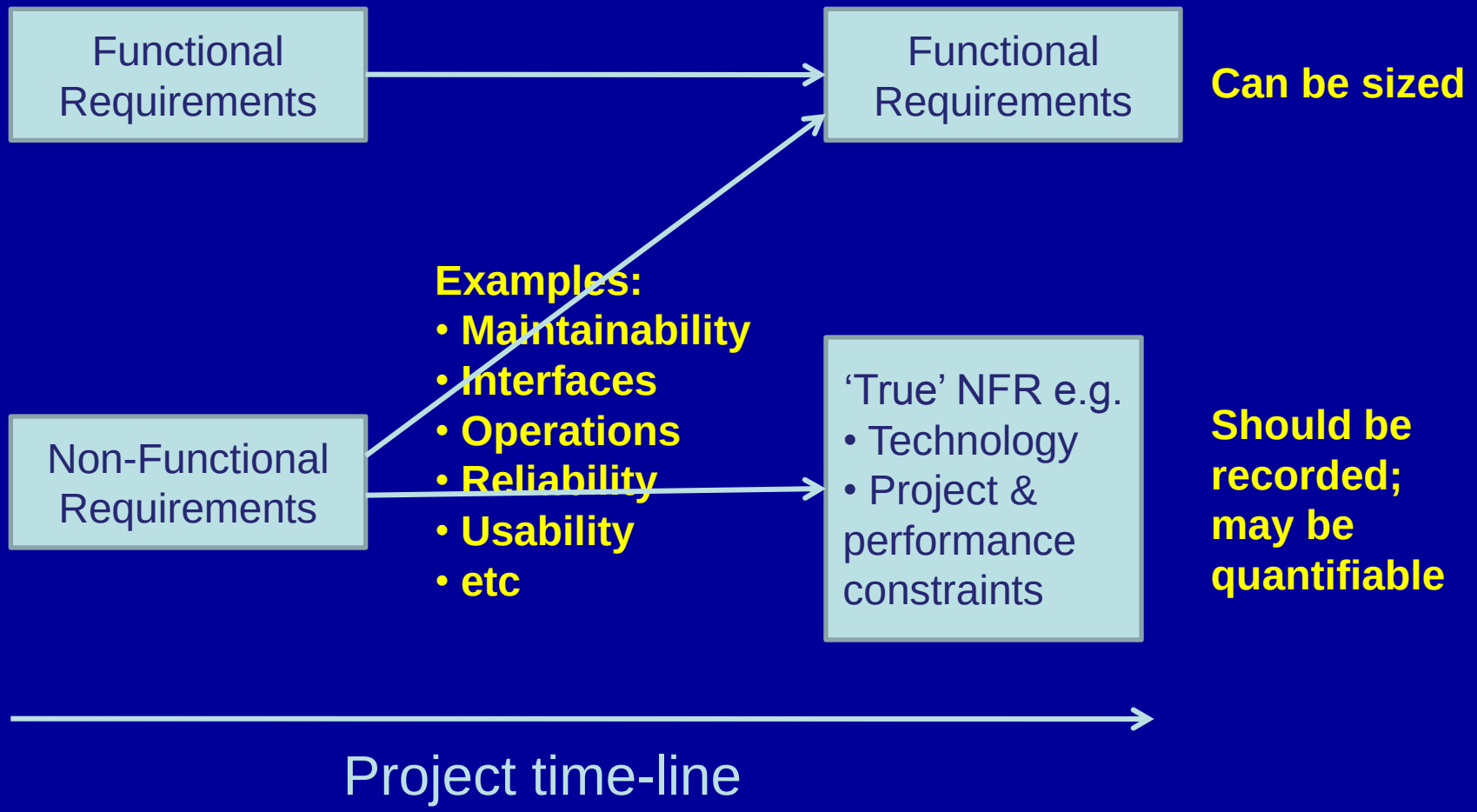
So let's take an example requirement: 'Security'

'The system is required to *ensure security* against unauthorised access" (Functional?)

or

"The system is required to *be secure* against unauthorized access" (Non-Functional?)

Many requirements that initially appear as NFR evolve to software FR[12]

