Nesma on sizing
Part 3: Other sizing methodologies

In this whitepaper Nesma presents information to enlarge your knowledge about sizing methodologies other than FPA.

The information is structured around five subjects:

1. Cosmic FSM
2. Lines of Code
3. Usecase points
4. Story points
5. Cyclomatic Complexity (McCabe)

1. COSMIC FSM

The COSMIC method defines the principles, rules and a process for measuring a standard functional size of a piece of software. 'Functional size' is a measure of the amount of functionality provided by the software, completely independent of any technical or quality considerations.

Applicability of the method

The COSMIC method may be used to size software such as business applications; real-time software; infrastructure software such as in operating systems; and hybrids of these. The common characteristic of all these types of software is that they are dominated by functions that input data, store and retrieve data, and output data. The method is not designed to be applicable to size software that is dominated by functions that manipulate data, as in typical scientific and engineering software.

Subject to the above, the method may be applied to measure the FUR of software:

- At any level of decomposition, e.g. a 'whole' piece of software or any of its components, sub-components, etc;
- In any layer of a multi-layer architecture;
- At any point in the life-cycle of the piece of software;

The principles for measuring the COSMIC functional size of a piece of software

The COSMIC method measures a size as seen by the 'functional users' of the piece of software to be measured, i.e. the senders and/or intended recipients of the data that must enter or exit from the software, respectively.

The method uses a model of software, known as the 'COSMIC Generic Software Model', which is based on fundamental software engineering principles, namely:

- Functional user requirements of a piece of software can be analyzed into unique functional processes, which consist of sub-processes. A sub-process may be either a data movement or a data manipulation;
- Each functional process is triggered by an 'Entry' data movement from a functional user which informs the functional process that the functional user has identified an event that the software must respond to;
- A data movement moves a single data group of attributes describing a single 'object of interest', where the latter is a 'thing' of interest to a functional user;

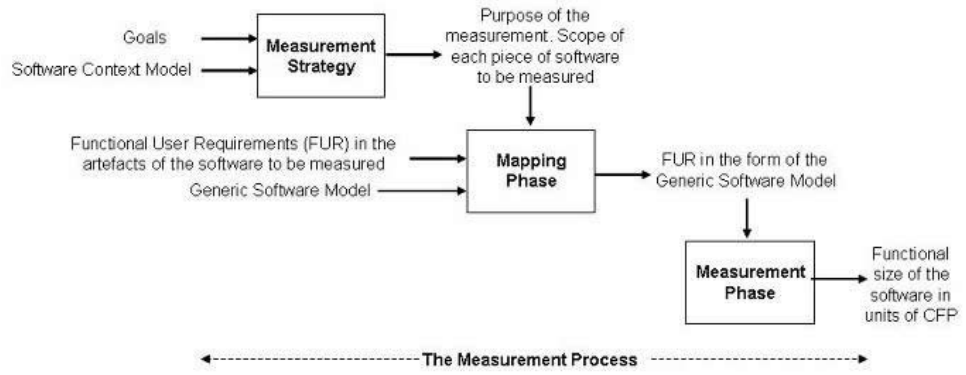
There are four types of data movement sub-processes. An **'Entry'** moves a data group into the software from a functional user and an **'Exit'** moves a data group out. **'Writes'** and **'Reads'** move a data group to and from persistent storage, respectively.

As an approximation for measurement purposes (and in light of the applicability of the method, described above), data manipulation sub-processes are not separately measured.

The size of a piece of software is then defined as the total number of data movements (Entries, Exits, Reads and Writes) summed over all functional processes of the piece of software. Each data movement is counted as one 'COSMIC Function Point' ('CFP'). The size of a functional process, and hence the size of a piece of software, can be a minimum of 2 CFP, with no upper limit.

The process for measuring the COSMIC functional size of a piece of software

The COSMIC measurement process has three phases, as shown in the Figure below:



For the detailed measurement rules, see 'The COSMIC Functional Size Measurement Method v4.0: Measurement Manual, the COSMIC Implementation Guide for ISO/IEC 19761: 2011'. There is another important document in the v4.0 series, namely: Introduction to the COSMIC method of measuring software (A more detailed overview of the method than is given here).

