The COSMIC Functional Size Measurement Method

Version 3.0.1

Guideline for the use of COSMIC FSM to manage Agile projects

VERSION 1.0

September 2011

ACKNOWLEDGEMENTS

Version 1.0 authors and reviewers 2011 (alphabetical order)		
Enrico Berardi*, TRS, Italy	Luigi Buglione*, Engineering IT, Italy	Juan Cuadrado-Collego, University of Alcala, Spain
Jean-Marc Desharnais, Bogaziçi Univ, Turkey: ETS Canada	Cigdem Gencel, University of Blekinge, Sweden	Arlan Lesterhuis, COSMIC Measurement Practices Committee, Netherlands
Luca Santillo*, Agile Metrics, Italy	Charles Symons*, COSMIC, United Kingdom	Sylvie Trudel*, Pyxis Technologies, Canada

* Authors and co-editors of this guideline

VERSION CONTROL

The following table gives the history of the versions of this document.

DATE	REVIEWER(S)	Modifications / Additions
September 2011	COSMIC Measurement Practices Committee	First version 1.0 issued

FOREWORD

Purpose of the guideline and relationship to the Measurement Manual

The purpose of this guideline is to provide additional advice beyond that given in the COSMIC Measurement Manual [1] on how to apply the COSMIC Functional Size Measurement (FSM) method v3.0.1 to size software developed through a so-called 'agile' project, and to use those measurements to estimate and control the project activities.

The COSMIC Measurement Manual contains the concept definitions, principles, rules, and measurement processes of the COSMIC method. It also contains much explanatory text on the concepts, plus examples of application of the method. This guideline expands on the explanatory text and provides additional detailed guidance and more examples for sizing software of an Agile project than can be provided in the Measurement Manual.

Agile projects develop software differently from traditional projects. Sizing software resulting from Agile projects raises particular issues since the software planned in the early stages of a project may not be the software that is finally developed once the Agile team and the customer reach common understanding of the true needs and the most suitable software solution. The COSMIC method for measuring new software developments and for measuring changes to existing software is perfectly suited for measuring software evolving through iterations and increments as typically found in Agile projects, without needing to adapt the COSMIC method in any way.

Intended readership of the guideline

The guideline is primarily intended to be used by expert 'measurers' (specialists or developers who measure) who have the task of measuring functional sizes of software developed by Agile projects according to the COSMIC method. It should also be of interest to those who have to interpret and use the results of such measurements in the context of project performance measurement, software contract control, project estimating, etc.

Readers of this guideline are assumed to be familiar with the COSMIC Measurement Manual, version 3.0.1 [1], the glossary [2] for COSMIC terminology and with any COSMIC guidelines relevant to the domain of the software to be measured.

For specialists in Agile methods ('Agilists') who are unfamiliar with the COSMIC method of software sizing, Appendix A explains why the COSMIC method is recommended instead of using User Story Points for Agile project measurement purposes. It also describes how best to go about obtaining more information on and learning the COSMIC method.

Introduction to the contents of the guideline

This guideline focuses on examples that illustrate how the principles and rules of the Measurement Manual should be interpreted when sizing software developed through Agile projects. The guideline is not tied to any particular agile development methodology.

Chapter 1 discusses the concepts of Agile projects in general terms and the properties that characterize the development and evolution of software applications developed through Agile projects. It treats some aspects of how Functional User Requirements (FUR) are developed through iterations and increments, as far as relevant for measurement. The limitations of Story Points for sizing and estimating are discussed.

Chapter 2 discusses the Measurement Strategy phase for software functional sizing, by considering the four key parameters of the measurement that must be addressed, namely the purpose and scope of the measurement, the identification of functional users and the level of granularity of FUR that should be measured.

Chapter 3 discusses the sizing of software through both the Mapping and the Measurement phases. In the Mapping phase, the FUR of the software to be measured must be mapped to four main concepts of the COSMIC method, namely events and functional processes, objects of interest and data groups. In the next Measurement phase, the individual data movements of the functional processes must be identified and counted, as the basis for measuring the functional size. (As almost all examples treat both the functional processes and data movements together, both phases are considered in this one chapter.). This chapter provides practical guidance and examples on applying the COSMIC Generic Software Model to size software at the level of user stories, and for the backlog, iterations and releases.

