CHAPTER 17

The COSMIC Method for Measuring the Work-Output Component of Productivity

Charles Symons, Common Software Measurement International Consortium (COSMIC), UK

The productivity of a software activity may be defined generally as work-output/work-input, where work-input is the effort needed to produce the work-output. In this chapter, we describe the ISO standard COSMIC method, which was designed to measure a size of the work-output from a software process. Measured sizes must be useful for both productivity measurement and for effort estimation, for most types of software.

For this chapter, we leave aside all the issues of how to interpret and exploit measurements of the productivity of software activities (e.g., the factors that affect productivity, the effect of measurements on the persons measured, etc.). Our challenge is how to measure a size of the work-output of software developers in a way that:

- Is independent of the technology used (e.g., language, platform, tools etc.), enabling productivity comparisons across different technology-sets
- Is credible and acceptable to the team or project whose performance is measured so that there is a clear connection with their total work-input, so not just, for example, the code size produced by the programmers in the team

C. Sadowski and T. Zimmermann (eds.), *Rethinking Productivity in Software Engineering*,
https://doi.org/10.1007/978-1-4842-4221-6_17

- Is demonstrably useful for estimating the effort for future activities
- Does not take up too much time and effort in relation to how the results will be used (automatic measurement being the ideal)

As well as being able to measure a *delivered* size and/or a *developed* size in the case of new software, the method must be able to measure a *changed* size in the case of a maintenance or enhancement task or a *supported* size in the case of support activities.

Measurement of Functional Size

In the late 1970s, Allan Albrecht proposed a method for measuring a size of the functional requirements for a piece of software, an "amount of functionality delivered to the user." This was a nice piece of lateral thinking that led to the development of function point analysis. His method is now maintained by the International Function Point Users Group (IFPUG) and is still widely used.

Function point analysis was a big advance over counting source lines of code as a size measure since the latter are technology-dependent and cannot be estimated accurately until a software project is well advanced—too late for most project budgeting purposes. In contrast, sizes of requirements measured in units of function points are technology-independent. Hence, their use enables comparisons of productivity across different technologies, development methods, etc., and a software size can be estimated quite early in a project, as requirements-elicitation proceeds.

However, Albrecht's function point analysis has a number of disadvantages in the context of modern software development. In 1998, therefore, an international group of software measurement experts established the Common Software Measurement International Consortium (COSMIC) aiming to develop a new method for measuring functional requirements that overcomes the weaknesses of function points. Table 17-1 summarizes the key differences between Albrecht's function point analysis and the COSMIC method. (FP = function points; CFP = COSMIC function points.)

Table 17-1. *Comparison of Albrecht's FPA Method with the COSMIC Method*

Factor	Albrecht's FPA Method	COSMIC Functional Size Measurement Method
Design origin	A 1970s-era IBM effort- estimation method.	Fundamental software engineering principles.
Design applicability	Whole business applications.	Business, real-time, and infrastructure software, at any level of decomposition.
Size scale	Limited size ranges for any one process or file. For example, a single process must have a size in the range 3–7 FP.	Continuous size scale. The smallest possible size of a single process is 2 CFP, but there is no upper limit to its size.
Measurement of changes	Can only measure the size of a whole process or of a whole file that must be changed.	Can measure the size of a change to any part of a process, so the smallest size of a change is 1 CFP.
Availability	Membership subscription.	Open, free [1].

The COSMIC Method

The method's design rests on two fundamental software engineering principles that are illustrated in Figures 17-1 and 17-2. In the following, all words in italics are precisely defined COSMIC terms [2].

- Software functionality consists of *functional processes* that must respond to *events* outside the software, detected by or generated by its *functional users* (defined as the "senders or intended recipients of data"). *Functional users* may be humans, hardware devices, or other pieces of software.
- Software does only two things. It moves data (entering from its *functional users* and exiting to them across the software *boundary* and from/to *persistent storage*), and it *manipulates* data.

