Accounting for Non-Functional Requirements in Productivity Measurement, Benchmarking & Estimating

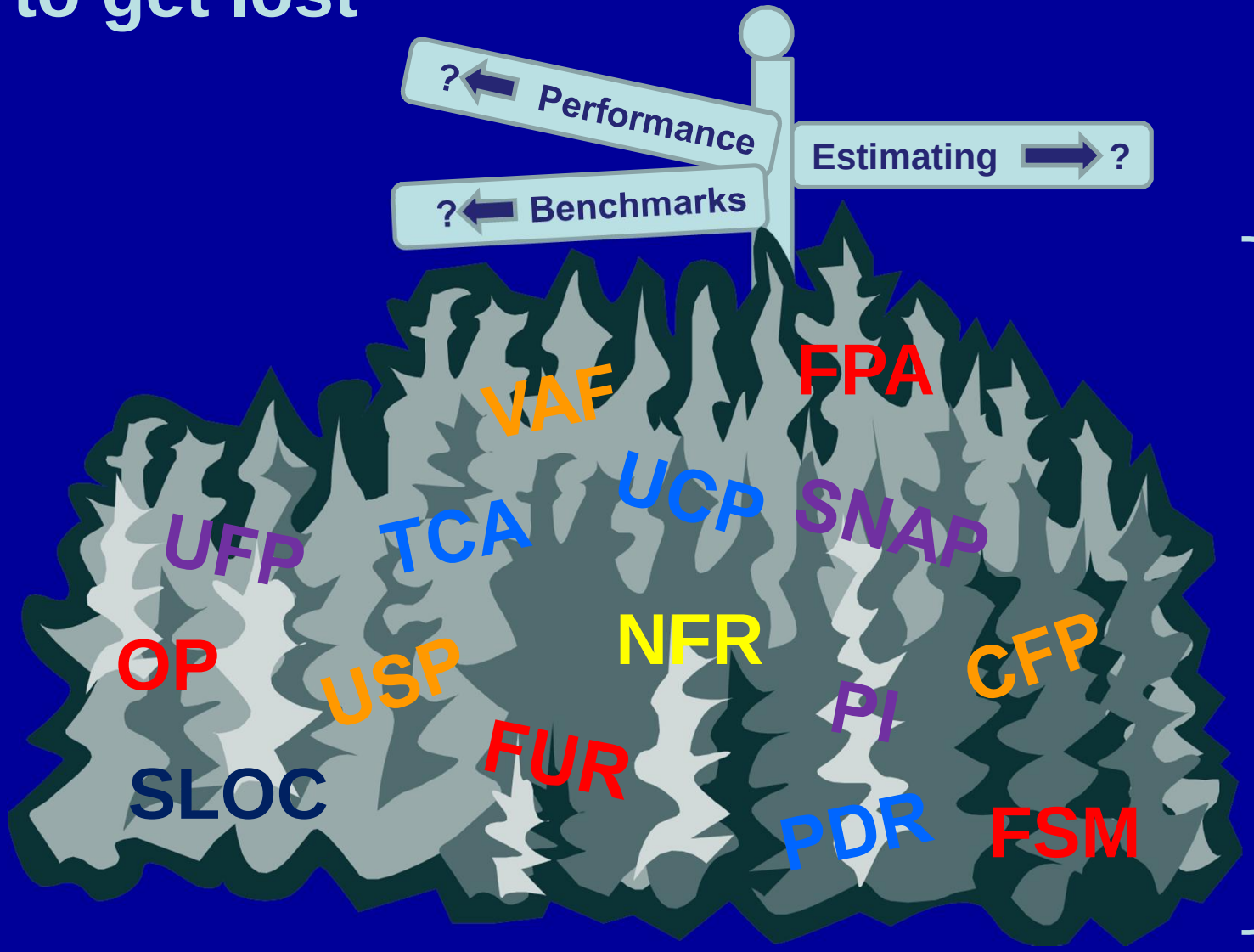
Charles Symons

President

The Common Software Measurement International Consortium

UKSMA/COSMIC International Conference on Software Metrics & Estimating, London, October 27/28th, 2011

We have created a forest in which it's easy to get lost



Project Size?