Chapter 4 discusses a number of important points on using functional size measurements at the project level for performance measurement, estimating and control of Agile projects.

Appendix A explains why it should be considered adopting the COSMIC method of measuring software size instead of using User Story Points (USP) in Agile processes.

Appendix B discusses Agile project team members' typical resistance to measurement of functional size and of performance, which seems to be an aspect of their culture, and how to deal with the issue.

TABLE OF CONTENTS

1

INTRODUCTION TO AGILITY

Agility is a working philosophy implementing values and principles of the Agile Manifesto [3], [4]. Four values are defined through the following statements:

> "**Individuals and interactions** over processes and tools
> **Working software** over comprehensive documentation
> **Customer collaboration** over contract negotiation
> **Responding to change** over following a plan
> That is, while there is value in the items on the right, we value the items on the **left** more."

Since its publication, several Agile methods have been proposed. These have been widely adopted, especially by projects that must manage highly dynamic changes of requirements requested by the customer[1], the need to re-prioritize requirements and to transform them into functioning product features in the shortest possible time, whilst all the time aiming to deliver the highest value functionalities first. There are two main categories of Agile methods:

- Agile Software Development (ASD) methods, such as Extreme Programming (XP) [5], [6], Crystal Clear [7], Test Driven Development (TDD) [8], and Domain Driven Development (DDD) [9];
- Agile Project Management (APM) methods, such as Scrum [10], FDD (Feature Driven Development) [11] and DSDM (Dynamic Systems Development Method) [12].

ASD methods tend to provide less detailed or useful advice about planning, sizing and estimating, and hence project management. In this guideline we make no distinction and refer only to 'Agile methods'.

1.1 Documenting requirements in Agile projects

Agile projects will often document their requirements in the form of "User Stories" [13]. A User Story briefly summarizes each functional request made by a customer and takes the following format:

> "As a <user type>, (part I)
> I want to <feature or functionality>, (part II)
> so that <value or expected benefit>". (part III)

For an example of a User Story, see Figure 3.1.1.

A User Story fits its purpose in an Agile context when it meets the three "C"s characteristics:

- **C**ard: the User Story fits on a (small) card: it is small enough to be expressed using the format described above;
- **C**onversation: the User Story, expressed simply, is a means to promote conversations between the customer and team members so they can reach common understanding of the feature or functionality;

[1] In this guideline we refer to the 'customer' as the function or individuals who are responsible for specifying requirements and who may make decisions on questions like priorities. The customer may also be the 'owner' of the software. We refer to the 'user' as the individual(s) who will actually use the software. In practice in an Agile project, these roles may be shared by the same persons.

- **C**onfirmation: Exactly what behaviour will be verified to confirm the scope of this User Story. This information is referred to as the 'test plan' and usually fits on the back of the card.

Mike Cohn, in his book, proposes the following "INVEST" criteria to verify the quality of a well defined User Story [13]:

- Independent: Is this high priority User Story independent of other User Stories of a lesser priority in the product backlog? This criterion aims to avoid as much as possible planning a large number of prerequisite items in early iterations instead of high priority User Stories.
- Negotiable: Does this User Story express a need rather than describe a solution? This is to foster creative ideas and exchanges between the customer and the team.
- Valuable for business: Does this User Story define a value for the customer's business? Can a user verify that the new system behaviour fulfil the customer's needs once the User Story is implemented? A User Story must be written in a language and form that are understandable by the customer and its intended users.
- Estimable: Does this User Story and any linked information allow the team to estimate the development effort, at least approximately? This is to avoid writing User Stories at a too-high level that become hard to estimate and plan reasonably or with missing information so that the team has no idea how this User Story must behave or which data it must manipulate.
- Small: Is this User Story small enough to be developed in a single iteration? This is to avoid spending several iterations developing a high-priority feature without the customer being able to see that feature until it is done, eroding the team's accomplishment feeling.
- Testable: Does this User Story have a defined functional test plan? This is to ensure that customer expectations are well understood regarding the scope of this User Story. This functional test plan should be aligned with the defined value for business (see above).

These qualities of a well-defined User Story lead very naturally to a close alignment between a User Story and one or a very few functional processes, as defined in the COSMIC FSM method.