All COSMIC Method documents are obtainable free-of-charge from the downloads section of the COSMIC Portal

2. Lines of Code

The oldest metric for software projects is that of “Lines of Code” (LOC). This metric was first introduced around 1960 and was used for productivity measurement, quality measurement and economic calculations. Productivity was measured in terms of “lines of code per time unit.” Quality was measured in terms of “defects per kLOC”. The economics of software applications were measured using “currency units per LOC.” Lines of Code were reasonably effective for all three purposes.

When Lines of Code were first introduced there was only one widely used programming language and that was basic assembly language. Programs were small and coding effort comprised about 90% of the total work. Physical lines and logical statements were the same thing for basic assembly language. In this early environment, LOC metrics were useful for productivity measurement, quality analyses and economic calculations.

As the software industry changed, the Lines of Code as a metric did not change and so became less and less useful without very many people realizing it. The advent of COBOL, FORTRAN, PL/I, and other third generation programming languages would soon lower coding effort. Larger applications would expand requirements and design effort. LOC began to lose accuracy. This loss of accuracy continues further with the advent of fourth generation programming languages like Ingres, PowerBuilder and RPG.

For sizing existing software Lines of Code is still frequently used. More often than people realize this metric is combined with functional size measurement in a technique called “Backfiring”: many of the current estimation tools still use Lines of Code under the hood of their calculation engines.

3. Use Case points – Object Oriented Size Measurement

Use Case Points (UCP) is a software estimation technique used to forecast the software size for software development projects. The concept of UCP is based on the requirements written in use cases, part of the UML technique. The UCP size is calculated based on elements of use cases with factors to account for technical and environmental considerations. The UCP for a project can then be used to calculate the estimated effort for a project.

The UCP technique was developed by Gustav Karner in 1993 while employed at what was known at the time as Objectory Systems, which later merged into Rational Software and then IBM. The UCP method was created to solve for estimating the software size of systems that were object oriented. It is based on similar principles as function points, but was designed for the specific needs of object oriented systems and system requirements based on use cases.

4. Story Points - Sizing in an Agile context

Story Points are a metric used in agile project management and development to estimate the difficulty of implementing a given story. In this context, a story is a particular business need assigned to the software development team. Story Points are a relative measure of effort with respect to a simple story that is known to the whole team and defined as 1 Story Point. Story Points sizes are assigned according to (an adaptation of) a Fibonacci sequence. Elements considered in assigning a story point size include the complexity of the story, the number of unknown factors and the potential effort required to implement it.
Assigning story point sizes to stories is usually done per iteration or sprint in a planning poker session. Each team member can assign a story point size by means of a planning poker card with a number from (an adaptation of) a Fibonacci sequence. During the discussion of a story, numbers must not be mentioned at all to avoid anchoring, where the first number spoken aloud sets a precedent for subsequent estimates. After discussion each team member assigns a size value by playing a card. When all team members play the same card, there is agreement about the size. When the numbers vary, the lowest and highest values are explained and discussed and a new round of cards is played.

This process is a variation of the Wideband Delphi method. It is commonly used in agile software development. The planning poker method was first defined and named by James Grenning in 2002.

5. Cyclomatic Complexity

Cyclomatic complexity does not measure the size of software in the same way as the other sizing techniques described. Cyclomatic complexity is a software metric developed by Thomas McCabe in 1976 and is used to indicate the complexity of a program. It directly measures the number of linearly independent paths through a program's source code.
Cyclomatic complexity is computed using the control flow graph of the program: the nodes of the graph correspond to indivisible groups of commands of a program, and a directed edge connects two nodes if the second command might be executed immediately after the first command. Cyclomatic complexity may also be applied to individual functions, modules, methods or classes within a program.

Mathematically the cyclomatic complexity is defined as:

$$M = E - N + 2P$$

where

- E = the number of edges of the graph
- N = the number of nodes of the graph
- P = the number of connected components