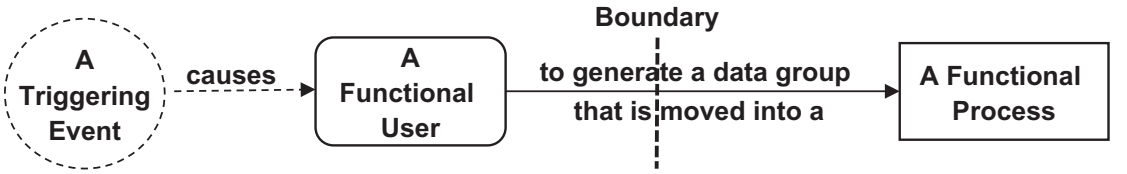

Figure 17-1. *The event/functional user/data group/functional process relationship*

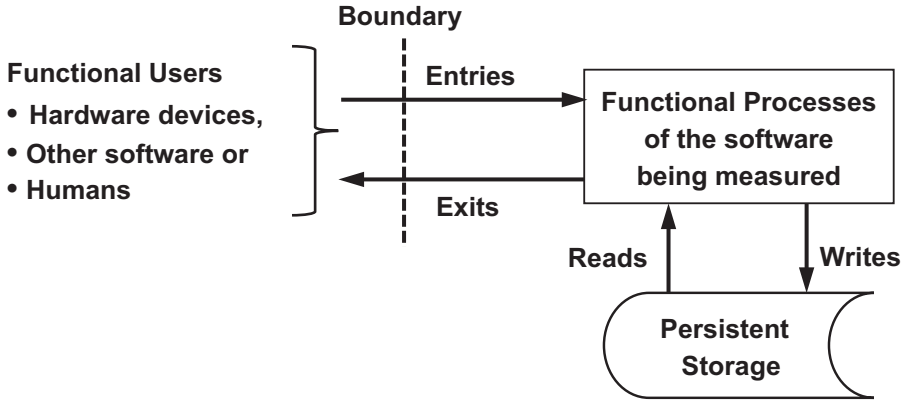

Figure 17-2. *The types of data movements of functional processes*

As there is no simple way to account for *data manipulation*, especially early in the life of a piece of software when requirements are still evolving, the COSMIC size of a *functional process* is measured by counting its *data movements.* In other words, this approach assumes that each *data movement* accounts for any associated *data manipulation*.

By definition, a *data movement* is a subprocess that moves a group of *data attributes* that all describe a single *object of interest* (think of an object-class, a relation in 3NF, or an entity-type). The unit of measurement is one *data movement*, designated as 1x COSMIC function point, or 1 CFP.

A *functional process* has a minimum size of 2 CFPs. It must have an *Entry* plus either an *Exit* or a *Write*, as the minimum outcome of its processing, but there is no maximum size. Single processes of size 60 CFP have been measured in business applications and more than 100 CFP in avionics software.

The functional size of a piece of software in CFPs is the sum of the sizes of all its *functional processes*. The size of any required change to a piece of software in CFPs is the count of its *data movements* that must be changed, regardless of whether changes must be to the *data group* moved and/or to the associated *data manipulation*.

Two examples illustrate the application of the method.

A simple functional process for a human functional user to enter data online about a new employee would have an Entry to move the new employee data, a Read of the database to check whether the employee already exists, a Write to create the new record, and an Exit to convey any validation error messages. The total size would be 4 CFP.

A functional process of a military aircraft may receive a triggering Entry from a sensor warning "missile approaching." The process will output several messages as Exits. Each Exit becomes the triggering Entry to a process in another part of the aircraft's distributed avionics system, for example, to issue warnings to the pilot to instruct the aircraft to take evasive action and other countermeasures. All communicating software components are functional users of each other; all input and output hardware devices are functional users of the software components with which they communicate.

Discussion of the COSMIC Model

In this section, we discuss various aspects of the model that might be argued to limit its practical value as a measure of work-output.

> *For effort estimation, we need size estimates long before we know the requirements in sufficient detail for a precise COSMIC size measurement.*

When there is a new software requirement, the thought process for an estimator is usually first "how big is it?" and then "what productivity figure should I use to convert size to effort?" For example, an agile team would estimate the size of a user story in story points and use a velocity figure measured on past sprints as the productivity value. This same thought process is involved when estimating the effort to develop or change a piece of software at any level of aggregation from a single user story all the way up to a major new system. Estimators need a software size scale and a size/effort relationship, i.e., productivity data, at each relevant level. The productivity data will have been established from measurements on past, completed tasks, or projects with characteristics similar to the new challenge.

However, a sponsor of a new software development typically needs a cost estimate for budget purposes long before the requirements have been spelled out in sufficient detail for a precise COSMIC size measurement. In practice, therefore, measurements of approximate sizes of early requirements for effort estimation may be as commonly needed as are precise sizes of delivered requirements for productivity measurement.

If the COSMIC models illustrated in Figures 17-1 and 17-2 and the definitions of the various terms are to succeed, it must mean that for any given artifacts of some software to be measured, everyone will identify and agree on the same set of functional processes. (The artifacts may be early or detailed statements of requirements, designs, implemented artifacts such as screen layouts and database definitions, or working code.) Correctly identifying the functional processes is the basis for ensuring measurement repeatability.