Using the Scrum terminology and concepts, User Stories are gathered as backlog items into a 'product backlog', that may also contain other items such as defects to be fixed or some product characteristics. Backlog items are prioritized by the customer.

In industry practice, many Agile projects produce more traditional requirements documentation such as Software Requirements Specifications documents (SRS) or, more often Use Case (UC) documents. This COSMIC guideline should apply regardless of the requirements documentation format.

1.2 Current measurement and estimation practices in Agile projects

Using the Extreme Programming recommendations [5] as a basis for discussion, the following sub-sections describe some ways that measurements are used in Agile projects. We stress the weaknesses in these approaches in order to determine possible improvements.

1.2.1 Backlog items relative size and complexity

Looking at Agile projects from a measurement viewpoint, one common weakness is the confusion of software size and effort estimation. When requirements are defined as User Stories, teams roughly estimate the relative "size and complexity" of each User Story by experience and analogy, compared to other User Stories in the same project. The unit of the assigned value is the "User Story Point" (USP) [14]. USP's are often chosen to fit on a Fibonacci-like scale (0-1-2-3-5-8-13-21- etc[2]). Briefly examining all User Stories, the team decides which value is best for their "average size" User Story.

[2] Where the fourth and each subsequent number in the series is the sum of the previous two numbers.

They then assign a USP value to each User Story, reflecting its size and complexity relative to what they consider their "average" User Story.

However, the USP value cannot be considered as a reliable measure of a User Story size simply because in practice it is a value representing the relative effort to develop the User Story, not a measure of its size. It is a number that is only meaningful to the team that assigned it. From one team to another, this value, for the same User Story, is likely to vary since every team chooses which value they consider as "average". Thus, a USP is not an objective product size unit, does not comply with basic metrology concepts [15] and cannot be defined as a standard software sizing measure.

1.2.2 Velocity

Teams applying an Agile method generally measure their project 'velocity', defined by the number of USP completed per iteration[3] and use it to track the trend over time, as in Figure 1.1. When a team has worked together for some projects, they should have accumulated data from many iterations. This historical data can provide an average velocity which can be used to estimate a new set of backlog User Stories, once the team knows the associated USP values. However, not all teams collect or have access to historical data, and they often rely on an estimate of their upcoming velocity to perform their preliminary project estimate. Within an on-going project, a team may use the most recent iteration velocity to adjust the estimate for the overall project or for the next iteration.

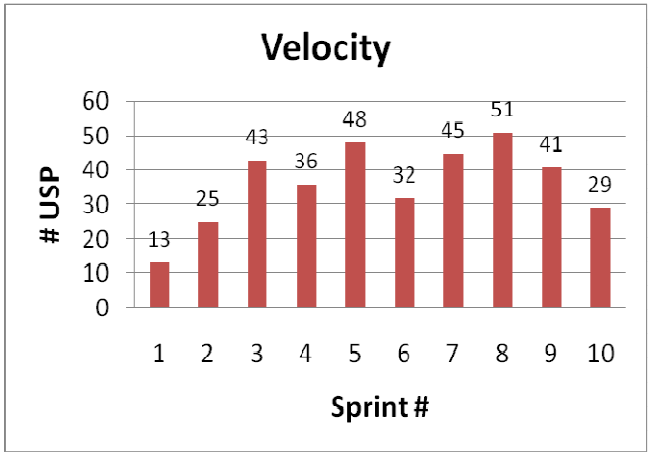

Figure 1.1 – Example of a velocity chart

However, as any one team's velocity cannot be used as a benchmark value because its USP component is not a standard measure, so velocity cannot be considered as a standard measure.

'Velocity' is an accepted term among the Agile community rather than 'productivity', which in turn is rarely accepted as a term for use mainly, it seems, for philosophical and emotive reasons. See the Appendix B for further discussion of this issue.

1.2.3 Task effort

At the beginning of every iteration, the team performs an iteration planning activity where top priority backlog User Stories are detailed into tasks of "manageable size", i.e. less than two days of effort per User Story, as per the example in Figure 1.2, until they reach the team capacity for that iteration. They can then commit to their customer to perform all backlog items they have planned that fit their capacity, in order to manage the customer's expectations in terms of functionalities to be reviewed at the iteration end.

