A ‘SCATTER-GUN’ OR ‘RIFLE-SHOT’ APPROACH TO MANAGING AND ESTIMATING SOFTWARE PROCESSES?

IWSM-Mensura Conference

Beijing, September 2018

Charles Symons

Agenda

- Goals and terminology
- The challenges of estimating, measuring and controlling the performance of software processes
- The ‘Scatter-gun’ approach
- The ‘Rifle-shot’ approach
- Conclusions

Let us explore ….

…… how to measure and control the performance of software processes, and to estimate future processes:

using
external data

(and/or)

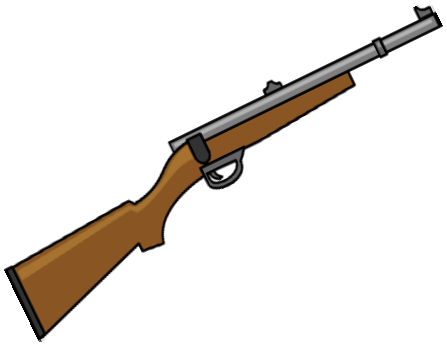
using
internal data

the **‘Scatter-gun’** approach

the **‘Rifle-shot’** approach

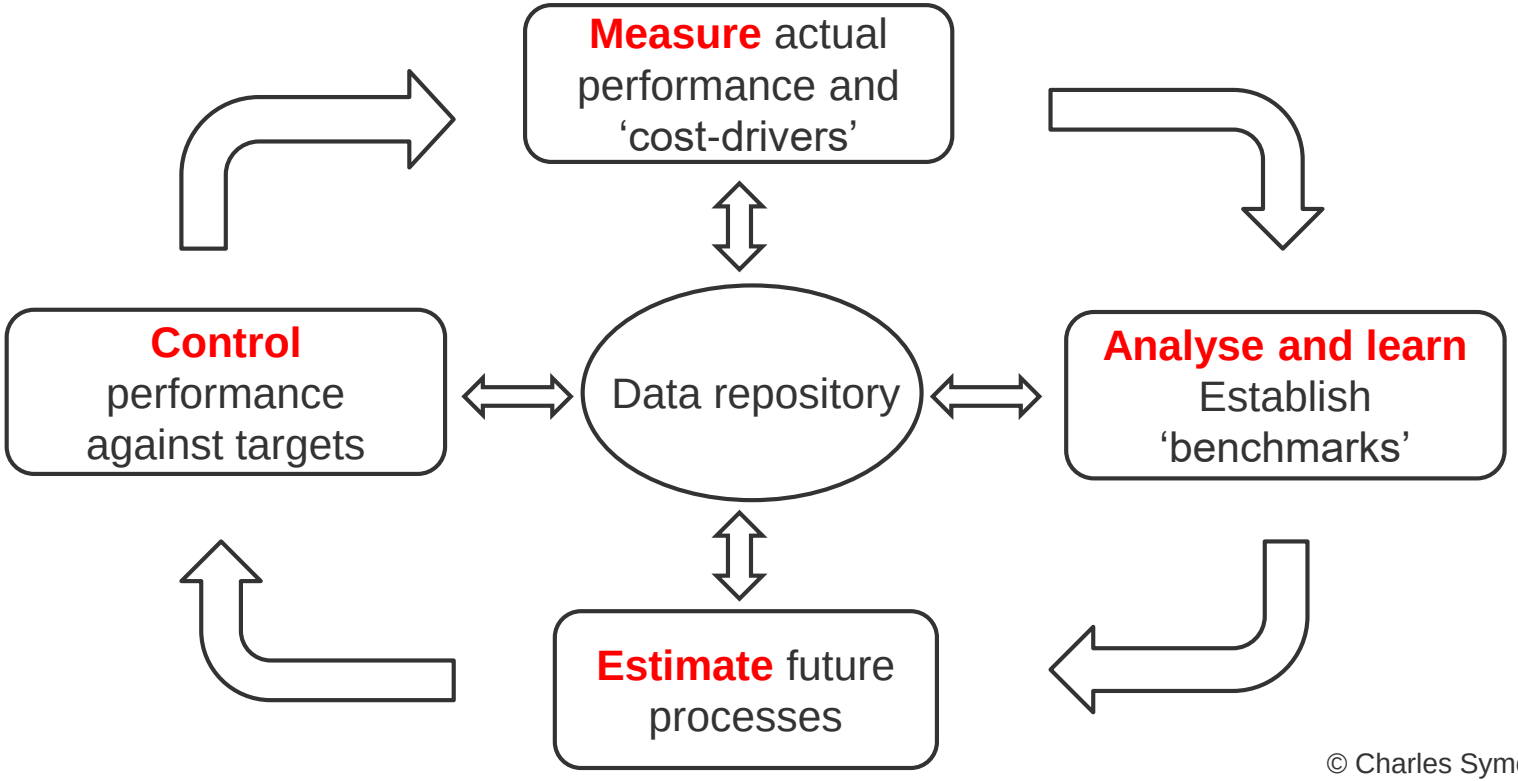
The analogy

The ‘**Scatter-gun**’



The ‘**Rifle-shot**’

Our goal: master the whole cycle of managing software processes with the aid of measurements



Example: using past productivity measurements to estimate effort for a new project

Completed projects:

$$\text{Measure productivity} = \frac{\text{Software size}}{\text{Project effort}}$$

