

## COSMIC-based Project Management in Agile Software Development and Mapping onto related CMMI-DEV Process Areas

Enrico Berardi<sup>1</sup>, Luca Santillo<sup>2</sup>

<sup>1</sup>Tecnologie nelle Reti e nei Sistemi, <sup>2</sup>Agile Metrics

[enrico.berardi@trs.it](mailto:enrico.berardi@trs.it), [luca.santillo@gmail.com](mailto:luca.santillo@gmail.com)

### **Abstract:**

*Functional Size Measurement (FSM) methods have been introduced in the industry since late 70's and have been successfully adopted in software development projects for sizing the amount of functional requirements, often for software estimation purposes. Typically, measurements with FSM are performed in two moments of the project – early design and delivery, respectively supporting project estimation and validation. This approach „works“ for traditional waterfall cycles, typically requires adjustments in real world requirements changes over the projects, but is simply un-effective in modern agile development methodologies, where Evolutionary Requirements Analysis is the rule and the user (providing requirements and feedbacks) is constantly involved in the development process.*

*This work depicts the adoption of the new generation FSM COSMIC method in order to address the agile development process with a standard measurement approach. Such approach results in fine-tuned measures all over the project cycle for agile projects – with a single metrics that holds valid from early design to end phases, while adequately covering all the intermediate iterations and refinements.*

*Project management practices, such as Earned Value assessment, as well as general process improvement practices, such as the CMMI-DEV framework, are easily mapped onto COSMIC measurement and benefit from the proposed approach.*

### **Keywords**

*Agile, measurement, COSMIC, project management, Capability Maturity Model*

## **1. Introduction**

The thesis of this work is that the adoption of a standard Functional Size Measurement (FSM) method [13] in Agile Development methodologies (such as SCRUM, XP – eXtreme Programming, etc.) provides valuable benefits and improvements to project management and process improvement practices. Particularly, the COSMIC FSM method [12, 14] is proposed and investigated for adoption, due to the nature and characteristics of Agile methodologies. Impacts of the proposed approach on typical project management techniques (e.g. Earned Value, Change Management) and general process improvement (e.g. Capability Maturity Models) are further investigated and illustrated.

Agile Methodologies (Scrum, XP, etc.) use *ad hoc* metrics to plan the amount of work to do over the entire project (Release Plan) and in each single iteration (Iteration Plan) [2, 6]. Requirements (including both functional and non-functional types) contained in the Product Backlog [18] are estimated:

- directly, in terms of effort required for their implementation (i.e. in *person-hours*, *ph*, or multiples), or
- by assigning them a “weight”, denoted as “story points” (from the representation of user requirements in the form of user stories [7]).

Note that in the former case, estimating the effort is appropriate when referred to the Tasks that in fact are considered to be included in the given Iteration’s Backlog. In the latter case, to deal with effort estimates analogously, metrics of “productivity” (*story points/ph*) and “velocity” (*story points / iterations*) are derived in practice. These concepts are analogous to those of productivity (function points/ph) and product delivery rate (function points/month) in common software projects benchmarking practices [8], although the latter is obviously based on a different time measure.

Story points clearly remind of FSM (e.g. “Function Points”) [10, 13], but the unit of measurement that story points possibly represent is to be considered arbitrary, and their values – subjective (i.e. not derived from a model, either by means of a standard measurement method), therefore they cannot be transferred outside the business environment in which they are produced (the single project team). In other words, story points’ validity as a measurement method is not ensured, and their accuracy, precision, and comparability can vary significantly.

Because of this conceptual similarity between story points and Functional Size, the introduction of a standard FSM as COSMIC [12] can be very simple from a methodological point of view, even within an approach that would (and should) remain “agile”. (Note that non-functional aspects, such as technical or quality requirements, complexity, etc., are handled in Agile Methods [18] with some correction factors, respect to “nominal” values, according to the research lines currently open in the community of Software Measurement, e.g. [4], [8].)

In the authors’ opinion, the advantages of applying a FSM method in the context of agile development are quite self-evident, such as:

- measurement data of the project can be compared with (and contribute to) benchmarking data of the international community [11],
- the management may benefit from a uniform measurement model across different projects (and organizations),

- the application of the FSM can enhance collaboration between customers and developers adding transparency and objectivity (“*Customer collaboration over contract negotiation*”, such as stated in the Agile Manifesto [3]),
- the COSMIC software model, with concept of “functional process”, is a valuable tool for functional analysis and supports high-quality requirements analysis in terms of consistency and completeness.

From a process improvement perspective – e.g. within the “CMMI for Dev. 1.2” model [5], the introduction of the COSMIC method in an Agile Software Development framework would help to institutionalize at Capability Level 3 (Organization Level) several Process Areas such as Project Planning, Project Monitoring and Control, and Measurement and Analysis, which would otherwise remain at Capability Level 2 (Project Level) [19].