[3] In the SCRUM methodology, an iteration is referred to as a 'sprint'.

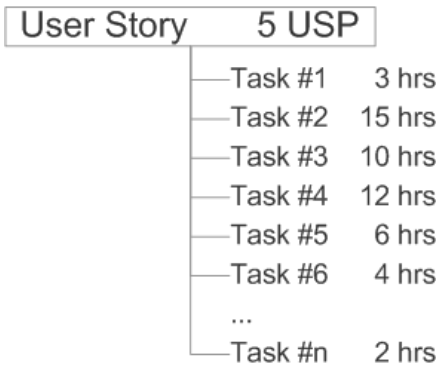

Figure 1.2 – Example of task effort estimates in hours

During an iteration, at the end of each day each team member will record, following Agile conventions, the remaining effort for every task of the iteration that they have worked on in order to produce the 'burndown' chart (see section 1.2.4).

But collecting actual effort data, which is essential to measuring actual costs and understanding project performance, is not part of the normal Agile process. Normally, an Agile team is only concerned with the remaining effort per task. Some teams can derive actual effort, if they are asked for it, by the number of team members times the number of work days in all their iterations, since they were – supposedly – fully dedicated to their project, and that they have made their planning in such a way that there should not be any overtime work. However, some roles or individuals may be assigned part-time to the team. It would be much easier to establish the project actual effort if a mechanism for recording actual effort were adopted, including recording overtime, if any.

1.2.4 Burndown chart

An Agile team makes its progress visible and publicly-available by publishing its "burndown chart", a graphical trend of the total remaining effort at the end of each day in the current iteration. 'Burndown' is calculated as the sum of the estimated remaining effort for all tasks defined in this iteration. The burndown chart can also contain a line representing the "ideal burndown", a linear representation of total effort from day one, to zero effort on the last iteration day (see Figure 1.3). This chart helps interpret the current trend and assists the team in decision-making, e.g. add a new backlog item to the current iteration if items are being developed faster than planned, or plan to work overtime or redistribute tasks or change implementation strategy if items are being developed slower than planned, or any other decision the team finds appropriate in their specific context.

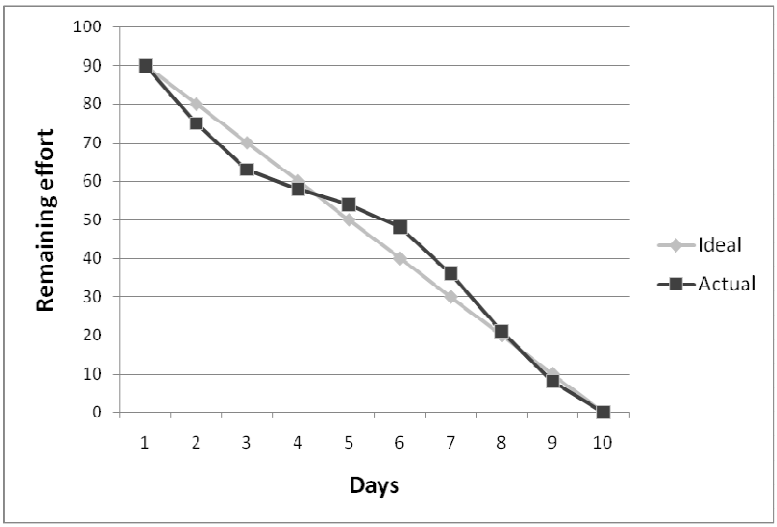

Figure 1.3 – Example of a burndown chart

1.2.5 Estimating total project effort

In an ideal world, Agile teams would prefer to estimate and commit on completing specific User Stories only as far as their next iteration. However, in the real world, the organization will often require an up-front estimate for the total project in order to release the necessary budget for the project.

From their known velocity and the estimated USP values for the backlog items, an Agile team can determine which User Stories can be included in the next iteration, and how many iterations should be necessary to complete the project. A preliminary estimate of the total project effort obtained this way assumes that several things will not change, including:

- The current content of the product backlog;
- The team's average velocity, or factors impacting the velocity that have been determined from past projects, such as:
  - Team composition;
  - Chosen technology;
  - The development process;
  - Non-functional requirements.