COSMIC method publications include a guideline [1] that describes several approaches, of varying sophistication, for measuring an approximate size of early requirements. All such approaches rely on being able to identify or estimate, directly or indirectly, the number "n" of functional processes in the early requirements for the new software. As an example, the simplest way of estimating an approximate COSMIC size of such requirements is to multiply the estimated "n" by an estimated average size of one process. More sophisticated approaches to approximate sizing include identifying patterns of functional processes that are known to occur for the type of software being estimated.

An organization wanting to use any of these approaches to approximate COSMIC size measurement will need to measure some software sizes accurately and use the results to calibrate the chosen approximate sizing approach.

What about nonfunctional requirements?

A method that aims to measure a size of functional requirements might appear to intentionally ignore nonfunctional requirements (NFRs). This would be nonsense since NFRs may need a lot of effort to implement. Loosely speaking, functional requirements define what the software must do, whereas NFRs define constraints on the software and the way it is developed or, in other words, how the software must do it.

A joint COSMIC/IFPUG study developed a clear definition of NFRs and a comprehensive glossary of NFR terms [3] and divided them broadly into two main groups.

- Technical NFRs such as the programming language or hardware platform to be used, or constraints from the environment such as the number of users to be supported. These NFRs do not affect software functional size. Rather, they may be factors that you need to understand when interpreting productivity measurements and that must usually be taken into account when estimating costs for a new development.

- Quality NFRs such as requirements for usability, portability, reliability, maintainability, etc. These evolve as a project progresses, wholly or largely[1], into requirements for software functionality. The size of this functionality can be measured in the normal way, using the standard rules of the COSMIC method, or can be estimated if required for a new development.

So, sizes measured using the COSMIC method should reflect all the functionality output as a result of the work-input on the software, regardless of whether this functionality was initially stated in terms of functional or nonfunctional requirements.

What about complexity?

Productivity measurements based on functional sizes are sometimes criticized for not reflecting software complexity. In a discussion of simplicity versus complexity, Murray Gell-Mann (in "The Quark and the Jaguar") shows that crude complexity can be defined as "the length of the shortest message that will describe a system at a given level of coarse graining." According to this definition, therefore, a COSMIC size closely measures the crude complexity of the functional requirements of a software system at the level of granularity of the data movements of its functional processes.

However, as already noted, COSMIC sizes do not take into account the size or complexity of the data manipulation associated with each data movement, i.e., algorithmic complexity. Experience suggests, however, that for a large part of business, real-time and infrastructure software, the amount of data manipulation associated with each type of data movement does not vary much. I know of only one actual measurement of the number of lines of algorithm (LOA) per data movement, which was for a very large chunk of a real-time avionics system. This showed, for example, that the median number of LOA associated with one data movement was 2.5, with 99 percent of data movements having no more than 15 LOA. This one piece of evidence supports the validity of the COSMIC method design assumption for this domain that the count of data movements reasonably accounts for any associated data manipulation, except for any areas of software that are dominated by mathematical algorithms. In business, real-time, and infrastructure software, these areas are typically few and concentrated.

[1]An NFR for a system response time may give rise partly to the need for specific hardware or use of a particular programming language (i.e., technical NFRs) and partly for requirements for specific software functionality. The latter can be taken into account in the measure of functional size.

If the development of some software requires significant amounts of new algorithms, the effort associated with this work should probably be separated out in any productivity measurement or should be estimated separately. Developing a new algorithm is essentially a creative process for which there may be no meaningful size/effort relationship. Alternatively, the functional size associated with the algorithms may be measured, e.g., by a locally defined extension to the standard COSMIC method.

Are sizes of functional requirements still relevant in a world of component-driven software development?

This question can be expressed more generally as "Can COSMIC sizing be used, and is it still relevant in the world of modern software development, where much software is assembled from reusable components, e.g., in the IoT or for mobile apps; when agile developers don't believe in detailed documentation and their processes may involve much rework; in outsourced software contracts; etc.?"

The first obvious point to make is that if we are ever to understand software productivity and use the measurements for estimating purposes, then we need a plausible, repeatable, technology-independent measure of work-output. The COSMIC method meets this need; sizes may be measured at any point in the life of a piece of software.

It is up to each organization to determine the problem it is trying to solve and then decide for itself *how and when* to apply the COSMIC method and *how to use* the resulting measurements.