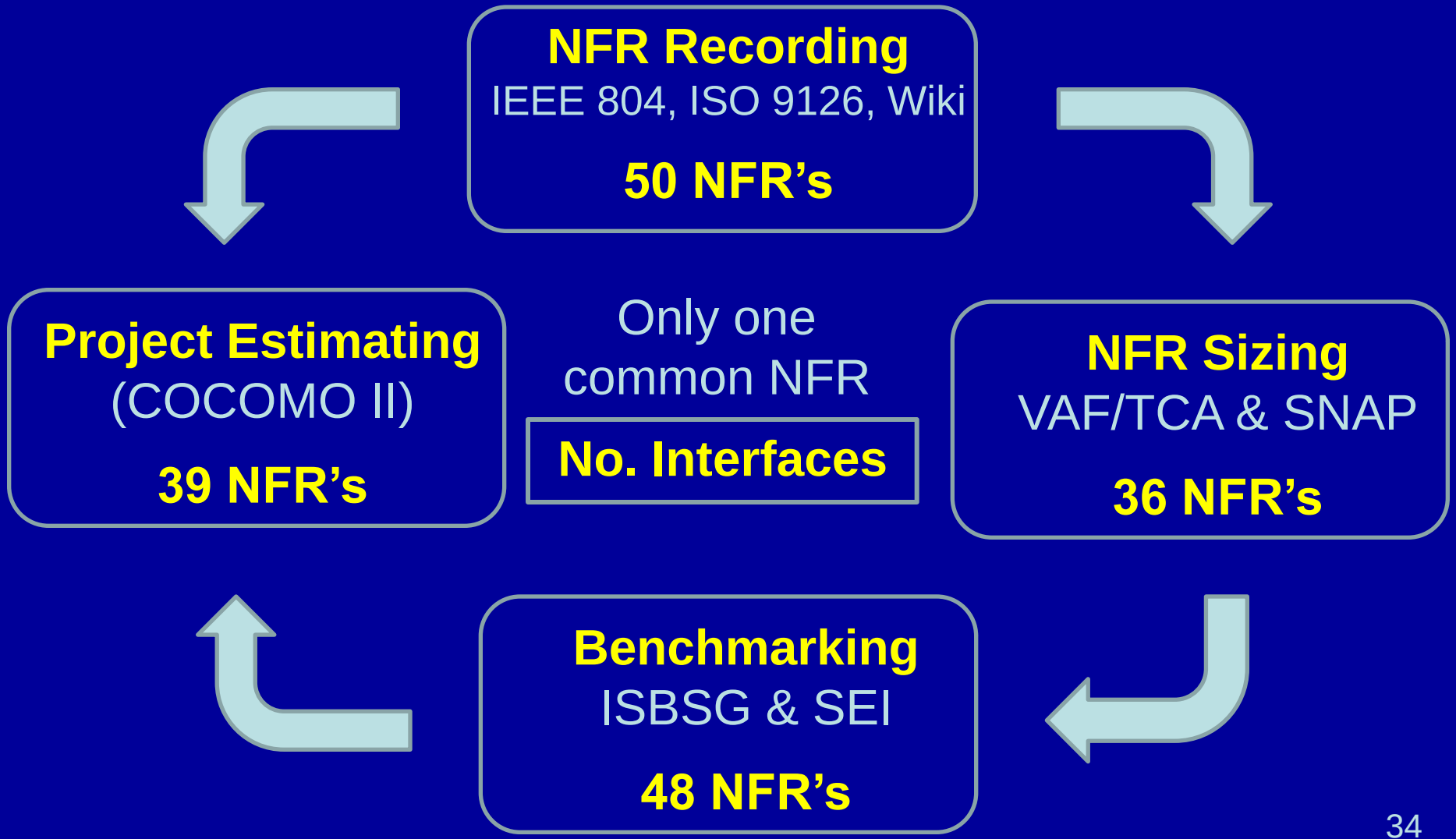


Abran/Al Sarayreh's findings are endorsed by Butcher [11] for Mission-Critical Systems

"Most Non-Functional Requirements evolve into requirements for software functionality and for hardware"

"I prefer to distinguish 'direct' and 'indirect' functional requirements"

I found 108 possible types of NFR. Their usage is grossly inconsistent [14]



Summary so far
Industry understanding of NFR is chaotic

- There is no clear *accepted* definition of NFR.
  ISO/IEC definitions are actually harmful
- Many NFR that are important for estimating evolve into functional requirements as a project progresses
- Approaches to performance measurement, benchmarking and estimating use largely different sets of NFR and constraints

Now some ideas on the way forward - real WIP!

Assume the COSMIC definition for a 'Non-Functional Requirement'

NFR = 'Any requirement for or constraint

- *on a hardware/software system*
- *or on a project* to develop or maintain such a system,*

except:

- *a functional user requirement** for software*
- *or a requirement that evolves into a functional user requirement for software'*

*If you prefer to define 'project requirements' separately from NFR, that's OK by us

** 'Functional user requirements' is quite well defined in ISO/IEC 14143/1

Divide 'traditional' NFR into 'Quasi NFR' and 'True NFR"

Quasi NFR (34): requirements that may evolve wholly or partly into FUR e.g. usability, interfaces, security

True NFR: requirements ... that cannot be implemented as software functions

- Software Constraints (10): e.g. must use C#, execute in batch mode
- Technology Constraints (16), e.g. must run on Unix, use data communications
- System Constraints(9), e.g. response time, no. of users
- 'Project Constraints (35)', e.g. budget ≤ $1M, multi-site teams
- 'Other (System) Deliverables (4)', e.g. documentation, training

Measure their FS

Record and take into account in software project performance measurement, benchmarking, & estimating

Treat separately

Rationalise NFR to a manageable set that can be recorded with performance measurements for their interpretation and use

Example: Only 9 of 34 Quasi NFR's need be recorded

No./Type	Examples	Proposed Action
16 x Common	Usability, Reporting	Can be 100% accounted for in the FS; no need to record?
9 x Common	Availability, interfaces	Measure the FS and record on a nominal or ratio scale
2 x common for all systems	Auditability	Ignore (common overhead)
2 x very uncommon	Emotional factors	Ignore
5 x Synonyms or sub-types	Reliability, resilience	Account for in the above

The biggest need: reconcile data recording for benchmarking with data needed for estimating

Data needed for estimating / not considered in benchmarking (?)