or

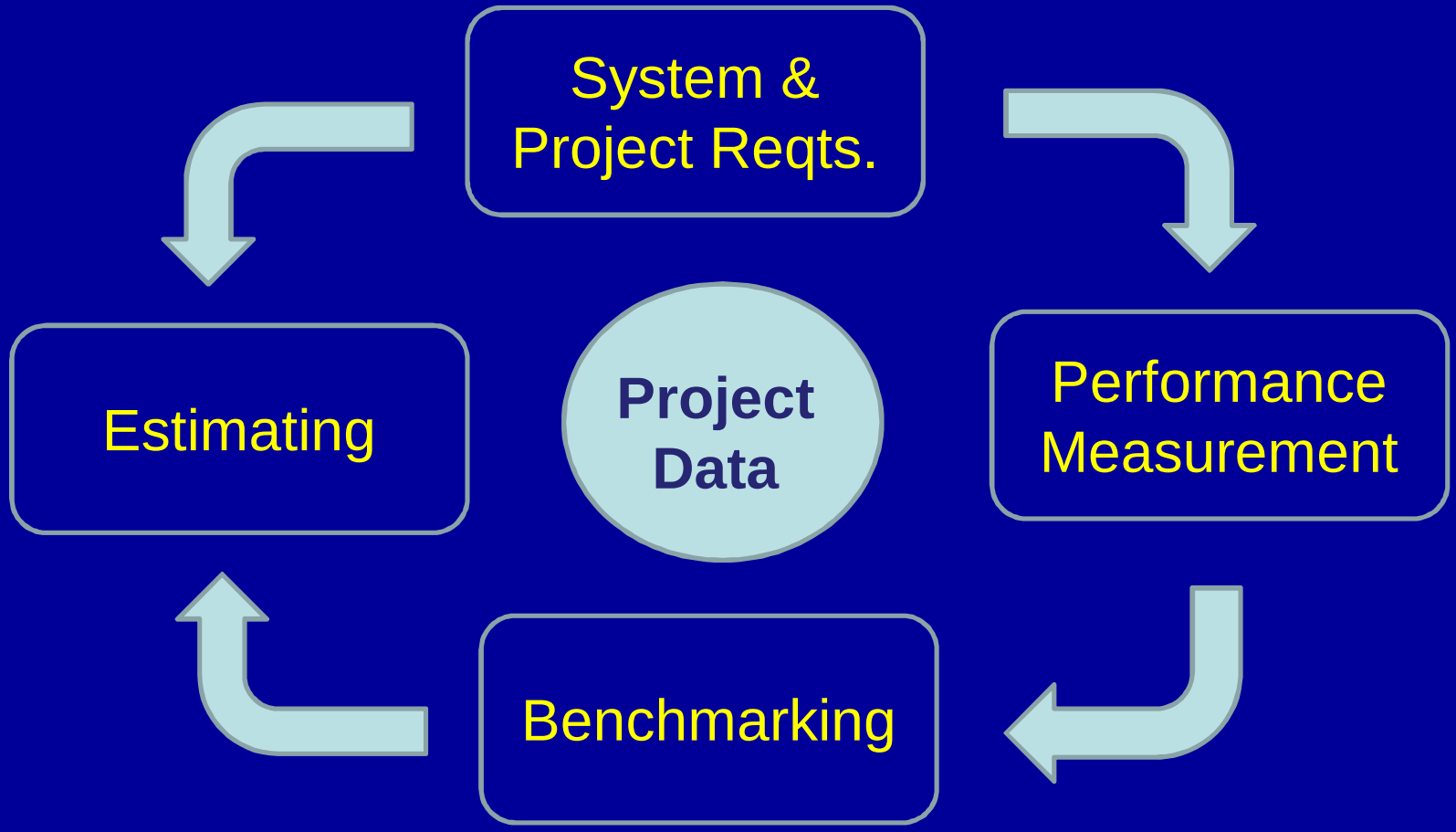
Software Size?