Establish benchmark productivity values for each type of project

New project:

$$\text{'Typical' estimated effort} = \frac{\text{Estimated software size}}{\text{Benchmark project productivity}}$$

$$\text{'Best' estimated effort} = \left\{\frac{\text{Estimated software size}}{\text{Benchmark project productivity}}\right\} \text{x} \left\{\begin{array}{c}\text{Adjustments for}\\ \text{project-specific}\\ \text{'cost-drivers'}\end{array}\right\}$$

By ‘cost-drivers’ we really mean ‘performance-drivers’, excluding financial factors

‘Cost’ = (Performance-drivers) x (Financial factors)

Excluding financial factors:

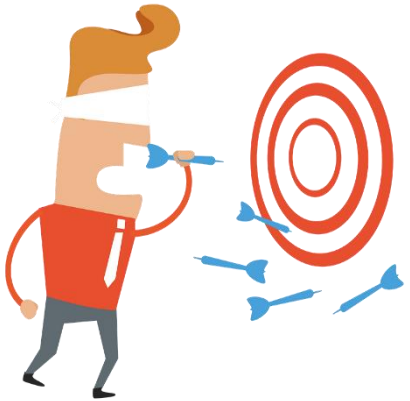
- People costs (salary, social costs, overheads, etc.)
- Technology costs (capital, maintenance, etc.)
- Exchange rates, accounting practices, etc.
- Benefits realization

Agenda

- Goals and terminology
- The challenges of estimating, measuring and controlling the performance of software processes
- The ‘Scatter-gun’ approach
- The ‘Rifle-shot’ approach
- Conclusions

1. Few organizations really master the control cycle for managing and estimating software processes

- High proportions of software project failures and cost over-runs
- Who does best?
  - Commercial software suppliers – a matter of survival
  - Agile teams may benefit from the rapid feedback cycle, but estimating is still poor



Why the problems? Developing and maintaining software is a partly unpredictable process

2. The performance of software processes can be measured in various ways, that are tradeable

Project achievement vs plan
- Actual vs. estimated: Effort, Duration, Size

Project speed
- Size / Duration

Project productivity
- Size / Effort

Product quality
- Defect density (# Defects/Size)
- Functional (e.g. business needs)
- Technical (e.g. maintainability, response time, etc.)

… and the performance of on-going **maintenance and enhancement** processes

3. Mastering the control cycle requires a good method for measuring software size

Size:

- Is a key component of performance measures,
- … and the biggest driver of effort and time,
- … and risk increases with size

Only ‘Functional Size Measurement’ methods can be used for the whole control cycle

- ✓ Technology-independent
- ✓ International standard methods
- ‘First Generation’ methods have limitations
- Manual measurement requires experience

4. There are very many possible cost-drivers ….

4. … and there are many different views on what are the most important cost-drivers

- The ISBSG collects data for a new development project via 33 questions on size and ~70 questions on other cost-drivers [1)]
- The ‘open’ COCOMO estimating model requires data on size and 22 cost-drivers [2)]
- A COSMIC/ISBSG study lists 42 Non-Functional requirements and 19 Project Requirements & Constraints [3)]
- Commercial estimating tools take account of very large numbers of cost-drivers [4)]

Summary: implementing the software control cycle faces many inherent challenges

- Software processes are part-routine, *part-unpredictable*
- The performance of software processes has multiple, *tradeable* aspects
- There are so many variables, it is *impossible* to build general statistically-valid estimation models for more than a few variables

  (Existing estimation models are mainly based on expert judgement)

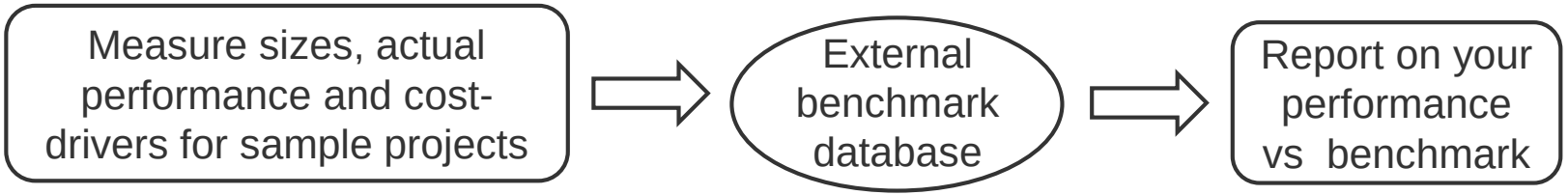
Agenda

- Goals and terminology
- The challenges of measuring and controlling the performance of software processes
- The **‘Scatter-gun’** approach
- The ‘Rifle-shot’ approach
- Conclusions