### 2. COSMIC FSM Method – Background

For an overview of the COSMIC method, the reader can refer to the COSMIC documentation; for an exhaustive illustration of concept definitions and measurement practices, the public domain COSMIC 3.0.1 Measurement Manual is suggested [14]. We only recall here the COSMIC Generic Software Model’s principles:

- a) Software receives input data from its functional users and produces output, and/or another outcome, for the *functional users*.
- b) Functional user requirements of a piece of software to be measured can be mapped into unique *functional processes*.
- c) Each functional process consists of *sub-processes*.
- d) A sub-process may be either a *data movement* or a *data manipulation*.
- e) 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*.
- f) A data movement moves a single *data group*.
- g) A data group consists of a unique set of *data attributes* that describe a single *object of interest*.
- h) There are four types of data movement. An *Entry* moves a data group into the software from a functional user. An *Exit* moves a data group out of the software to a functional user. A *Write* moves a data group from the software to persistent storage. A *Read* moves a data group from persistent storage to the software.

- i) A functional process shall include at least one Entry data movement and either a Write or an Exit data movement, that is it shall include a *minimum* of two data movements.
- j) As an approximation for measurement purposes, data manipulation subprocesses are not separately measured; the functionality of any data manipulation is assumed to be *accounted for* by the data movement with which it is *associated*.

The COSMIC definition of “Functional Process” is relevant for application in Agile methodologies and is hereby reported:

*“A functional process is an elementary component of a set of Functional User Requirements comprising a unique, cohesive and independently executable set of data movements. It is triggered by a data movement (an Entry) from a functional user that informs the piece of software that the functional user has identified a triggering event. It is complete when it has executed all that is required to be done in response to the triggering event.”*

Further details, measurement principles and rules can be found in [14, 15].

### **3. Applying COSMIC in Agile Development – A Case Study**

We make use of a case study to illustrate how the COSMIC method can provide a quantitative standard assessment of an agile software project, covering from its initial estimate to its latest iteration, and taking into account any requirement variation in between.

#### **3.1 Initial Estimate**

In most agile projects customer and supplier start sharing a “vision”, with the common aim to refine requirements in detail during the iterative development (evolutionary requirements). But if a formal contract is needed, they have to agree a “guess” about the overall budget from the very start (see [16], Tool 22: Contracts, for different types of contracts).

After a breadth-first analysis of the whole scope of involved requirements and related functionality, it is possible to estimate a Functional Size in CFP (COSMIC Function Point) for the scope of the Project. (If the COSMIC method is well known by the stakeholders, producing an *early or rapid approximate sizing* [15] is not more complicated than making a guess in *story points* or directly in effort.) On the other hand, the advantage is that having the (estimated) size expressed in CFP, one can compare against productivity benchmarking data to achieve a better effort estimate – being the benchmark being considered either local, or global.

In any case, a first estimate of the effort of the whole project can be easily derived, and at the same time one would get a reference baseline value (i.e. a Performance Baseline) for the average productivity and further sizing along the project, as well.

Needless to say, with a standard FSM method such as COSMIC, the size of the project will be expressed in a standard measurement unit (since the very beginning of the project), and that will remain the same along the whole project lifecycle, even if its practical values will be continuously refined, changing from estimation to measurement, and eventually increasing in case of scope creeping.

Suppose that for a new application we get, at an initial analysis, an estimate of the Functional Size of 320 CFP and that some related benchmarking data suggests a productivity figure of 0.04 CFP/ph (or, assuming a conversion factor of 8 ph per day, and 19 person days per month, 6.08 CFP/person-month).

Thus, we would have an overall effort estimate of ca. 8000 ph. (Of course, an error interval, or relative uncertainty, should also be estimated; this would depend on the degree of approximation of the size estimate and on the reliability of the productivity figure from the related benchmarks.)