- Project Risk
- Project resource or time constraints

Possible solution for benchmark data?

- A ‘Relative Risk’ Index’?
- Minimum: record the constraints

  Ideal: a means of analysing project data to quantify the effect of such constraints

... And do not record data for benchmarking studies that is never used for interpretation, nor for estimating

The ‘still-to-do’ list

- Refine and define the taxonomy of NFR’s and measurement scales where possible
- Test with a range of experts
- Publish and publicise
- Promote the ideas to suppliers of benchmarking services and estimating methods

Please contact Chris Woodward of the UK Software Metrics Association if you would like to help with this approach chris.woodward@btinternet.com

Agenda

- Estimating: the issues
- How can we improve the early estimation of project effort from requirements?
- What about Non-Functional Requirements?
- Conclusions for Requirements Engineers

Conclusion: early estimating is enormously important. It must be improved

Obvious areas for improvement:

- Recognise the real problems of conversion between size measurements and of error propagation in estimating
- Improve the consistency of data collected for performance measurement and benchmarking, and data needed for estimating
- NFR must be better defined and understood

Requirements Engineers can do a lot to support better estimating

- Distinguish all requirements (‘direct’ & ‘indirect’) that can be implemented in software
- Document these requirements to facilitate functional size measurement (first, understand FSM!)
- Support the development of a standard taxonomy of ‘true’ NFR’s
- Record software, hardware, system and project ‘true’ NFR’s using a standard taxonomy

The framework presented here is still work-in-progress; there is a lot to do!

Final remark: measurement is not an overhead

As this is a ‘Requirements Engineering for Software Quality’ conference, please consider:

- Measurement is intrinsic to any engineering discipline
- Functional size measurement provides a quality control on the requirements

If software requirements cannot be measured with some confidence, you cannot begin to estimate the project or build the software reliably

Thank you for your attention

Charles Symons

cr.symons@btinternet.com

www.cosmicon.com

Cigdem Gencel

cigdem.gencel@unibz.it

References (1)

1. Standish CHAOS Report, 2009, www.standishgroup.com/newsroom/chaos_2009.php
2. McManus, J. and Wood-Harper, T., 'A Study in Project Failure', www.bcs.org June 2008
3. Whitfield, D., 'Cost over-runs, delays and terminations: 105 outsourced public sector ICT projects', European Services Strategy Unit, Research Report No. 3, December 2007
4. 'Are We Really That Bad? A look at software estimation accuracy Hill, P., ISBSG, CAI webinar, February 2013, www.ITMPI.org/library
5.'Why your IT project may be riskier than you think' Bent Flyvbjerg, Alexander Budzier, Harvard Business Review, September 2011
6. 'A review of studies on expert estimation of software development effort', Jørgensen, M., The Journal of Systems & Software 70 (2004), 37 – 60.
7. 'Practical Software Project Estimation', the International Software Benchmarking Standards Group, Peter Hill (Ed.), McGraw Hill, ISBN 978-0-07-171791-5, 2011
8. 'Software Cost Estimation with COCOMO II', Boehm, B. et al, Englewood Cliffs, NJ:Prentice-Hall, 2000. ISBN 0-13-026692-2
9. Santillo, L., 'Error Propagation in Software Measurement and Estimation', 16th International Workshop on Software Measurement, Potsdam, Germany, 2006

(continued)

References (2)

10. Gencel, C., Heldal, R., Lind, K.: On the Relationship between Different Size Measures in the Software Life Cycle. Asia-Pacific Software Engineering Conference (APSEC 2009), p.19-26, Malaysia, December 2009

11. Butcher, C., 'Delivering Mission-Critical Systems', BCS Central London Branch meeting 18th November 2010

12. *ISO/IEC. 24765:2009. 'Systems and Software Engineering Vocabulary'

13. (Example paper) Al-Sarayreh, K.T. and Abran, 'A. Specification and Measurement of System Configuration Non Functional Requirements, 20th International Workshop on Software Measurement (IWSM 2010), Stuttgart, Germany, 2010

14. Symons, C., 'Accounting for Non-Functional Requirements in Productivity Measurement, Benchmarking & Estimating', UKSMA/COSMIC International Conference on Software Metrics & Estimating, London, October 2011 , www.uksma.co.uk

Information on the COSMIC method and its uses is available for free download from www.cosmicon.com