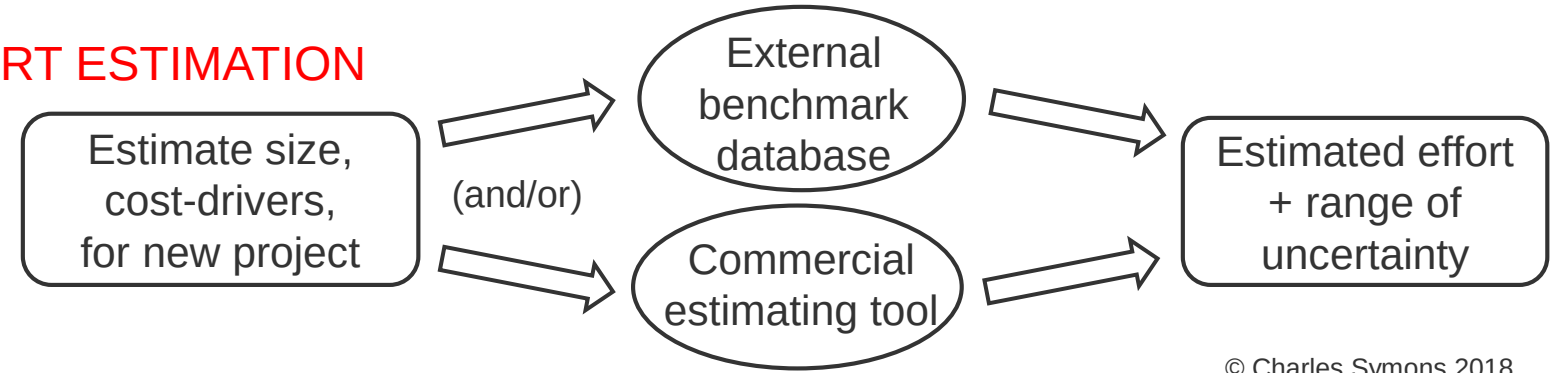
Suppose you want to use external benchmark data and estimating tools for the control cycle

The processes are simple in principle:

BENCHMARKING

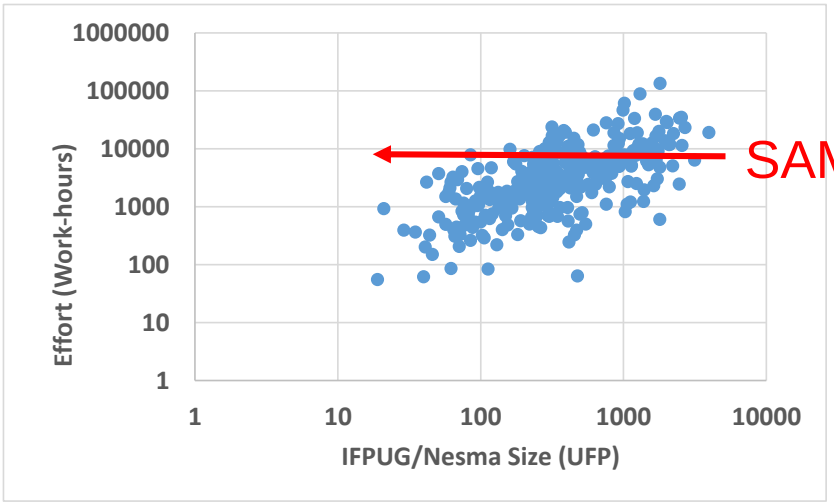


EFFORT ESTIMATION



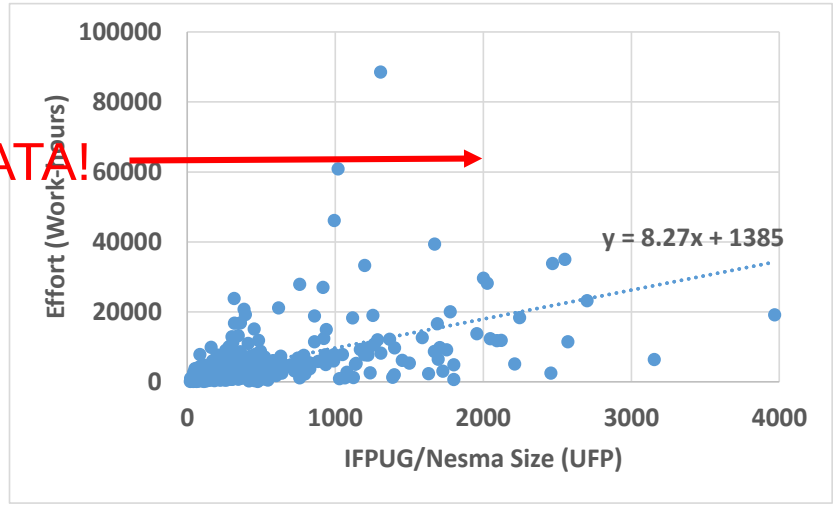
External benchmarking databases typically show large variations in performance across projects

A size/effort relationship?
(Note the log-log scales!)



SAME DATA!

Actually, a typical 'fan-shaped' size/effort distribution [5)]



Why? Organizations differ in their real performance, and report data inconsistently

Example

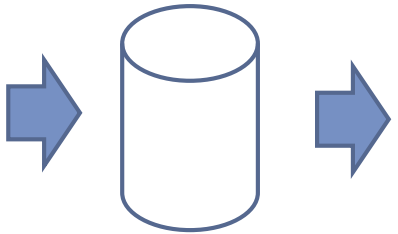
A project reports its total effort as ‘1550 work-hours’

- What activities were included in the effort figure?
  - All of feasibility study …… to implementation, or?
  - ‘Overheads’, specialists, customers?
- Standard hours or including overtime?

Benchmarking services, e.g. ISBSG, do their best to normalise reported effort data, and to check data quality.