IFPUG NESMA COSMIC FiSMA MkII

Agenda

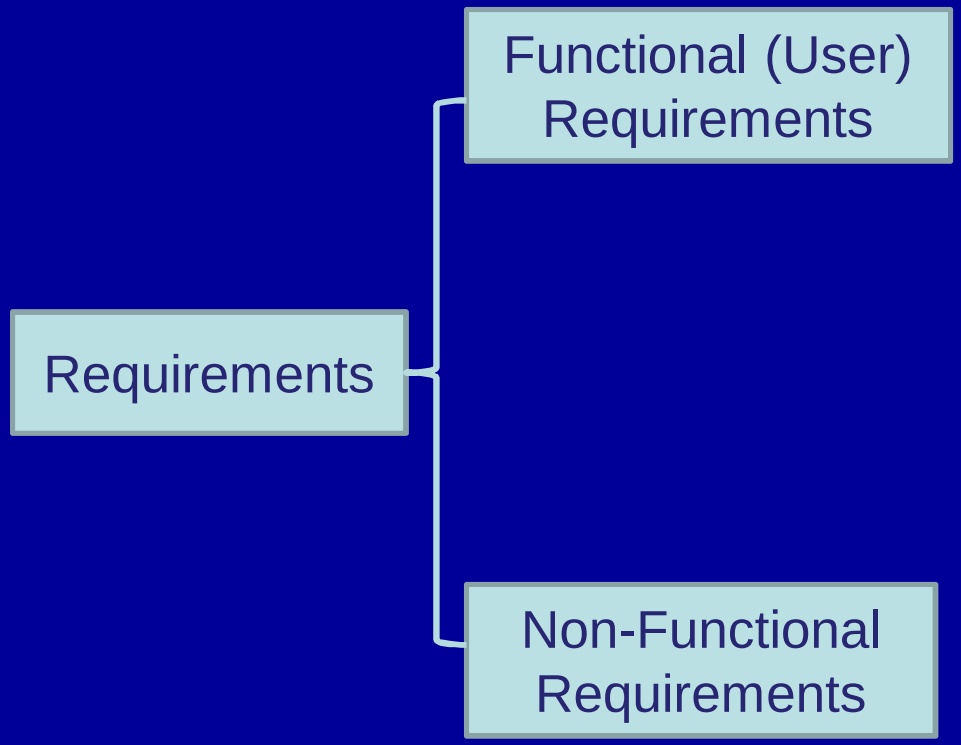
- Where are we in the forest?
- Some helpful directions
- The way out of the forest?
- Conclusions

Our subjects should be closely inter-related



For all of these we need consistent measures of ‘product size’ and many other parameters

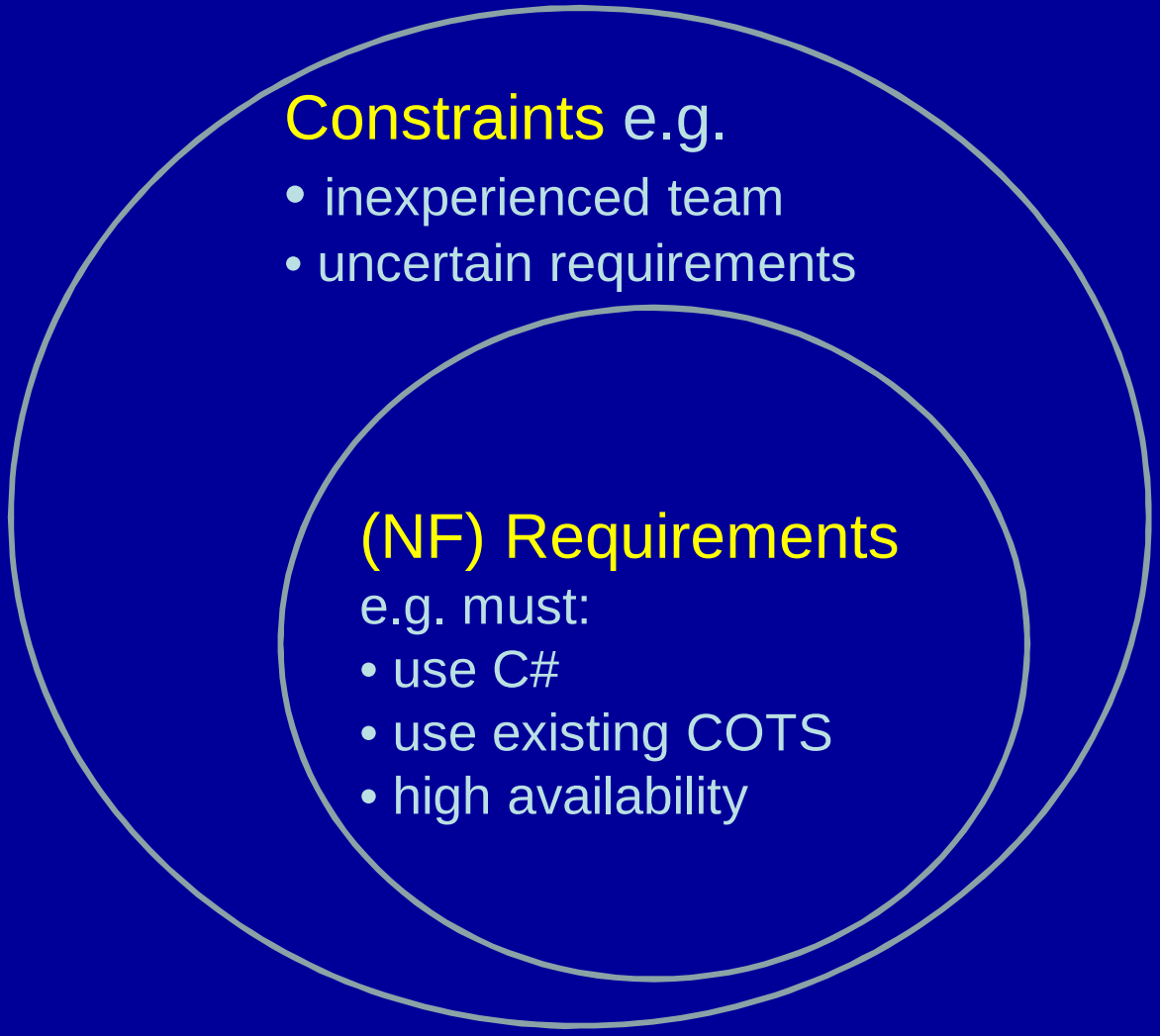
We'll start with a simple distinction between Functional (FUR) and Non-Functional Requirements (NFR)