Provision is often made to allow for scope changes during the project, as well as any other changes the team can foresee. Estimates of duration are often expressed in number of iterations of a fixed number of weeks. Estimates of effort are usually derived from the number of team members, assuming they are dedicated full-time on the project. But there are many weaknesses in this approach, namely when some skills or individuals are required part-time and when many average or low-priority backlog items are defined at such a level that it becomes hard to estimate in USP.

1.2.6 Embrace change or r*espond to change versus following a plan*

Key features of Agile methods are the openness to change [4], the ability to refine detailed requirements as development proceeds, and the ability to make changes on portions of software that are already implemented – otherwise known as 're-work'. (There is no concept of a requirements baseline frozen at the beginning of the project.)

This flexibility to adjust is possible due to the continuous feedback and learning that iterations give to customers, users and developers (referred to as "Amplify learning", or the "IKIWISI" syndrome: "I'll Know It When I See It" [14]). Indeed, Agile proponents claim that it is only by opening the development process to users and customers in this way that they can obtain what they actually want by the end of a development cycle; or, at least it is claimed, the best ROI in terms of features most important for them, against the budget spent [16].

However, this flexibility brings with it the cost of re-work which should be compared against the well-chronicled [17] costs of problems with the traditional 'waterfall' approach to project management (projects failing or delivering over budget and resulting in features that are often never used), Agile proponents claim that rework resulting from changes in an Agile approach appears as an acceptable price to pay for the end-user satisfaction. Measuring the amount of re-work in an iteration could be useful, to ensure it does not get out of control, but is clearly difficult using USP.

2

THE MEASUREMENT STRATEGY PHASE

The Measurement Strategy phase of the COSMIC functional size measurement process requires that four key parameters of the measurement be addressed, namely the purpose of the measurement, its scope, the identification of the functional users and the level of granularity of the Functional User Requirements (FUR).

2.1 The purpose of the measurement

Among many reasons why Agile teams are likely to need software size measurements are the following:

- Measuring and benchmarking (of their software project productivity);
- Estimation, either for upfront budgeting of the software project or for iteration planning;
- Project monitoring and control;
- Internal process improvement; and
- Application governance.

2.1.1 Measuring and benchmarking project productivity

Having only User Story Points (USP) calibrated solely on effort and no common definition of what is – for instance – a USP, means it is difficult to perform internal benchmarking and impossible to exploit external benchmark data.

To adequately benchmark an organizational process such as project productivity, it is essential to be able to measure the project work-output, i.e. the size of the software produced, using an ISO standard Functional Size Measurement (FSM) method, such as the COSMIC method. Software sizes measured in units of COSMIC Function Points (CFP) can be used for benchmarking project productivity (in CFP/person-month), independently of the technology used to implement the software application and of whether or not the project used an Agile approach. Extensive benchmark data for COSMIC-measured new development and enhancement projects are available from the International Software Benchmarking Standards Group [18].

For this measurement purpose (measuring and benchmarking productivity) and all other purposes discussed in this section 2.1 (estimating, etc) a question that must be addressed is how to account for non-functional requirements (NFR). For the sake of simplifying the discussion in all parts of this section 2.1, we will assume that the projects whose productivity must be measured and benchmarked, or whose effort must be estimated etc, are all subject to similar NFR's. With this simplification, NFR's can be seen as imposing a 'fixed percentage overhead' on project effort. Software sizes measured in CFP are then a valid measure of work-output. For a discussion of alternatives to this simple approach of accounting for NFR's, see section 3.1.2.

2.1.2 Estimation

Estimating effort, at whatever level (User Story, iteration, release or project) needs, as a minimum, an estimate of the size that must be delivered and an appropriate productivity (or velocity), ideally obtained from past comparable activities. The COSMIC method is an objective way of sizing the FUR

(subject to uncertainty in those FUR) and should therefore help produce more reliable effort estimates than using subjective USP 'sizes'. For more on project estimating, see section 4.1.

2.1.3 Project monitoring and control

It is very straightforward when sizing with COSMIC to produce a product burndown chart (e.g. see Figure 4.2.2) showing the functional size measured in CFP remaining to be developed vs. the number of iterations, instead of using non-standardized USP as in Figure 1.3. Non-Functional Requirements (NFR) that may give rise to separate project tasks must not be forgotten, or else the chart may give a false impression of the degree of progress. For more on measuring project progress, see section 4.3

Similarly it is very straightforward to produce earned value analysis (EVA) results in Agile projects measured with COSMIC (see section 4.4).