Using e.g. ISBSG data for benchmarking or estimating is simple, but not very accurate

Search criteria:
Industry = Insurance
New development
Size measurement = IFPUG/Nesma
Programming language: Java



Work-hours/FP [6]	
Min	3.1
10%	5.3
25%	8.2
Median	11.5
75%	15.2
90%	19.7
Max	24.8

(174 projects)

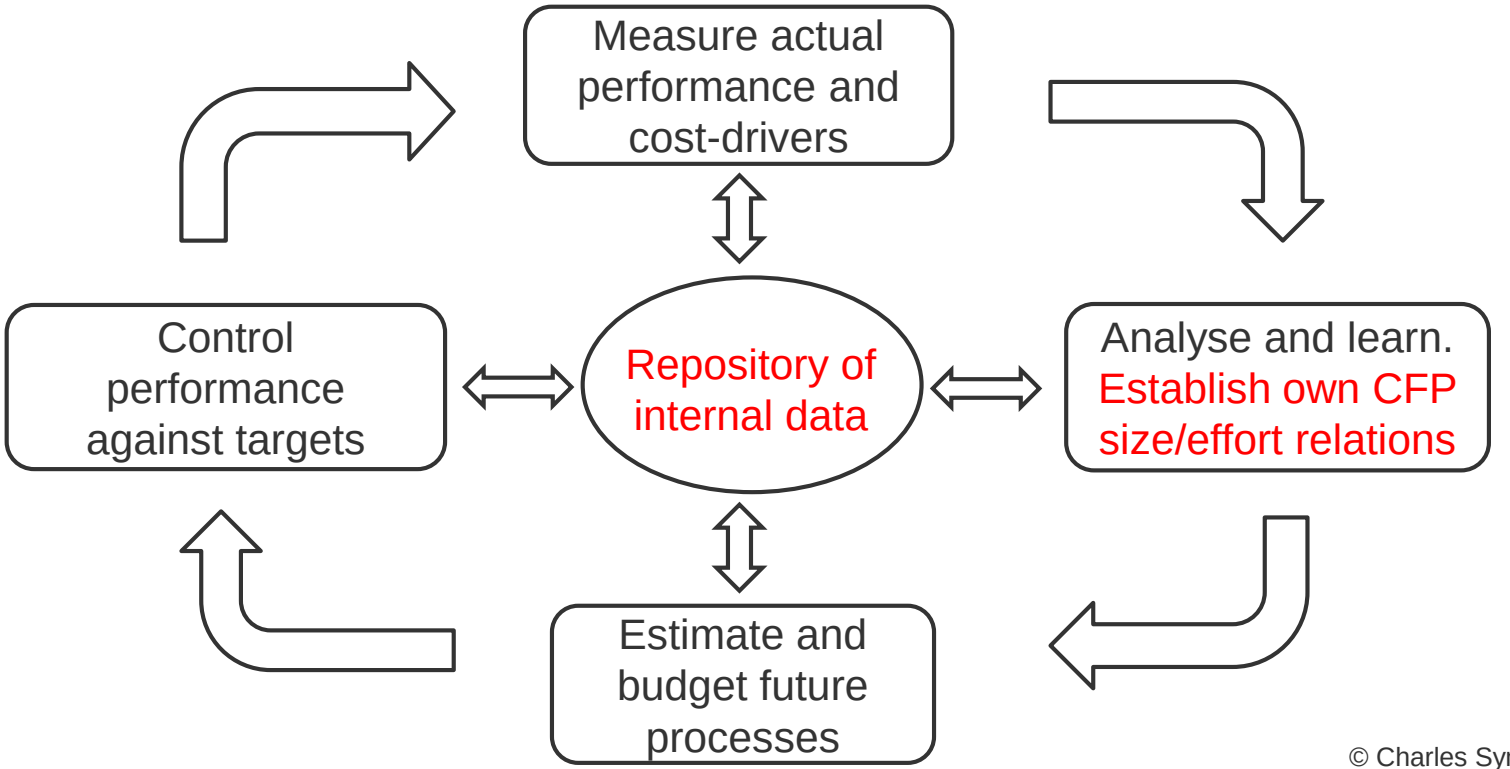
Estimating: new project software size = 200 FP ➡ Estimated effort = 2300 WH
(50% probability in range 1640 – 3040 WH)

Benchmarking: your average = 10 WH/FP ➡ You are 'slightly better than average'

Agenda

- Goals and terminology
- The challenges of measuring and controlling the performance of software processes
- The ‘Scatter-gun’ approach
- The ‘**Rifle-shot**’ approach
- Conclusions

Goal: master the cycle of managing software processes using COSMIC function point (CFP) sizes and internal data



Using the COSMIC method of measuring functional size has many advantages

- Based on fundamental software engineering principles, hence:
  - applicable to business, real-time and infrastructure software
  - at any level of decomposition
  - 'future-proof'
  - relatively easy to automate
- Variants exist for approximate size measurement, early in the life of a project
- 'Open', free, comprehensive documentation [7)]
- ISO/IEC standard; endorsed by US GAO, NIST, etc.