"What the software must do"

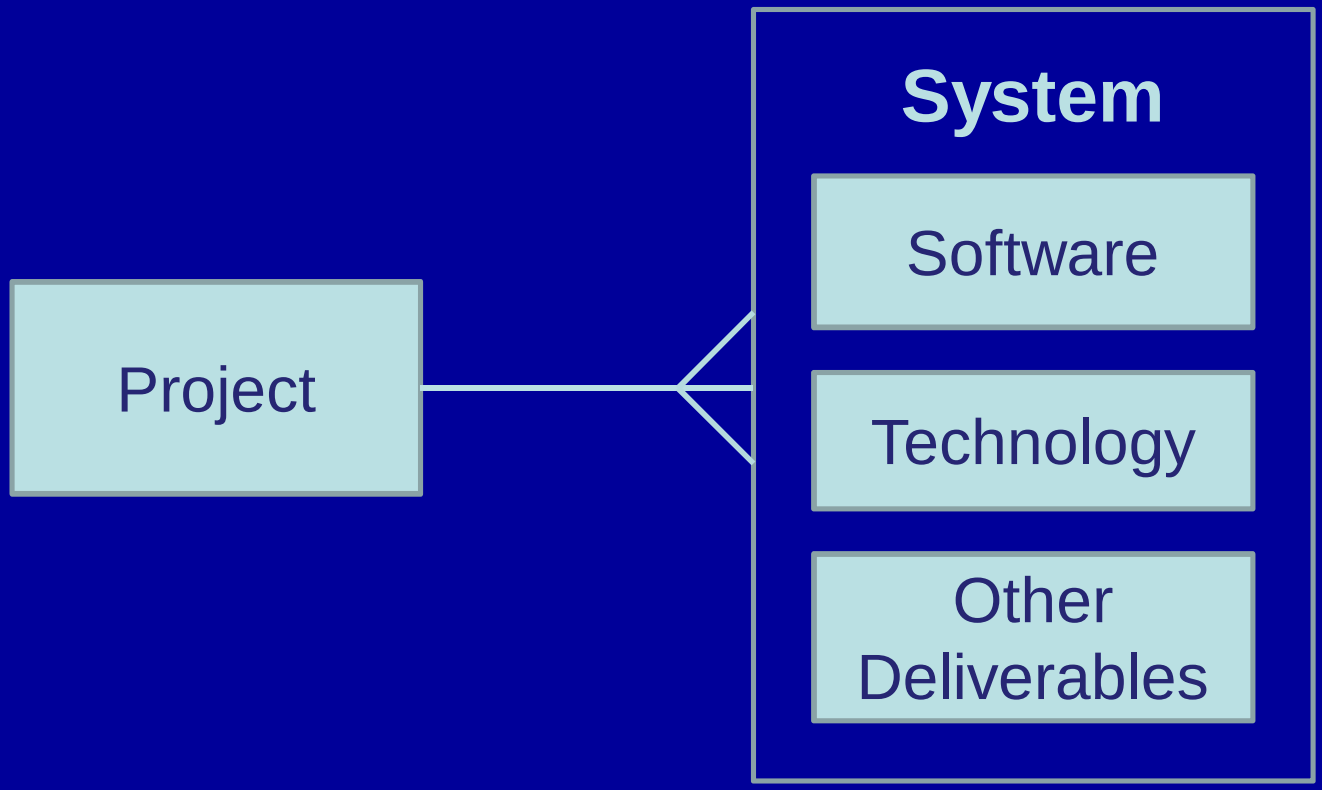
All other requirements on the system (and the project?)