Because any one software activity could result in many types of COSMIC size measurements, the parameters of each measurement must be recorded to ensure that its meaning will be clear for future users. These parameters include the domain of the software and its layer in the architecture and distinguish, for example the following:

- Sizes of new developments from sizes of changes or enhancements
- Sizes of developed from delivered software, where the latter includes bought-in or reused software
- The level of decomposition (or of aggregation) of the software

Experience suggests that an organization should start work-output measurement on its most commonly used software processes to build confidence in using the COSMIC method and in the resulting productivity measurements, before moving on to measuring more complex situations.

In summary, the design of the COSMIC method is a compromise between taking into account all the factors we might think of as causing work-output and the practical need that measurement should be simple and not need too much effort.

Correlation of COSMIC Sizes with Development Effort

The acid test of whether the COSMIC method is of real practical use is "Do CFP sizes, as measurements of work-output, correlate well with measurements of development effort, i.e., work-input?" If the correlations are good, then productivity comparisons should be credible, and the results can be used for new effort estimation purposes with known confidence.

Happily, studies over several years show that under repeatable conditions (same type of software, same technologies, common rules for effort recording, etc.), CFP sizes correlate well with effort for a variety of business and real-time software [4]. The correlations are significantly better, according to some studies, than when using Albrecht's FP sizes.

Recent studies on agile software developments [5] also show that CFP sizes correlate with effort far better than do story point sizes at the level of sprints or iterations. (Story points may be meaningful within individual teams, but they cannot be relied upon for productivity comparisons across teams, nor for higher-level effort estimation purposes.)

Figure 17-3 shows the measurements from one such study with a Canadian supplier of security and surveillance software. In their agile process, tasks are allocated to iterations lasting from three to six weeks. The effort for each task is estimated in Planning Poker sessions in units of story points on a Fibonacci scale, which are then converted directly to work-hours. Figure 17-3 shows the actual effort versus the estimated effort for 22 tasks in nine iterations that required a total of 949 work-hours.

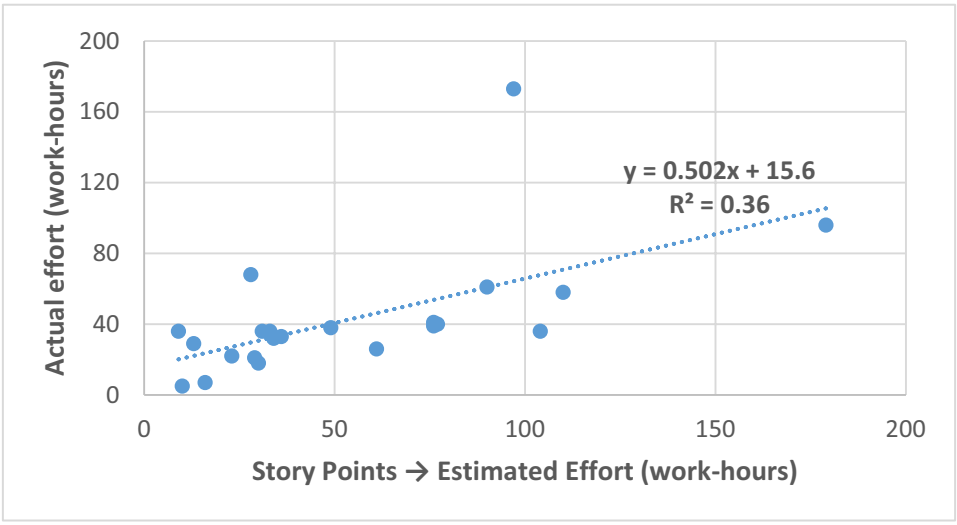

Figure 17-3. *Actual effort versus estimated effort*

The sizes of the 22 tasks were subsequently measured in units of COSMIC function points. Figure 17-4 shows the actual effort for these same 22 tasks plotted against the CFP sizes.