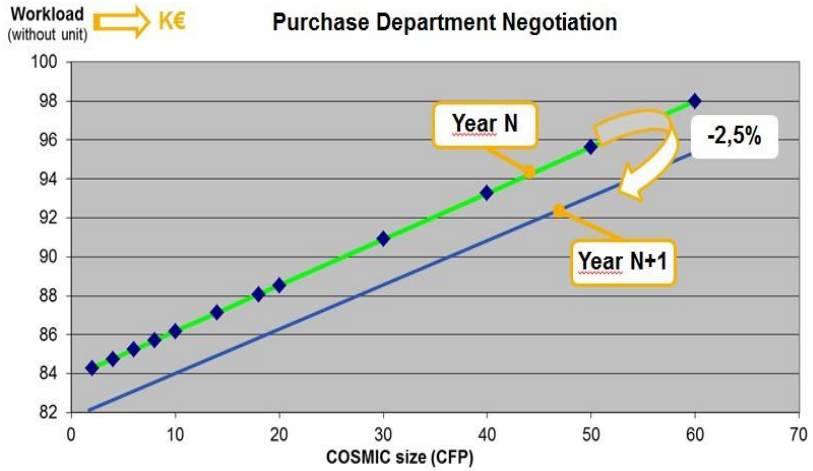
COSMIC-measured sizes correlate very well with effort. Case 1: Renault Automotive

Renault [8)] uses CFP sizing to control the development and enhancement of Electronic Control Units (ECU's)

- tracks progress of ECU specification teams…
- who create designs in Matlab Simulink…
- which are automatically measured in CFP

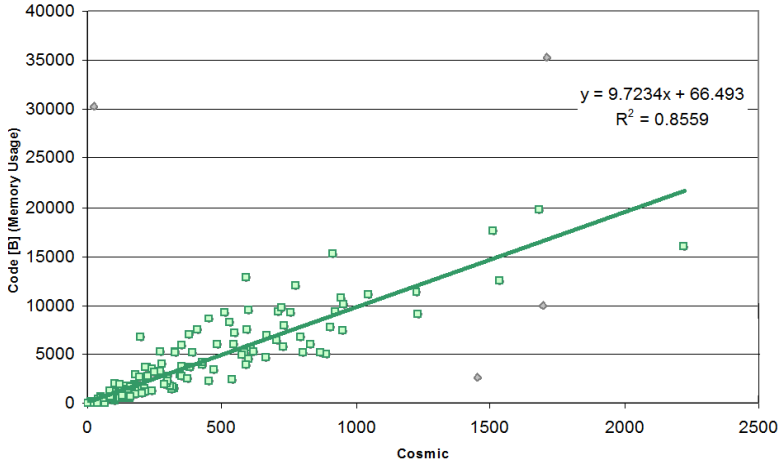
Motivation for automation: speed, accuracy of measurement

Renault achieves remarkable cost estimation accuracy from its ECU designs

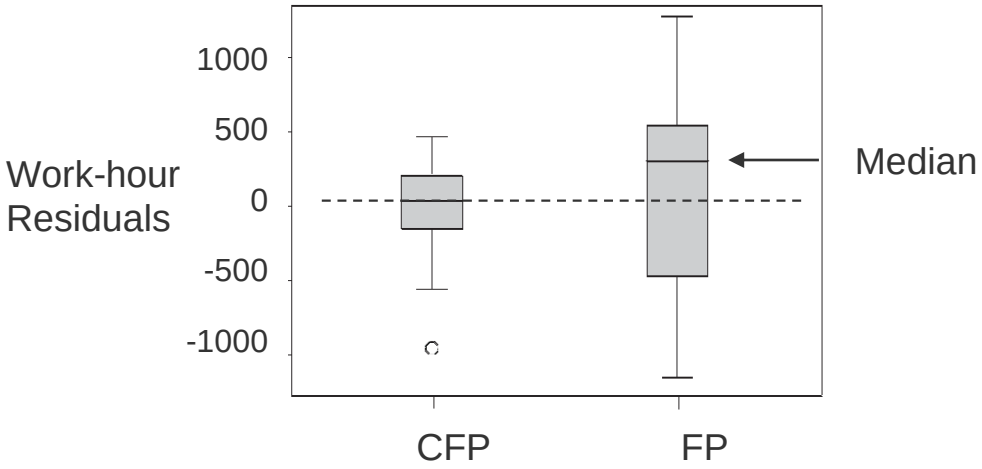


Cost vs size (CFP)

Memory size vs software size (CFP)



Case 2: Web effort estimation is more accurate with COSMIC than using classic FPA