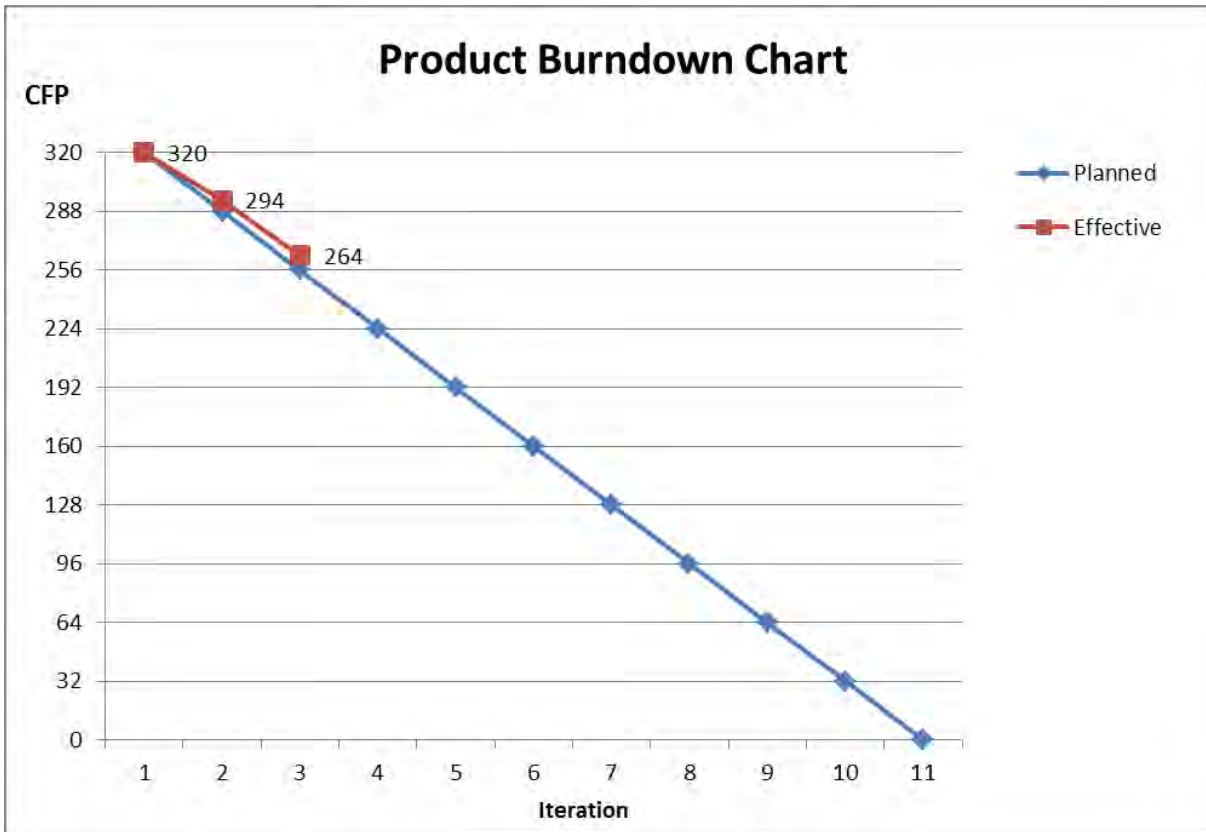
### 3.2 Planning the Project

Adopting an iterative lifecycle (with iterations of 4 weeks) and establishing a staff of 5 FTE (Full Time Equivalent), the duration of the project will be of 10 iterations, with an average velocity of 32 CFP per iteration.

The Product Burndown Chart in SCRUM, when applying the COSMIC method, can easily represent the Functional size “undone” in CFP vs. the number of Iterations (“Sprints”). The initialization of the Product Burndown Chart would look like the longer line (“Planned”) in Fig. 1 (for sake of simplicity, a linear relationship between effort, size and time is assumed – this assumption does not affect the generic case study results within the scope of this work, and would be replaced by other nonlinear trends at need). At initial stage, we would have established the first baseline of performance data, as reported in Tab. 1.

Overall Effort	8000 ph
Total Planned Size	320 CFP
Avg. Productivity	0,04 CFP/ph
Avg. Velocity	32 CFP/iteration

**Table 1:** Initial performance baseline.



**Figure 1:** Burndown Chart at initial stage and at a later stage.

### 3.3 The Backlog

In SCRUM, the Product Owner is continuously refining the Backlog, defining in more detail the requirements of higher priority and, together with the team, updating their estimates. Transposing to COSMIC, the Backlog items should correspond – at least for the higher priority section of it – to Functional Processes (see Sect. 3). Any Functional Process in the backlog would be assigned a value denoting its *measured* size expressed in CFP.

(Note that when a Functional Process is identified and described in detail, it is straightforward in the COSMIC method to achieve its Functional Size, thus this value would be no more an “estimate”.) In our example, the Product Backlog, at an early stage, would look like in Figure 2, left side.

We have supposed the existence of four Functional Areas. At an early stage the first is analysed in detail and their Functional Processes identified and measured in terms of Functional Size. The second Functional Area is analysed with less detail and only the number of possible Functional Processes is estimated, together with an average size for each of them. The two last Functional Areas are not analysed yet, but only estimated by analogy, always in terms of Functional Size.

Functional Area	Functional Process	Size (CFP)	Priority/Iteration	Done
F.A. 1  <i>High detail, functional size measured!</i>	F.P. 1	8	1	
	F.P. 2	10	1	
	F.P. 3	8	1	
	F.P. 4	6	1	
	F.P. 5	4	2	
	F.P. 6	4	2	
	F.P. 7	8	2	
	F.P. 8	8	2	
	F.P. 9	8	2	
	F.P. 10	6	3	
	F.P. 11	10	3	
F.A. 2  <i>Less detail, estimated the number of f.p. with an average size</i>	F.P. 1	8	3	
	F.P. 2	8	3	
	F.P. 3	8	4	
	F.P. 4	8	4	
	F.P. 5	8	4	
	F.P. 6	8	4	
	F.P. 7	8	5	
F.A. 3  <i>Low detail, Size estimated by analogy</i>		120	5 - 8	

Functional Area	Functional Process	Size (CFP)	Priority/Iteration	Done
F.A. 1  <i>High detail, functional size measured!</i>	F.P. 1	8	1	1
	F.P. 2	10	1	1
	F.P. 3	8	1	1
	F.P. 4	6	1	2
	F.P. 5	4	2	2
	F.P. 6	4	2	2
	F.P. 7	8	2	2
	F.P. 8	8	2	2
	F.P. 9	8	2	2
	F.P. 10	6	3	
	F.P. 11	10	3	
F.A. 2  <i>Less detail, estimated the number of f.p. with an average size</i>	F.P. 1	8	3	
	F.P. 2	8	3	
	F.P. 3	8	4	
	F.P. 4	8	4	
	F.P. 5	8	4	
	F.P. 6	8	4	
	F.P. 7	8	5	
F.A. 3  ...	...	...	...	...