We must consider NF ‘constraints’, as well as ‘requirements’, to achieve our goal



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

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



Requirements can apply to any of these things

Assume we want to measure productivity of **software** development

$$\text{Development Productivity} = \frac{\text{Work Output}}{\text{Work Input}}$$

$$= \frac{\text{software size}}{\text{project effort}}$$

of all requirements related to software

for the effort related to software (only)

Note: A count of SLOC is a size measure that reflects all software requirements - but is not an easy measure to use

Albrecht invented ‘FPA’, aiming to account for all requirements that affect effort

Size (FP) = ‘Unadjusted FP’ x ‘Value Adjustment Factor’

A measure of ‘functional size’

Accounts for 14* ‘General System Characteristics’

Was this the origin of the FR vs NFR distinction?

* Data communications, Distributed data processing, Performance, Heavily-used configuration, Transaction rate, On-line data entry, End user efficiency, On-line update, Complex processing, Reusability, Installation ease, Operational ease, Multiple sites, Facilitate change

I developed ‘MkII FPA’, mainly to improve the UFP size measurement

Size (MkII FP) = ‘Unadjusted MkII FP’ x ‘Technical Complexity Adjustment’

A measure of ‘functional size’

Accounts for 19+* ‘General Application Characteristics’

* Albrecht’s 14 GSC’s, plus Interfaces, Security, Third party direct access, User Training facilities, Documentation ... and locally-defined characteristics

Then an ISO Working Group defined the principles of Functional Size Measurement in ISO/IEC 14143/1

Functional Size

"a size of the software derived by quantifying the Functional User Requirements"

Functional User Requirements (FUR)

"a sub-set of the User Requirements .... that describe what the software shall do, in terms of tasks and services" .....

*"Examples of User Requirements that are **not** FUR include but are not limited to: quality, organizational, environmental and implementation constraints"*

The ISO/IEC 14143/1 standard helped kill off the VAF & TCA

The VAF & TCA

- can never be calibrated properly, except using effort, which obviously varies hugely over different types of software project
- mostly had only two values (~ 0.8 or 1.2)
- are now obsolete, meaningless numbers
- are ignored in:
  - International Standards for the IFPUG, MkII, NESMA, etc FSM Methods
  - ISBSG benchmark data
  - Conversion of UFP to SLOC, e.g. in COCOMO II

But IFPUG again aims to measure NFR's by a 'Software Non-functional Assessment Process' (SNAP)*

Main objective: ... *"to ensure the framework can be 'used to establish a link between non-functional size and the effort to provide the non-functional requirements"*, ... in order to:

- *"better plan, schedule and estimate projects*
- *identify areas of process improvement*
- *assist in determining future technical strategies ...(etc)*
- *assist in communicating non-functional issues to various audiences"*

* Software non-functional assessment process (SNAP): Assessment practices manual, release 1.0', IFPUG , September 2011

So what do we really mean by ‘NFR’s? ISO/IEC* definitions are seriously 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.

* ISO/IEC. 24765:2009. ‘Systems and Software Engineering Vocabulary’

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*"

