THE COSMIC METHOD OF SOFTWARE SIZING AND ITS USES IN MANAGING AND ESTIMATING SOFTWARE ACTIVITIES

BCS Advanced Programming Group meeting
12th April 2018

Charles Symons
Agenda

- Goals: the importance of measuring software size
  - Overview of the COSMIC method
  - The acid test. Do COSMIC sizes correlate with effort?
  - Conclusions and future
The goal: master the whole cycle of managing software processes

- Measure actual performance and ‘cost-drivers’
- Establish benchmarks
- Control performance against targets
- Analyse and learn
  - Establish benchmarks
- Estimate future processes

Data repository

© Charles Symons 2018
Few organizations really master the control cycle

- High proportions of software project failures and cost over-runs
- Who does best?
  - Commercial software suppliers – a matter of survival
  - Agile method practitioners – (maybe) but only at the team level
- Why the problems? Developing software is partly an unpredictable process

© Charles Symons 2018
The performance of software processes has various aspects, and they 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.

© Charles Symons 2018
Mastering the control cycle requires a sound method for measuring software size

Sizing method options:

<table>
<thead>
<tr>
<th>Counts of Source Lines of Code:</th>
<th>Can’t estimate until software designed</th>
<th>Technology-dependent, no standards</th>
</tr>
</thead>
<tbody>
<tr>
<td>Development method specific, e.g. UCP, OOP, Story Points, etc.</td>
<td>No reliable standards; benchmark data possible only locally</td>
<td></td>
</tr>
<tr>
<td>Functional size:</td>
<td>International standard methods</td>
<td>Technology-independent</td>
</tr>
<tr>
<td></td>
<td></td>
<td>‘First Generation’ methods have limitations</td>
</tr>
</tbody>
</table>
Simple example: using the control cycle data to estimate effort for a project, iteration, etc.

Completed projects:

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

(Establish average ‘benchmark’ productivity for the type of project)

New project:

‘Typical’ estimated effort = \( \frac{\text{Estimated software size}}{\text{Benchmark project productivity}} \)

‘Best’ estimated effort = \( \left\{ \frac{\text{Estimated software size}}{\text{Benchmark project productivity}} \right\} \times \left\{ \text{Adjustments for project-specific ‘cost-drivers’} \right\} \)
A huge number of possible cost-drivers can affect performance
Summary: there are inherent challenges to implement the software control cycle:

- 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 of them
- Conclusions:
  - Collect your own size, effort, etc., data
  - Establish your own size/effort relationships
Agenda

- Goals: the importance of measuring software size
- Overview of the COSMIC method
- The acid test. Do COSMIC sizes correlate with effort?
- Conclusions and future
Method Goals

- A measure of functional requirements based on fundamental software engineering principles
- Applicable to business, real-time and infrastructure software
- Independent of technology or processes used for the software or project
- (Hopefully) produces sizes that correlate well with effort
- Open, free
All software functional requirements can be broken down into ‘functional processes’
There are four types of ‘Data Movement’ sub-processes

Functional Users
- Hardware devices,
- Other software or
- Humans

The ‘Data Movement’ is the unit of measure: 1 CFP (COSMIC Function Point)
A Functional Process responds to an ‘Event’ that a ‘Functional User’ detects or generates:

1. **Triggering Event** causes a **Functional User** to generate a **Data Group**.
2. The **Data Group** is moved into a Functional Process (FP) by its ‘Triggering Entry’.
3. **New employee starts work**.
   - Personnel Officer types employee details and adds them to the DM.
   - Personnel Officer then types the employee ID/name and adds it to the DM.

© Charles Symons 2018
Some real-time examples

Triggering Event causes a Functional User to generate a Data Group. Boundary that is moved into a FP by its Triggering Entry. Functional Process.

End of time interval → Clock → Clock tick → Entry DM of tick (= ‘start processing’)

Missile approaching → Aircraft radar → Message ‘Missile approaching’ → Entry DM with radar info

© Charles Symons 2018
Definition of a Functional Process (abbrev.)

a) A set of data movements … of the functional requirements …. being measured, …. that can be defined independently of any other functional process in those requirements.

b) … Each functional process starts processing on receipt of a data group moved by its Triggering Entry data movement.

c) The set of all data movements of a functional process is the set that is needed to meet its requirements for all the possible responses to its Triggering Entry.
Some more definitions

A **data movement** (E, X, R or W) moves a single **data group**, where:

- A **data group** consists of one or more **data attributes** that describe a single **object of interest**

- An **object of interest** is any ‘thing’ (physical or conceptual) in the world of the **functional user**, about which the software being measured must process or store/retrieve data

(Think of an entity-type, a relation in 3NF, or the subject of an object class)
Example business application Functional Processes

Create Employee

Enquire on current salary

‘Maintain’ employee salary

Update salary

4 CFP

6 CFP

3 CFP
Example real-time Functional Processes

Simple thermostat

- Clock Tick
- Actual Temp.
- Target Temp.
- On/Off command

4 CFP

X to heater

Complex avionics

- Missile detected
- (Guess) Multiple X’s
  - Sound alarm
  - Pilot info
  - Release chaff
  - Evasive action
  - Etc.