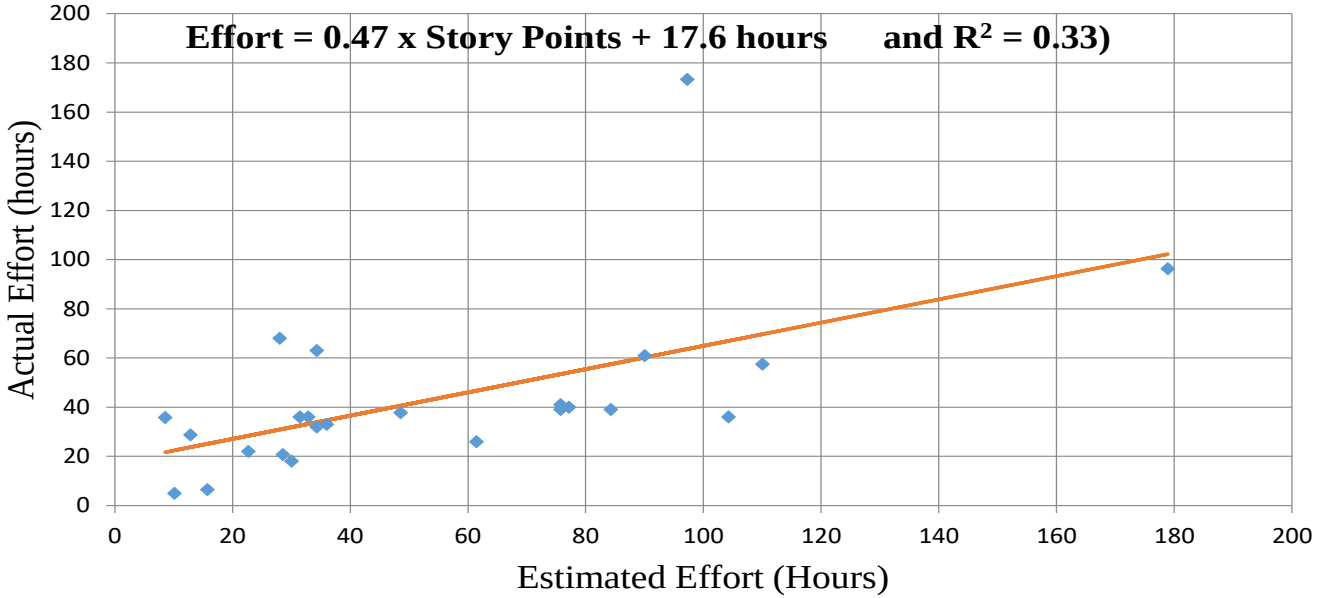
25 industrial Web applications [9)]

Conclusions:
'*The results of the … study revealed that COSMIC outperformed Function Points as indicator of development effort by providing significantly better estimations'*

Case 3: A Canadian supplier of security and surveillance software systems

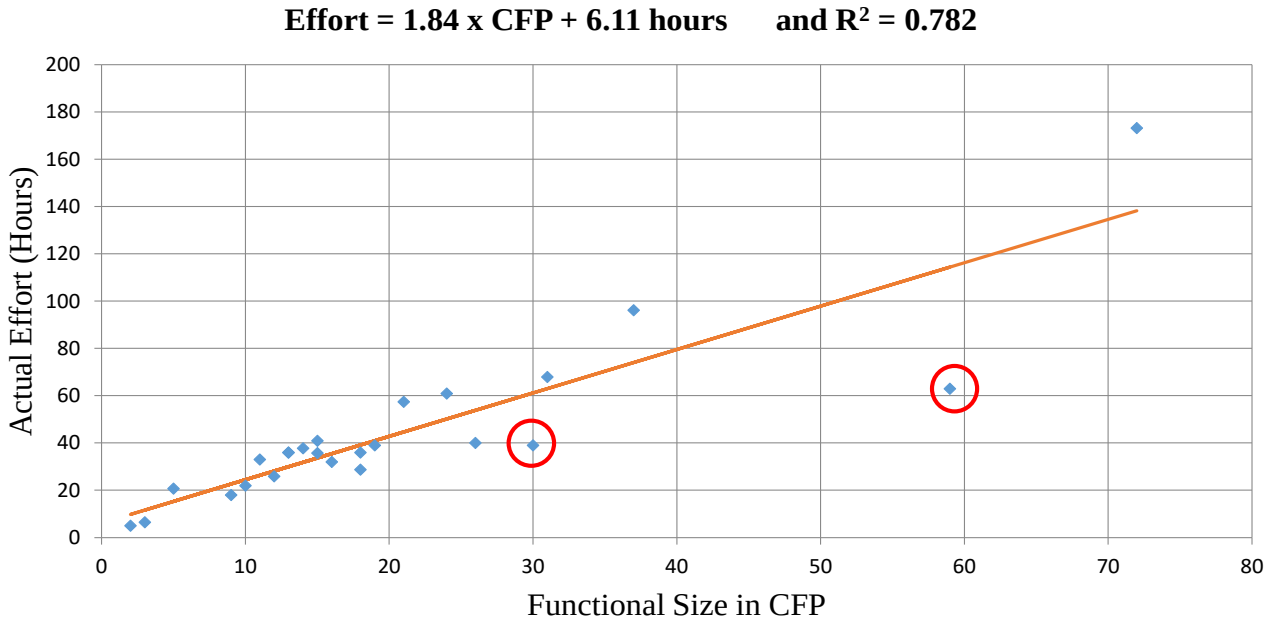
- A customer request for new or changed function is called a ‘task’
- Uses Scrum method; iterations last 3 – 6 weeks
- Teams estimate tasks within each iteration in User Story Points, and convert directly to effort in work-hours
- CFP sizes were measured on 24 tasks from nine iterations, for which USP ‘sizes’, estimated and actual effort data were available [10)]

User Story Point sizes are a poor predictor of effort



Notice the wide spread and the 17.6 hours 'overhead'