**Figure 2:** Product Backlog at early stage (left) and at a later stage (right).

The Product Backlog contains no effort estimates, but only functional size measures (or estimates). At the initial stage, the performance baseline tell us that the team should be able to implement about 32 CFP per Iteration, with the estimated average productivity of 0.04 CFP/ph.

### 3.4 On the meaning of “done” in Agile development

A basic concept in Agile development is the meaning of “done” (the criterion by which an item in the Backlog may be marked as completed). In this regard, a Functional Process can be considered “done” if at least one acceptance test for each alternative path has passed with success. (Note also that COSMIC is more effective, for Backlog management, than “1<sup>st</sup> generation” methods, such as IFPUG [10]; in fact the latter keeps Transactional Functions separate from Data Functions, so one cannot actually consider a Transactional function “done” if (some parts of) Data Functions are “done” as well, and vice versa.)

In such an iterative approach, the team could implement multiple (incremental) versions of the same functional processes and put them in the Backlog as separate items. E.g. consider the case of a Functional Process with a size of 20 CFP. It might appear in the Backlog as two distinct items: FP<sub>x</sub>v1 with a size of 12 CFP, and FP<sub>x</sub>v2 with a size of 8 CFP; FP<sub>x</sub>v1 does not implements some data movements (for example when some external interfaces are stubbed) or does not manage conditions for alternative paths, or other.

### 3.5 Measuring the progress of the project

Continuing with our example, after the first two iterations we might have the following situation: in the first iteration the team has “done” 26 CFP, and in the second 30 CFP (Fig. 2, right side, column “Done”). The Product Burndown Chart would then look like in Fig. 1 (“Effective”). Thus, we might argue that the initial reference baseline was “optimistic”, or be confident that, from now on, after some initial difficulties, the team will increase its performance and will reach the initial objective. In the first case we should update the performance baseline – eventually leaving the overall estimated effort and schedule unchanged, but tuning the scope (i.e. the size initially planned), as reported in Tab. 2.

Overall Effort	8000 ph
Total Planned Size	<b>300</b> CFP
Avg. Productivity	0,0375 CFP/ph
Avg. Velocity	<b>30</b> CFP/iteration

**Table 2:** Updated performance baseline.

The percentage of completion (PoC) is easily calculated by the ratio:

$$\text{PoC} = \text{Size}_{\text{done}} / \text{Size}_{\text{total}} \quad (1)$$

where  $\text{Size}_{\text{done}}$  represents the Size implemented (“done”) up to now (i.e. at the  $i^{\text{th}}$  iteration) and  $\text{Size}_{\text{total}}$  is the Total Planned Size. So, with respect to the baseline of Table 2, the percentage of completion, after the 2<sup>nd</sup> iteration would be:  $\text{PoC} = 56/300 = 18.7\%$ .

The overall size may also vary for later analysis and more details achieved on the other Functional Areas, but this is a commonplace in Agile Projects. You should keep the Actual Performance Data updated, controlling the evolution of the whole scope (Total Planned Size) of the Project and the Actual Avg. Productivity, both measured in terms of CFP. If the Total Planned Size increases, but the Avg. Productivity remains the same, more Budget should be allocated to the Project.

Apart from possible scope creeping effects, the economic goal for the project would be to keep (or improve) the Average Productivity of the last Performance Baseline agreed between all the stakeholders.

#### 4. COSMIC-based Earned Value assessment

It's worthwhile investigating the contribution that the COSMIC method would provide in producing a typical Project Management assessment (as well as in Agile project management such as in the example carried out in previous sections) such as the Earned Value Analysis (EVA) [1].

EVA results to be fully applicable and interestingly straightforward in Agile projects measured with COSMIC – the main benefit being the availability of a measurement unit for the software product to support numerical calculations, while being standard, accurate and not-bounded.

Recalling the PoC factor, defined in the previous section (eq. 1), the BCWP (Budgeted Cost of Work Performed) can be defined as:

$$BCWP = BAC \times PoC = BAC \times (Size_{done} / Size_{total}) \quad (2)$$

where BAC (Budget At Completion) is the Overall Budget of the Project.

The BCWS (Budgeted Cost of Work Scheduled) can be expressed as:

$$BCWS = BAC \times (Size_{planned} / Size_{total}) \quad (3)$$

where  $Size_{planned}$  represents the Size planned for implementation up to now (i.e. at the  $i^{th}$  iteration). Even for Performance Indicators, like CPI (Cost Performance Index) and SPI (Schedule Performance Index), there are simple formulas (see [17] for an in-depth discussion on this topic, except that Story Points are to be replaced by COSMIC Function Points):

$$CPI = BCWP / ACWP = (BAC / Size_{total}) \times (Size_{done} / ACWP), \text{ or} \quad (4a)$$

$$CPI = \text{Actual Avg. Productivity} / \text{Baseline Avg. Productivity} \quad (4b)$$

$$SPI = BCWP / BCWS = (Size_{done} / Size_{planned}) \quad (5)$$

where ACWP (Actual Cost of Work Performed) is the Effort spent for  $Size_{done}$ .