2.1.4 Internal process improvement

If an organization wishes to understand and demonstrate that its processes have improved over time, it will need objective performance measures, quality data and sound data analysis. When transitioning to Agile methods, managers and team members will most probably want to know if the software process performance is improving, compared to their traditional process performance. This answer can only be provided if both 'before and after' processes collect and report data with the same unit of work-output measure, e.g. a functional size unit such as CFP. This is equally valid for new development and enhancement projects and for maintenance activities.

2.1.5 Applications governance

Organizations put in place mechanisms and measurement to perform governance of their application parameters (e.g. cost, effort, quality, scope), by looking at indicators that require an objective size measure, such as the maintenance unit cost (i.e. cost per CFP) – so they can concentrate investigation effort on portfolio applications whose value is outside expected thresholds. In that case, an objective total application size must be measured or updated after any given release, regardless of the number of projects that build or enhance the application.

2.2 The scope and timing of the measurement

Deciding what and when to measure for an Agile project needs more planning than for a project managed in the traditional waterfall manner, since there are so many possibilities. For example, will measurements be made at the iteration, release or project level, for all the software 'as a whole' or for parts being developed separately, and when will the measurements be made (before or after the work is done)? All decisions depend on the purpose of the measurement. Determining the measurement scope is particularly important,

2.2.1 In relation to the total project or to a release

An Agile project may deliver one or more pieces of software and/or may be planned as one or more releases, the overall scope of a measurement will consist of the FUR of all pieces of software to be delivered[4] by a release or by the whole project. This overall scope may have to be divided into a number of partial scopes, each of which must be measured separately, depending on the purpose.

[4] The Measurement Manual [1] distinguishes between the size of software 'delivered', which may include pre-written software such as COTS or re-usable software, as opposed to the size 'developed'. The purpose of the measurement will determine which size should be measured. However, in this guideline, for simplicity, we will not consider further the additional factors to consider when measuring delivered and developed sizes when an Agile project includes pre-written software. Here, 'developed' applies to all that was produced; 'delivered' applies to all that was finally handed over to the customer.

Examples would be when the system to be developed consists of different kinds of software, with different development environments, different operating platforms, with different expected productivities. If the purpose of the sizing is for estimating, there would then be no point in adding the sizes of these different types of software. When the software is developed over different layers or components, the estimator must also take into account that different productivities may apply.

Figures 2.2.1.1 to 2.2.1.3 show examples of various possible measurement partial scopes (all the functionality delivered by the activities within a dotted box). The timing of the measurement could be before starting to develop the functionality in a box (for estimating purposes) and/or on completion of the work (to measure the actual size delivered).

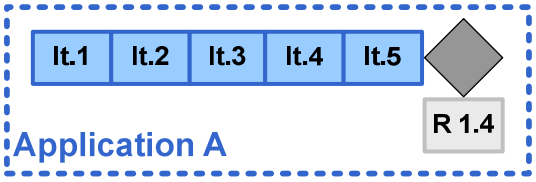

Figure 2.2.1.1 - Simple Agile (multi-iteration) development project: one release, one single application

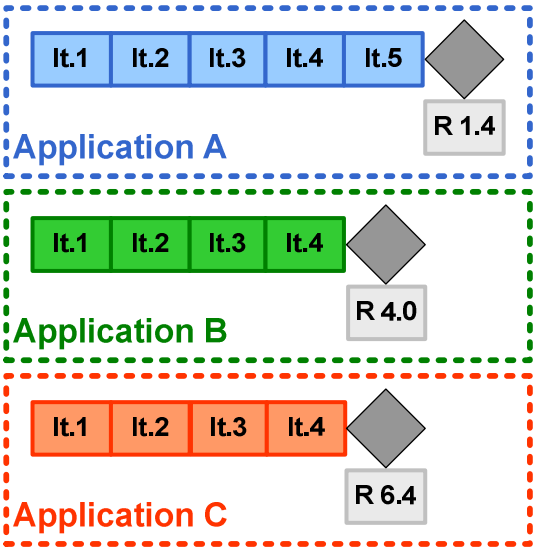

Figure 2.2.1.2 - Project to enhance several applications to be released simultaneously; measurements required for each application on completion of the releases