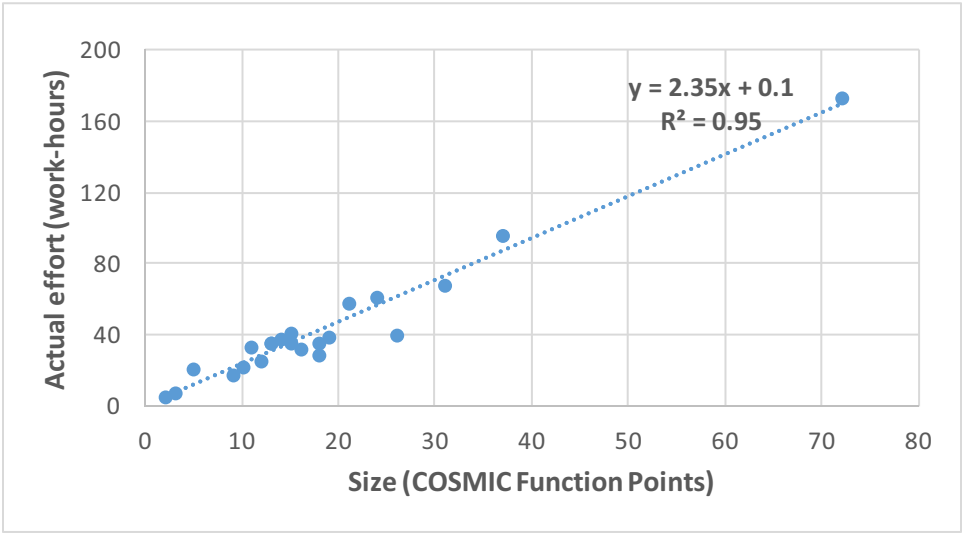

Figure 17-4. *Actual effort versus CFP sizes*

These two graphs show clearly the greatly improved correlation of task size versus effort when size is measured using COSMIC function points, rather than story points. Agile developers can substitute CFP sizes for story points to estimate or measure their work-output without any need to change their agile processes.

In addition to its uses in effort estimation, studies in the domains of embedded real-time and mobile telecoms software show that CFP sizes correlate well with the memory size needed for the corresponding code.

Organizations using the COSMIC method are now routinely exploiting these correlations to help estimate development effort from early software requirements or designs, or in agile environments.

Automated COSMIC Size Measurement

COSMIC size measurement automation is underway in three areas, in varying stages from early exploration to commercial exploitation.

a) Automated COSMIC sizing from textual requirements using natural language processing or artificial intelligence is still in the development stage. This step has great potential as it would allow early life-cycle estimating, e.g., of approximate sizes from user stories.

b) Automated COSMIC sizing from formal specifications or designs has reached the commercial exploitation stage in a few organizations. Here are two examples:

- Automatic CFP size measurement from UML models. Several Polish public-sector organizations rely on the results to help control price/performance of their software outsourcing contracts.
- Renault, the French automotive manufacturer, has implemented automatic COSMIC sizing of specifications held in the Matlab Simulink tool for the software embedded in its vehicle electronic control units [4]. CFP sizes are used to predict the development effort and the hardware memory size needed for the ECUs and to estimate the ECU execution times. The data is then used to control price/performance for the supply of ECUs and their embedded software. Other automotive manufacturers are known to be implementing these processes.

c) Automated COSMIC sizing from static and from executing Java code has been achieved with some manual input "seeding" of the code, with high accuracy.

Conclusions

The ISO-standard COSMIC method has met all its design goals and is being used around the world for measuring a functional size, i.e., work-output, for most types of software.

Measured sizes have been shown to correlate well with development effort for several types of software. The derived size/effort relationships are being used for effort estimation with, in some known cases of real-time software, great commercial benefits. The method has been recommended by the U.S. Government Accountability Office for use in software cost estimation.

The method's fundamental design principles are valid for all time. The method definition [2] is mature and has been frozen for the foreseeable future. Automatic COSMIC size measurement is already happening. As a further consequence of the universality of the method's underlying concepts, measured sizes should be easily understood and therefore acceptable to the software community whose performance is measured.

Measuring and understanding the productivity of software activities is a multifaceted topic. The COSMIC method provides a solid basis for the many needs of work-output measurement, a key component of productivity measurement.

Key Ideas

Here are the key ideas from this chapter:

- It's important for productivity measurement and estimating to have a measure for work output that can be compared across different contexts.
- COSMIC function points are such a measure.

References

[1] All COSMIC documentation, including the references below, is available for free download from www.cosmic-sizing.org. For an introduction to the method go to https://cosmic-sizing.org/publications/introduction-to-the-cosmic-method-of-measuring-software-2/.

[2] "The COSMIC Functional Size Measurement Method, Version 4.0.2, Measurement Manual (The COSMIC Implementation Guide for ISO/IEC 19761: 2017)," which contains the Glossary of Terms.

[3] "Glossary of Terms for Non-Functional Requirements and Project Requirements used in software project performance measurement, benchmarking and estimating," Version 1.0, September 2015, published by COSMIC and IFPUG.

[4] "Measurement of software size: advances made by the COSMIC community," Charles Symons, Alain Abran, Christof Ebert, Frank Vogelezang, International Workshop on Software Measurement, Berlin 2016.

[5] "Experience of using COSMIC sizing in Agile projects," Charles Symons, Alain Abran, Onur Demirors. November 2017. https://cosmic-sizing.org/publications/experience-using-cosmic-sizing-agile-projects/