In the our example, the Baseline Avg. Productivity would be 0,0375CFP/ph (whereas the values from Tab. 2 are taken as the latest agreed Performance Baseline) and the Actual Avg. Productivity would be the actual value measured, at any time during the project, as  $Size_{done} / ACWP$ .

CPI values lower than 1 would indicate more costs than expected, while SPI values lower than 1 would indicate more time than expected.

## 5. COSMIC-based Change Management in Agile Software Development

A key feature of the Agile Software Development is the openness to change [3], or the ability to refine detailed requirements as development proceeds (there isn't any requirements baseline frozen at the beginning of the project), and also the ability to make changes on portions of software that is already implemented.

This adjustment is due to the continuous feedback that iterations give, both to end users and developers, and the ensuing learning cycle that is triggered during development ([16], "Amplify learning", or the "IKIWISI" syndrome: "I'll Know It When I See It"). Indeed, only with this opening users/customers can obtain what they actually want, at the end of a development cycle; or, at least, the best ROI in terms of features most important for them, against the budget spent [17].

As shown by the CHAOS Chronicle [9], a more traditional approach, based on detailed up-front specifications and waterfall development cycle, would result in 18% of cases in failure (project not completed) and in 53% of cases in serious issue (project severely late and/or over budget). In the remaining 29% of cases, about half of the features developed (by contract) will reveal never used (or rarely used) by end users. In the light of these data, rework for changes in an Agile approach, appear as an acceptable price to pay for the end-user satisfaction, possibly in exchange for less important features. However, it's worth measuring and monitoring the amount of rework over the entire development, and to this aim the COSMIC method can be very useful.

The COSMIC method allows us to distinguish the Functional Size Developed ( $Size_{DEV}$ ), that corresponds to the effort made by the supplier, vs. the Functional Size of the Product ( $Size_{PROD}$ ), which measures the net result obtained by the customer, by applying, for each iteration, the model used in maintenance projects.

In our example, suppose that for the third iteration the customer requires changes to Functional Processes numbered "7", "8" and "4", with higher priority than some implementations initially planned. The Backlog would be as in Fig. 3:

Changes to  $FP_7$  consist of the addition of 2 new data movements, the modification of 2 data movements and the cancellation of 1 data movement already implemented; the overall size of the process increases by 1 CFP.

Changes to  $FP_8$  consist of the modification of 3 data movement already implemented; the size of the overall process remains unchanged.

Changes to  $FP_4$  consist of the modification of 2 data movement and the cancellation of 1 data movement already implemented; the overall size of the process decreases by 1 CFP.

Functional Process	Size (CFP)			Priority/Iteration
	ADD	CHG	DEL	
F.P. 1	8			1
F.P. 2	10			1
F.P. 3	8			1
F.P. 4	6			2
F.P. 5	4			2
F.P. 6	4			2
F.P. 7	8			2
F.P. 8	8			2
F.P. 7 v2	2	2	1	3
F.P. 8 v2	0	3	0	3
F.P. 4 v2	0	2	1	3
F.P. 9	8			3
F.P. 10	6			3
F.P. 11 v1	5			3
...	...			...

**Figure 3:** Product Backlog with explicit Changes (Added, Changed, Deleted data movement, or their size) to Functional Processes.

In this case the changes would not contribute to the Size of the Product, but only to the Size Developed. The calculation for the Functional Size Developed is as follows:

$$\text{Size}_{\text{DEV}} = \text{Size}_{\text{ADD}} + \text{Size}_{\text{CHG}} + \text{Size}_{\text{DEL}} \tag{6}$$

where the terms of the sum are the totals of the corresponding columns in Fig. 4, and for the Functional Size of the Product ( $\text{Size}_{\text{PROD}}$ ):

$$\text{Size}_{\text{PROD}} = \text{Size}_{\text{ADD}} - \text{Size}_{\text{DEL}} \tag{7}$$

For iteration 3 of the example above, we would obtain:

$$\text{Size}_{\text{DEV}} = 21 + 7 + 2 = 30 \text{ CFP,}$$

and

$$\text{Size}_{\text{PROD}} = 21 - 2 = 19 \text{ CFP}$$

A useful indicator for the rework rate is also given, expressed as:

$$\text{Rework} = (\text{Size}_{\text{DEV}} - \text{Size}_{\text{PROD}}) / \text{Size}_{\text{DEV}} \tag{8}$$

The Rework value should be updated and reported at each iteration. Obviously, the rework rate will increase in those cases where the requirements are unclear and/or unstable, but exactly in those cases (as stated above), a more traditional approach would create even greater problems, including, ultimately, the project failure at all.