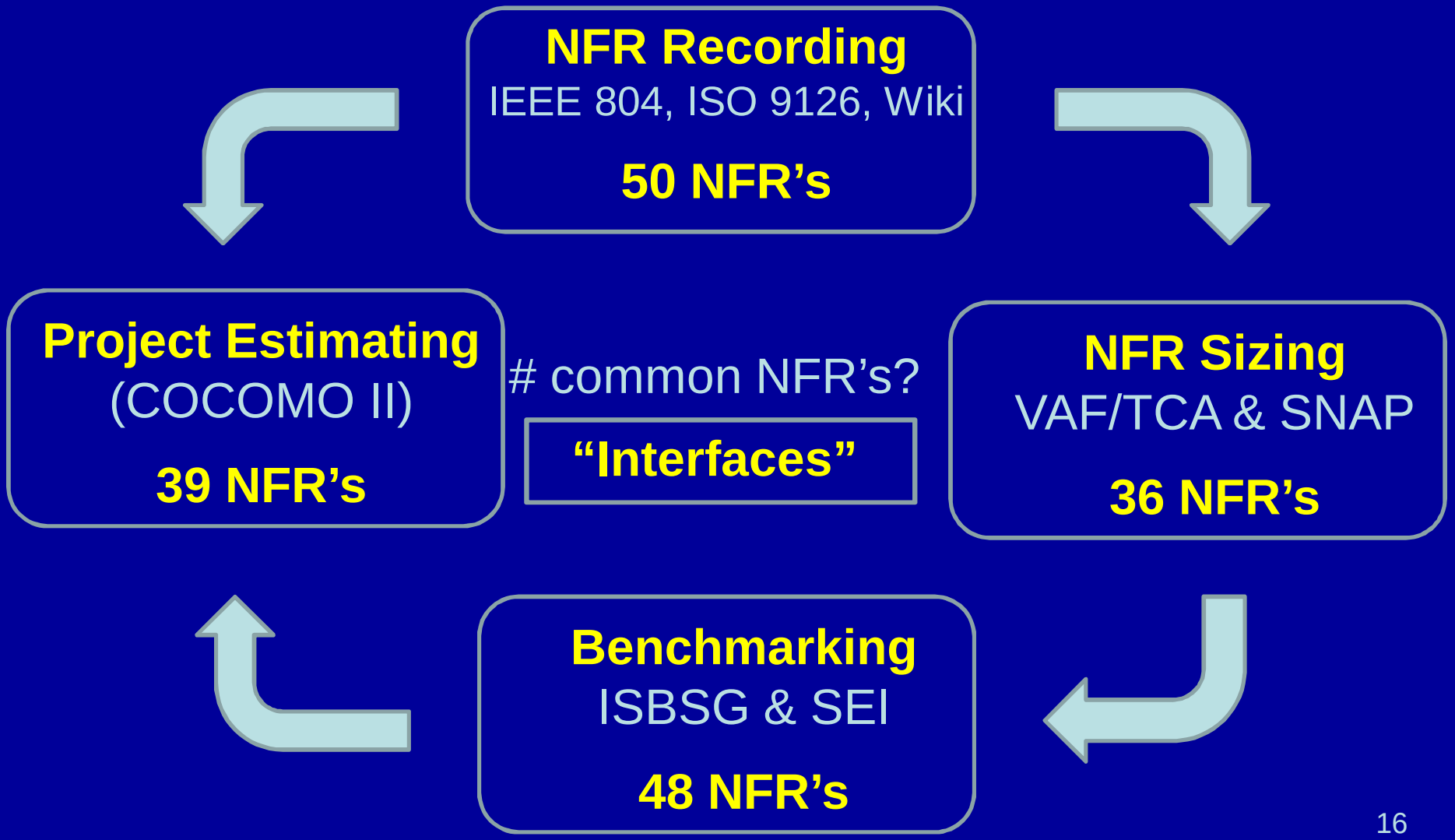
So let's take an example requirement: 'Security'