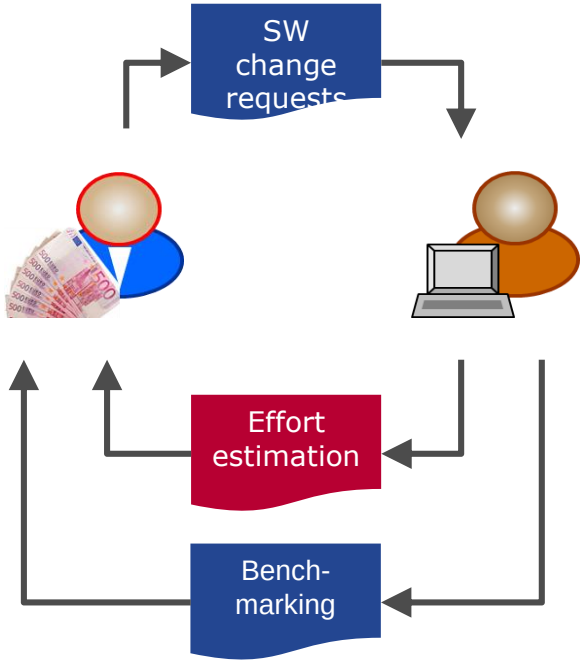
The CFP vs Effort graph shows a good fit, but reveals two outliers



Two tasks with low effort/CFP had significant software re-use.
Removing these outliers improves the R^2 to 0.977

Case 4: A global automotive manufacturer improved estimating for maintenance changes

- Context: real-time embedded software
- Starting point: text/diagrams for required changes
- A COSMIC-based measurement program [11] resulted in
  - Estimating precision of 10 – 20% within one year of starting
  - More disciplined, repeatable processes, internal benchmarks
  - Greater customer/supplier trust



Using only internal measurements of cost-drivers simplifies the data collection/analysis task

- No issues about consistency of your data with data from other organizations, e.g. you define:
  - rules for what to include in ‘effort’
  - ‘experience levels’ for your own staff
- In practice there will be fewer cost-drivers, e.g.
  - one industry, environment, culture, etc.
  - only a limited set of technologies

Although there are ‘fewer’ cost-drivers, they may still be quite varied

“Only a few factors affect the performance of a software project. The trouble is that these factors are different for every project.” ⟵ **TRUE**

Barbara Kitchenham, Professor, Keele University, UK [12)]

NOT ENTIRELY TRUE

Studies of project failures and of project risks show that a few cost-drivers are very common, e.g.

- uncertain or changing requirements
- staff experience in the business area or with a new technology
- management failures

So what internal data should we collect to achieve our goals?

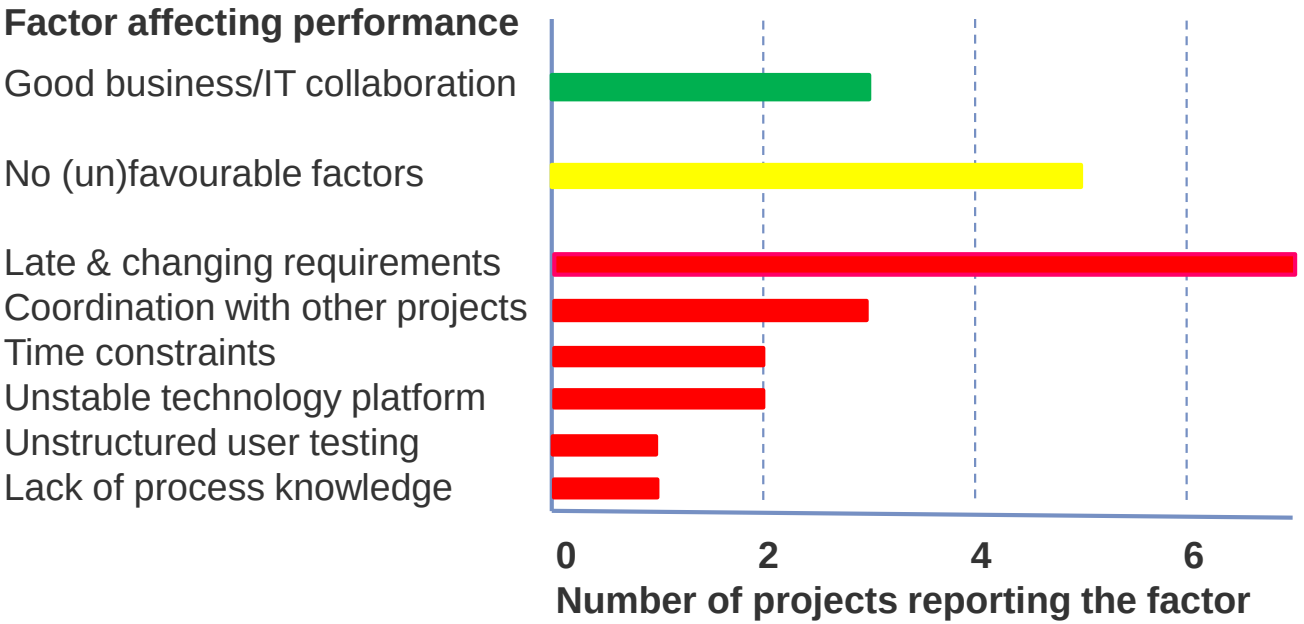
- Project ID, description, etc.
- Software size(s) in CFP
- Effort and time (estimated and actual), team size
- Product quality
- Technologies used (hardware/software)

and

- *'Describe the factors that affected the project favourably or unfavourably'*

Data from Post-Implementation Reviews (or Agile ‘retrospectives’) are very revealing and are actionable