When monitoring the different Sizes (planned, developed, product), a chart such as the one shown in Fig. 4 is more useful than the Burndown Chart. (The correct value to be compared with the Planned Size is that of the  $Size_{DEV}$ .)

While the removal of Defects, which contributes neither to  $Size_{DEV}$  nor to  $Size_{PROD}$ , is an extra cost borne by the supplier, the rework due to changes on functional processes already implemented, requested by the user, should be a cost borne by the customer.

Nevertheless, the CPI (eq. 4a) could be represented both from the supplier and customer point of view, simply using as  $Size_{done}$  respectively  $Size_{DEV}$  or  $Size_{PROD}$ .

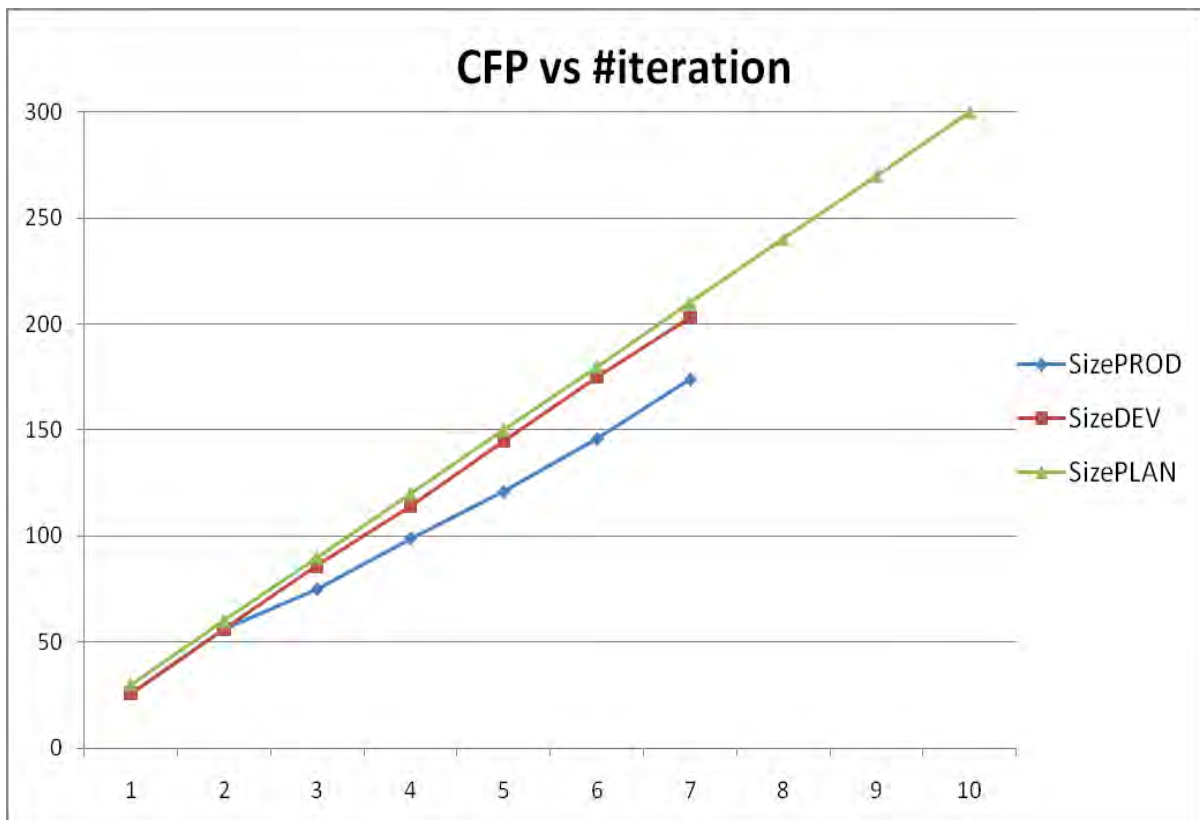


Figure 4: Size trends over Iterations.