Many CFP

© Charles Symons 2018
There is no upper limit to the size of a functional process

- A functional process must have at least 2 CFP
  - A triggering Entry
  - An ‘outcome’ - i.e. a Write or an Exit
- Largest reported functional processes?
  - In banking ~ 65 CFP
  - In avionics >100 CFP
- The smallest change to an existing functional process is 1 CFP
Measurement involves a three-phase process

Measurement sponsor input → Measurement Strategy → Definitions:

- Functional Requirements
- COSMIC Principles
- Software to be measured
- Required measurement

Mapping:

- Functional requirements
- COSMIC Principles
- Requirements in the form of the COSMIC Model of the software

Measurement:

- Functional size of the software in units of CFP

Definitions:

- Software to be measured
- Required measurement
- Functional requirements
- COSMIC Principles

© Charles Symons 2018
Measurement Strategy phase 1: define the measurement parameters

Software parameters
- Scope
- Functional users
- Layer(s)
- Level of Decomposition

Level of Granularity of the requts.

Recommendation: define ‘patterns’ for standard M’ment Strategy parameter sets

© Charles Symons 2018
Mapping phase 2: map the requirements to the COSMIC model

- Functional Requirements
- Events (via Functional Users)
- Functional Processes
- Objects of interest
- Data Groups
- Data Movements

© Charles Symons 2018
Measurement phase 3: count the data movements

Within a defined Measurement Scope:

\[
\text{Software size} = \text{Sum of sizes of Functional Processes} = \text{Count of all their Data Movements}
\]

\[
\text{Size of a change to software} = \text{Count of DM’s added} + \text{Count of DM’s modified} + \text{Count of DM’s deleted}
\]
An example result from a measurement

<table>
<thead>
<tr>
<th>Acme Car Hire Functional Processes</th>
<th>Data Group Names</th>
<th>Nos. of Data Movements</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Customer name</td>
<td>E R X</td>
</tr>
<tr>
<td>Search Customer by name</td>
<td>Customer Master Record</td>
<td>E X X</td>
</tr>
<tr>
<td>View Customer Summary details</td>
<td>Customer ID</td>
<td>R E X</td>
</tr>
<tr>
<td>View Customer Details</td>
<td>Customer Details</td>
<td>R E X</td>
</tr>
<tr>
<td>Update Customer details</td>
<td>Customer latest Invoice</td>
<td>X</td>
</tr>
<tr>
<td>Add new Customer</td>
<td>Existing Bookings</td>
<td>W E</td>
</tr>
<tr>
<td>Print current Invoice</td>
<td>Booking Details</td>
<td>W E</td>
</tr>
<tr>
<td>View Booking details</td>
<td>Error/Confirmation Message</td>
<td>R X</td>
</tr>
<tr>
<td></td>
<td>Totals for Acme System</td>
<td></td>
</tr>
<tr>
<td>Objection</td>
<td>Response</td>
<td></td>
</tr>
<tr>
<td>-----------------------------------------------</td>
<td>---------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Needs too much detail for early estimating</td>
<td>There are variants for approximate sizing</td>
<td></td>
</tr>
<tr>
<td>Non-functional requirements</td>
<td>Quality NFR evolve wholly or partly into functional requirements that COSMIC can measure. Other NFR affect cost, effort but not software size</td>
<td></td>
</tr>
<tr>
<td>Complexity</td>
<td>A COSMIC size closely measures the software ‘crude complexity’ of the functional requirements at the level of granularity of the data movements</td>
<td></td>
</tr>
<tr>
<td>Re-used software</td>
<td>Distinguish sizes of new and re-used software</td>
<td></td>
</tr>
</tbody>
</table>
In Agile processes, COSMIC sizes can be measured at any level of aggregation. Hence usable for:

- early total System sizing and effort estimation,
- User Story sizing and estimation,
- progress control, etc.

© Charles Symons 2018
Agenda

- Goals: the importance of measuring software size
- Overview of the COSMIC method
- The acid test. Do COSMIC sizes correlate with effort?
- Conclusions and future
Case 1: Renault Automotive use in embedded software

Renault\(^1\) 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 ‘1G’ FPA

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’
- Scrum method used with iterations of 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
User Story Point sizes are a poor predictor of effort

Effort = 0.47 x Story Points + 17.6 hours and \( R^2 = 0.33 \)

Notice the wide spread and the 17.6 hours ‘overhead’
The CFP vs Effort graph showed a good fit, but revealed two outliers

Effort = 1.84 x CFP + 6.11 hours and $R^2 = 0.782$

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 resulted in
  - Estimating precision of 10 – 20% within one year of starting
  - More disciplined, repeatable processes, internal benchmarks
  - Greater customer/supplier trust
Conclusions from case studies of size/effort relationships

COSMIC-measured sizes correlate very well with effort.

Investing in COSMIC measurement and recording cost drivers should help improve:

- estimating accuracy
- organizational learning for process improvement
- quality control of requirements

**Most accurate cost estimate → least cost project**

© Charles Symons 2018
Agenda

- Goals: the importance of measuring software size
- Overview of the COSMIC method
- The acid test. Do COSMIC sizes correlate with effort?

Conclusions and future
The COSMIC method has many advantages over other methods of measuring software size

- Based on fundamental software engineering principles, hence:
  - ‘future-proof’ (and stable)
  - relatively easy to automate
- Applicable to business, real-time and infrastructure software, at any level of decomposition
- ISO/IEC standard; endorsed by GAO\(^6\), NIST\(^7\), etc
- ‘Open’, freely available via [www.cosmic-sizing.org](http://www.cosmic-sizing.org)\(^8\)

\(^{6}\) GAO (Government Accountability Office)
\(^{7}\) NIST (National Institute of Standards and Technology)
\(^{8}\) Website URL
Estimating software processes can never be an exact science - so iterate!

Software development is partly mechanical, but partly creative and unpredictable

Repeat the control cycle frequently

AGILE!

using a proper size scale – Story Points

COSMIC Function Points

© Charles Symons 2018
Thank you for your attention

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


4. Private communication, Vector Consulting (Germany), 2016


© Charles Symons 2018