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

or

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

I found 108 possible types of NFR, but their usage is grossly inconsistent



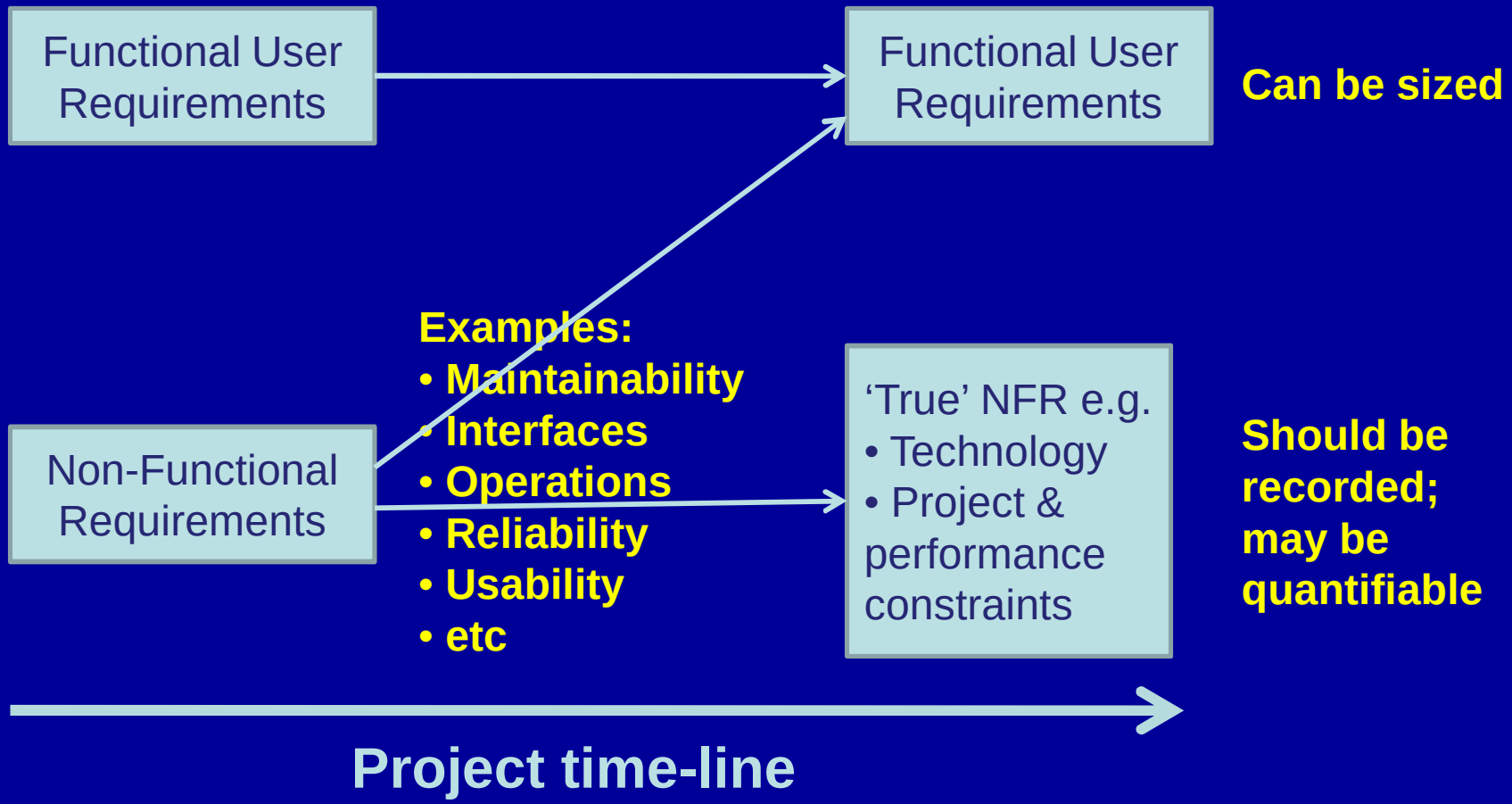
Summary so far

- There is no clear definition of NFR: ISO/IEC definitions are actually harmful
- NFR that are important, e.g. for estimating, evolve as technology develops
- Measurement of NFR is discredited by the VAF/TCA experience. SNAP will fail similarly
- Approaches to performance measurement, benchmarking and estimating use largely different sets of NFR and constraints

Agenda

- Where are we in the forest?
- Some helpful directions
- The way out of the forest?
- Conclusions

As a project progresses, requirements that initially appear as NFR result in software FUR*



* (e.g.) Al-Sarayreh, K.T. and A. Abran, *Specification and Measurement of System Configuration Non Functional Requirements, 20th International Workshop on Software Measurement (IWSM 2010), Stuttgart, Germany, 2010*

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(?)

vs.

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

- "NFR can account for half the pages of a statement of requirements" *

* Colin Butcher, 'Delivering Mission-Critical Systems', BCS Central London Branch meeting 18th November 2010

Abran/Al Sarayreh's findings are endorsed by Butcher* 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"

* Colin Butcher, 'Delivering Mission-Critical Systems', BCS Central London Branch meeting 18th November 2010

Eberhard Rudolph lamented in WG/12: '*we need to define functional' (as in 'FUR')*

So let's try: $y = f(x)$

where f (function) = transformation

Output = f (Input)

where 'f' = the processing rules

Then a 'Software Function' becomes:

"A process that transforms Input to Output, where:

- *Input = data that is entered into the process from users or from storage during execution*
- *Output = data that is sent out to users or to storage from the software during execution"*

SEVOCAB [1)] lists nine definitions of ‘function’ – of which one supports my view

The one clear definition:

“A transformation of inputs to outputs, by means of some mechanisms, and subject to certain controls, that is identified by a function name and modelled by a box” [2)]

1) Software and Systems Engineering Vocabulary; Definitions from ISO/IEC and IEEE standards
2) IEEE 1320.1-1998 (R2004) IEEE Standard for Functional Modeling Language - Syntax and Semantics for IDEF0

Now we can precisely define a ‘Non-Functional Requirement’

NFR = ‘any requirement for or constraint on a hardware/software system, or on the project to develop, enhance or maintain it, that cannot be implemented as software functionality**’*

(*If you prefer to define ‘project requirements’ separately from NFR, that’s OK by me)

** ‘Software functionality ‘ = all or any part of a set of software functions

Adopting this definition, many requirements thought of as NFR are actually implemented wholly or partly as functions ....