## COSMIC-based Project Management in Agile Development & CMMI-DEV

<b>Process Area: Requirements Management (REQM)</b>	
<b>Specific Goal 1: Manage Requirements</b> (Requirements are managed and inconsistencies with project plans and work products are identified. The project maintains a current and approved set of requirements over the life of the project.)	The form of <i>functional processes</i> makes the understanding of the FUR easier for all the stakeholders. Changes and scope creep, with respect to a requirements baseline, can be measured in CFP.
<b>Process Area: Project Planning (PP)</b>	
<b>Specific Goal 1: Establish Estimates</b> (Estimates of project planning parameters are established and maintained. Project planning parameters include all information needed by the project to perform the necessary planning, organizing, staffing, directing, coordinating, reporting, and budgeting.)	Project Planning may be based on COSMIC measures and reference baseline values of Productivity and Velocity. Functional Size is the primary input to any estimation model.
<b>Specific Goal 2: Develop a Project Plan</b> (A project plan is established and maintained as the basis for managing the project. A project plan is a formal, approved document used to manage and control the execution of the project. It is based on the project requirements and the established estimates.)	A Release Plan can be organized as a Product Backlog, where the items to be developed are Functional Processes (COSMIC meaning), ordered by priority.
<b>Specific Goal 3: Obtain Commitment to the Plan</b> (Commitments to the project plan are established and maintained. To be effective, plans require commitment by those responsible for implementing and supporting the plan.)	Estimates and Planning based on standard models, such as COSMIC, and related benchmarking data are easier to share with all the participants.
<b>Process Area: Project Monitoring and Control (PMC)</b>	
<b>Specific Goal 1: Monitor Project Against Plan</b> (Actual performance and progress of the project are monitored against the project plan.)	Burndown Charts or Earned Value Metrics can be derived from COSMIC measures (planned vs "done").
<b>Specific Goal 2: Manage Corrective Action to Closure</b> (Corrective actions are managed to closure when the project's performance or results deviate significantly from the plan.)	
<b>Process Area: Supplier Agreement Management (SAM)</b>	
<b>Specific Goal 1: Establish Supplier Agreements</b> (Agreements with the suppliers are established and maintained.)	The COSMIC method can support in formalizing contracts for software supply and audits.
<b>Specific Goal 2: Satisfy Supplier Agreements</b> (Agreements with the suppliers are satisfied by both the project and the supplier.)	The initial agreement can be easily kept aligned during and at the end of the project, by monitoring the COSMIC size being developed.
<b>Process Area: Measurement and Analysis (MA)</b>	
<b>Specific Goal 1: Align Measurement and Analysis Activities</b> (Measurement objectives and activities are aligned with identified information needs and objectives.)	COSMIC contributes directly to the implementation of this Process Area. Software Functional Size is a key factor to measure, to derive other meaningful metrics.
<b>Specific Goal 2: Provide Measurement Results</b> Measurement results, which address identified information needs and objectives, are provided. The primary reason for doing measurement and analysis is to address identified information needs and objectives. Measurement results based on objective evidence can help to monitor performance, fulfill contractual obligations, make informed management and technical decisions, and enable corrective actions to be taken.	Size measures can be combined with effort data and number of defects, in order to define Productivity and Defects Density metrics. Earned Value metrics can be also produced on the basis of the Functional Size planned and developed.

**Table 3 (cont.):** CMMI-DEV Process Areas / Goals covered by COSMIC (ML 2 & 3).

<b>Process Area: Process and Product Quality Assurance (PPQA)</b>	
<b>Specific Goal 1: Objectively Evaluate Processes and Work Products</b> (Adherence of the performed process and associated work products and services to applicable process descriptions, standards, and procedures is objectively evaluated.)	COSMIC represents an objective way to evaluate requirements artifacts, at the product level. COSMIC manuals and certifications are an objective reference to assess correct measurements.
<b>Specific Goal 2: Provide Objective Insight</b> (Noncompliance issues are objectively tracked and communicated, and resolution is ensured.)	
<b>Process Area: Requirements Development (RD)</b>	
<b>Specific Goal 1: Develop Customer Requirements</b> (Stakeholder needs, expectations, constraints, and interfaces are collected and translated into customer requirements.)	A “problem domain” analysis based on COSMIC is an effective way to elicit and describe functional requirements, from the point of view of the final user.
<b>Specific Goal 2: Develop Product Requirements</b> (Customer requirements are refined and elaborated to develop product and product component requirements.)	COSMIC allows for multiple levels of decomposition. Separate Peer Components of the application can be identified and functional processes can be defined separately for each scope.
<b>Specific Goal 3: Analyze and Validate Requirements</b> (The requirements are analyzed and validated, and a definition of required functionality is developed.)	Applying COSMIC is a direct way to share and validate the functional requirements.
<b>Process Area: Organizational Process Definition (OPD)</b>	
<b>Specific Goal 1: Establish Organizational Process Assets</b> (A set of organizational process assets is established and maintained.)	The Organizational Measurement Repository, which is one of the most important Organizational Process Assets, is largely based on COSMIC measures.
<b>Process Area: Technical Solution (TS)</b>	
<b>Specific Goal 1: Select Product Component Solutions</b> (Product or product component solutions are selected from alternative solutions.)	An analysis performed with COSMIC provides an excellent basis to effectively start the development.
<b>Specific Goal 2: Develop the Design</b> (Product or product component designs are developed.)	
<b>Specific Goal 3: Implement the Product Design</b> (Product components, and associated support documentation, are implemented from their designs.)	
<b>Process Area: Verification (VER)</b>	
<b>Specific Goal 1: Prepare for Verification</b> (Preparation for verification is conducted.)	The COSMIC software model, based on functional process, provides an excellent basis for the definition of the Test Cases and Test Procedures (e.g. at a decomposition level where Peer Components are verified separately).
<b>Specific Goal 2: Perform Peer Reviews</b> (Peer reviews are performed on selected work products.)	
<b>Specific Goal 3: Verify Selected Work Products</b> (Selected work products are verified against their specified requirements.)	
<b>Process Area: Validation (VAL)</b>	
<b>Specific Goal 1: Prepare for Validation</b> (Preparation for validation is conducted.)	The COSMIC software model, based on functional process, provides an excellent basis for the definition of the Acceptance Tests of the Product (at a decomposition level where the Application is treated as a whole).
<b>Specific Goal 2: Validate Product or Product Components</b> (The product or product components are validated to ensure that they are suitable for use in their intended operating environment.)	