Example [13)]
- UK insurance company
- 21 small enhancement projects



Agenda

- Goals and terminology
- The challenges of measuring and controlling the performance of software processes
- The ‘Scatter-gun’ approach
- The ‘Rifle-shot’ approach
- Conclusions

The Scatter-gun approach can be useful. The Rifle-shot approach offers greater benefits

Use when you:

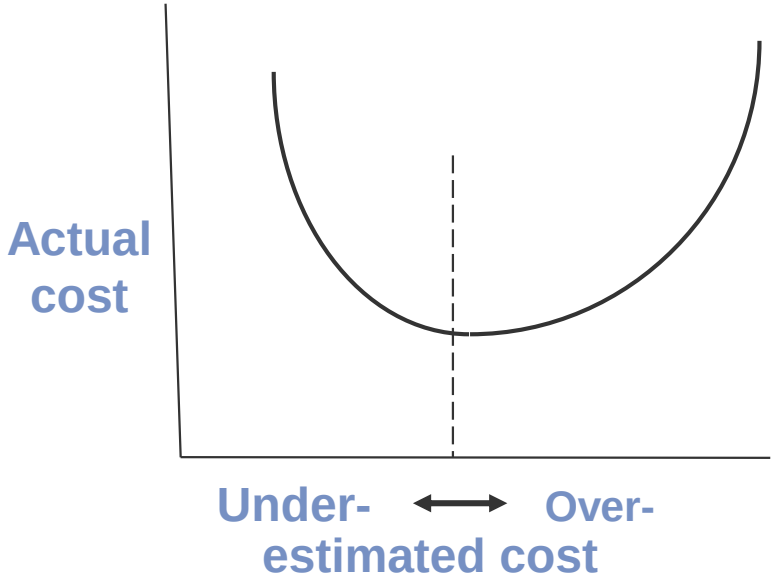
- have few measurements of your own projects
- have many technologies, processes
- need a quick ‘reality-check’ of an estimate for a new project
- want to compare your performance against peer organizations

Use when you

- are prepared to *invest* in measurement for the longer-term benefits
  - **improved processes**
  - **improved requirements quality**
  - **greater organizational learning**
  - **more accurate estimates**

Estimating accuracy is important: the most accurate estimate ⇨ the lowest project cost

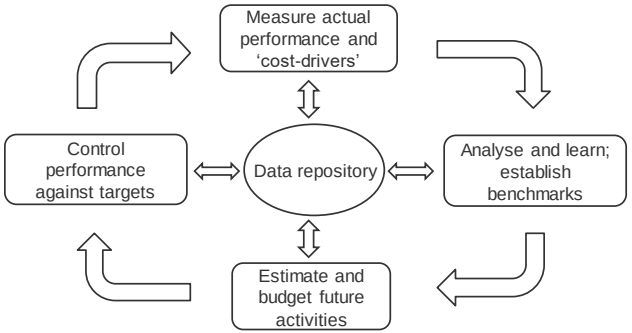
- Under-estimation leads to cost increases later in the project
- Over-estimation means the money will be spent ('Parkinson's Law')



…. but software estimation can never be an exact science, so repeat the control cycle frequently

Software development is partly mechanical,
but partly creative and unpredictable

repeat the control cycle
frequently



Agile Methods

using **COSMIC** Function Points!

~~Story Points~~

Thank you for your attention

Charles Symons (www.cosmic-sizing.org)
cr.symons@btinternet.com

References

1. 'ISBSG Data Collection Questionnaire: new development, redevelopment or enhancement, sized using IFPUG or Nesma Function Points'. www.isbsg.org
2. COCOMO II Model Definition Manual, 2000, http://sunset.usc.edu/research/COCOMOII/Docs/modelman.pdf
3. COSMIC/ISBSG Glossary of terms for Non-Functional Requirements and Project Requirements used in software project performance measurement, benchmarking and estimating', v1,0 September 2015. https://cosmic-sizing.org/publications/glossary-of-terms-for-nf-and-project-requirements/
4. 'Software sizing, estimation and risk management', Daniel Galorath, Michael Evans, Auerbach Publications, ISBN 0-8493-3593-0, 2006
5. ISBSG release 11 data, enhancement projects, 2009
6. Chapter on 'Benchmarking' in 'Dimensions of Productivity', Harold van Heeringen, Frank Vogelezang, to be published 2018
7. 'Introduction to the COSMIC method of measuring software', v1.1, https://cosmic-sizing.org/publications/introduction-to-the-cosmic-method-of-measuring-software-2/
8. 'Manage the automotive embedded software development cost & productivity with the automation of a Functional Size Measurement Method (COSMIC)" Alexandre Oriou et al, IWSM 2014, Rotterdam, www.ieeexplore.org
9. 'Web Effort Estimation: Function Point Analysis vs. COSMIC', Sergio Di Martino, Filomena Ferrucci, Carmine Gravino, Federica Sarro, Information and Software Technology 72 (2016) 90–109
10. 'Effort Estimation with Story Points and COSMIC Function Points - An Industry Case Study', Christophe Commeyne, Alain Abran, Rachida Djouab. 'Software Measurement News'. Vol 21, No. 1, 2016. Obtainable from www.cosmic-sizing.org
11. Private communication, Vector Consulting (Germany), 2016
12. Remark during British Computer Society talk, c1999
13. Private data, Symons Consulting (UK), 2003