	Total no. of Parameters	No. of Parameters implemented as functions
VAF	14	14
TCA	19	17

(.... and incidentally IFPUG 'Data Functions' are not 'functions')

Even IFPUG's SNAP scale measures most NFR by their impact on functions

An NFR is assessed in terms of its impact on 16 factors:

- **Data operations:** Data entry validation, Logical & mathematical operations, Data formatting, Internal data movements, Delivering functionality by data configuration
- **Interface design:** UI changes, Help methods, Multiple input methods, Multiple output methods
- **Technical Environment:** Multiple platforms, Database technology, System configuration, Batch processing system
- **Architecture:** Mission critical (real-time system), Component based software, Design complexity.

For 12 of the 16 factors, an NFR is assessed by considering its impact on elementary processes

(SNAP does not define any NFR types)

Conclusions so far

- Almost all ISO/IEC definitions of FUR, function, and NFR are vague and unhelpful
- Having defined a software ‘function’ precisely, we can then define a NFR precisely
- Three independent sources confirm that much of what is initially stated as NFR evolves into software FUR
  - the work of Al-Sarayreh & Abran on space systems NFR,
  - The experience of Butcher from Mission Critical Systems
  - (Indirectly) the way the IFPUG SNAP process works
- This ‘indirect’ functionality should be measured by a standard FSM method
- A separate NFR size index is harmful

Agenda

- Where are we in the forest?
- Some helpful directions
- The way out of the forest?
- Conclusions

We can now 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

Measure their FS

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

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

- 'Other (System) Deliverables (4)', e.g. documentation, training

Treat separately

Consequences of this approach: 1. for performance measurement.

We need to define carefully:

'Development productivity' = $\frac{\text{Size of all software developed}}{\text{Effort on all software developed}}$

where 'Size' is the total size of:

all Functional User Requirements

\+ all Quasi NFR that are delivered as custom software

\+ all 'glue' software needed to integrate any COTS or re-used software

N.B.

Do not include the size of any COTS or re-used software

2. We need a FSM method that can measure application, infrastructure and glue software

COSMIC is the only FSM method designed to measure

- business and real-time applications
- infrastructure software (includes ‘glue’)

with rules for aggregating sizes of software:

- components
- in different layers

(If a FSM Method cannot measure a particular type of functional requirement, this does NOT mean the requirement is a NFR !)

3. We must 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. (WIP!)

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

Example: a possible 5-point nominal scale for recording* ‘required availability’

1 = Not critical at all

2 = Periodically critically important, e.g. overnight batch, month-end payroll

3 = ‘Normal’ in-house, on-line availability(expensive when unavailable, standard BU/R needed)

4 = Continuous availability is essential for business continuity and reputation (e.g. for public access or for process control)

5 = Mission-critical (e.g. 24/7). Lives or the economy at risk if failure

*** Like the VAF but NOT for measuring!**

4. The biggest need is to 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

Example: is it possible to devise a ‘Relative Risk Index’, applicable for all projects?

What matters is the risk for a project of not delivering on time and budget relative to the local ‘context’

- Example: lack of domain experience may be a big risk for one organization, but no risk for another:
- A Relative Risk Index would take into account:
  - Size of the software to be developed
  - Novelty for the project team (domain, software language, tools, etc)
  - Uncertainty of the requirements
  - Criticality of the budget, timescales or product quality
  - Etc?
- A 4-point scale should be adequate?

5. The ‘still-to-do’ list

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

Please contact me if you would like to help with this approach

Agenda

- Where are we in the forest?
- Some helpful directions
- The way out of the forest?
- Conclusions

Conclusions

- Do not try to construct artificial ‘size’ indices to measure constraints or NFR
- Measure the functional size of all requirements (‘direct’ & ‘indirect’) that can be implemented in software
- Record software, hardware, system and project ‘true’ NFR’s as measurements or on standard nominal scales
- The framework presented here is still work-in-progress, but there is a better way to account for NFR’s

Thank you for your attention

Charles Symons

cr.symons@btinternet.com

www.cosmicon.com

Economists measure 'Multifactor Productivity'

- (Wikipedia) 'Multifactor productivity measures reflect output per unit of some combined set of inputs'

$$\text{MF Productivity} = \frac{\text{Output}}{\text{KLEMS*}}$$

- Used at the macro-economy level, not at the project level. (What we measure is 'labour productivity')
- Sometimes dismissed as 'meaningless numbers'

* KLEMS = Capital, labour, energy, materials, services