**Table 3:** CMMI-DEV Process Areas / Goals covered by COSMIC (ML 2 & 3).

## 6. Process Improvement – Mapping of FSM on CMMI-DEV

The COSMIC method can clearly support any process improvement framework, by providing a quantitative assessment of the (functional) requirements being examined. A coverage mapping between specific Goals of most Process Areas of CMMI Level 2 and 3 and the COSMIC method itself is suggested, to illustrate the effective support it can provide when implementing a process improvement program. We can summarize the main benefits of such mapping as the following:

- COSMIC models are powerful conceptual tools for requirements elicitation, analysis and management;
- COSMIC sizes can drive planning and project monitoring;
- by collecting historical data on productivity (based on COSMIC Function Points and person hours), a local Organizational Measurement Repository can be easily built for further improvement in estimating projects;
- COSMIC Functional Processes provide a common basis for developers, testers and users in several process areas.

Tab. 3 reports specific recommendations and remarks of COSMIC coverage on a subset of CMMI-DEV Process Areas and Goals. The key topic here is that – being COSMIC applicable in Agile approaches, it can efficiently provide a link to benefit from both Agile and Capability Maturity Models [19].

## 7. Conclusions

Usage of standard FSM method such as COSMIC results in benefits for both project management and process improvement practices. The COSMIC method adoption is highly recommended, due to its features in terms of applicability over software domains and project phases, in agile development / iterative approaches, and accurate and standardized measurement results.

In the authors' opinion, the COSMIC method, in conjunction with the CMMI-DEV and Agile methodologies, allows for a governance of software development processes and projects suitable for any organizational context and size, including medium to large ones, when “rapid” methodologies are applied as well.

This approach has been initialized in a real-world industrial framework by the authors, and further improvements will be recorded and reported at a later stage.

## References

1. ANSI: ANSI/EIA-748-B – Earned Value Management Systems, American National Standards Institute, 2007
2. Beck, K., Fowler, M.: Planning Extreme Programming, Addison Wesley, 2000
3. Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, J., Thomas, D.: Manifesto for Agile Software Development, [www.agilemanifesto.org](http://www.agilemanifesto.org), 2001
4. Boehm, B.W., Abts, C., Brown, A.W., Chulani, S., Clark, B.K., Horowitz, E., Madachy, R., Reifer, D.J., Steece, B.: Software Cost Estimation with COCOMO II, Prentice Hall, 2000
5. Chrissis, M.B., Konrad, M., Shrum, S.: CMMI – Guidelines for Process Integration and Product Improvement (2nd ed.), Addison-Wesley Professional, 2006
6. Cohn, M.: Agile Estimating and Planning, Prentice Hall, 2005
7. Cohn, M.: User Stories Applied for Agile S/W Development, Addison Wesley, 2004
8. GUFPI-ISMA, Software Benchmarking Committee: Glossario per il Benchmarking dei Progetti Software, v. 1.0, [www.gufpi-isma.org](http://www.gufpi-isma.org), 2009
9. InfoQ (Information Queue): Interview with Jim Johnson of the Standish Group, [www.infoq.com/articles/Interview-Johnson-Standish-CHAOS](http://www.infoq.com/articles/Interview-Johnson-Standish-CHAOS), August 2006
10. IFPUG: Function Point Counting Practices Manual, Release 4.3.1, International Function Point Users Group, [www.ifpug.org](http://www.ifpug.org), 2010
11. ISBSG, International S/W Benchmarking Standards Group, [www.isbsg.org](http://www.isbsg.org)
12. ISO/IEC: IS 19761:2003 – Software engineering – COSMIC-FFP – A functional size measurement method, [www.iso.org](http://www.iso.org), 2003
13. ISO/IEC: IS 14143-1:2007 – Information technology – Software measurement – Functional size measurement – Part 1: Definition of concepts, [www.iso.org](http://www.iso.org), 2007
14. Lesterhuis, A., Symons, C. (eds): The COSMIC Functional Size Measurement Method, Measurement Manual, Version 3.0.1, [www.cosmicon.com](http://www.cosmicon.com), 2009
15. Lesterhuis, A., Symons, C. (eds): The COSMIC Functional Size Measurement Method, Advanced and Related Topics, Version 3.0, [www.cosmicon.com](http://www.cosmicon.com), 2007
16. Poppendieck, M., Poppendieck, T., Lean Software Development: An Agile Toolkit, Addison Wesley, 2003
17. Rawsthorne, D., Monitoring Scrum Projects with Agile EVM and Earned Business Value (EBV) Metrics, [www.danube.com](http://www.danube.com), 2009
18. Schwaber, K., Agile Project Management with SCRUM, Microsoft Press, 2004
19. SEI, CMMI or Agile: Why Not Embrace Both!, Technical Note CMU/SEI-2008-TN-003, Software Engineering Institute, November 2008