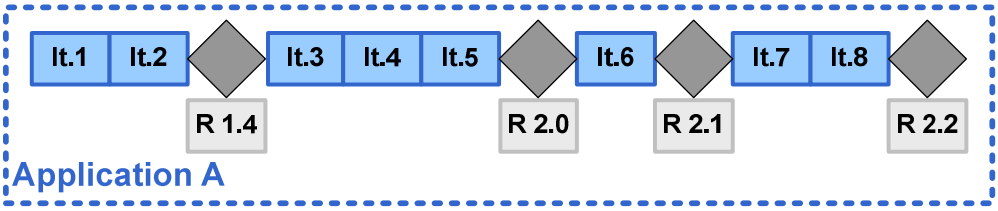

Figure 2.2.1.3 - Project delivering several releases for one single application; measurement required only on completion of the whole project

For a project or a release within a project, COSMIC size measurements may be needed, and thus measurement of partial scopes defined, at the times shown in the table below, depending on the purpose of the measurement.

Purpose	Timing of the measurement within each overall or partial scope, as appropriate	Must be known before the measurement begins	What to measure
Initial total project/ release effort estimating	Functionality to be delivered by the project or release, very early in the process	List of backlog items; assumed productivity; some estimate of the contingency for possible requirement changes	The total functional size to be delivered, maybe using an approximate variant of the COSMIC method [19]. An accurate measurement is probably impossible at this stage.
Project or release monitoring and control	Project/release functionality at intervals, e.g. end of each Iteration	Initially-estimated functional size, budget, schedule	Functional size delivered or enhanced since previous measurement[5].
Benchmarking and/or internal process improvement	Functionality developed or enhanced by the project or release	That the project is completed and the software delivered	Functional size delivered (if new development) or size of the enhancement (see section 3.2)
Application governance	Application functionality on completion of the project or release	Application functional size at start of the project or release	Net functional size increase on completion of the project or release (= added minus deleted CFP). Compute new application size.

2.2.2 In relation to iterations

In the Measurement Strategy phase, the decisions to be taken in relation to measuring iterations are:

- Will we measure functional size and use it to estimate effort at the backlog User Story and/or iteration level?
- Will we use past measurements of velocity and/or compute and refine velocity as the project proceeds?
- Will we measure re-work and/or earned value?
- How will we account for non-functional requirements (NFR) and how will we estimate contingency?

For the measurement of User Stories and iterations, see Chapter 3.

2.2.3 Summary

In practice, the most important measurements will be at the 'whole software' level (overall or partial scope, as appropriate) for initial project estimation, etc and at the User Story level for estimating, planning and control of iterations. All measurements shown in the table above can be obtained by adding up estimated or actual sizes measured at the User Story level, always subject to the COSMIC rules on aggregation of sizes (see the Measurement Manual [1], section 4.3). Obviously for these measurements to be useful, the actual effort must be recorded corresponding to each size that is estimated or measured.

[5] To measure real progress, calculate the 'percentage of completion' as in section 4.3. To do this, the initially-estimated size must be corrected for changes required and agreed by the customer. See section 3.2 for the calculation of 'Size PROD AFT'', to be updated after each iteration or release. To calculate the 'work done' to date, a measure that takes into account all changes required and agreed by the customer, calculate the 'Size DEV' as in section 3.2.

2.3 The functional users of the software application

Determining the functional users of the software being measured depends on the scope of the measurement, and ultimately on the purpose of the measurement. Identification of the functional users is no different in Agile projects from those subject to any project management method. Refer to the COSMIC Measurement Manual [1] or to a guideline related to your specific application type (e.g. [20] for guidance in identifying the functional users of the software application).

2.4 The level of granularity of the measurement

The level of granularity for sizing the software (to be) produced in a backlog of User Stories and in an Agile iteration must be that of the functional processes. A User Story is almost always written at this level of granularity.

Any sizing of FUR at higher levels of granularity, e.g. for total project effort estimation early in the life of the project, is independent of any project management method. See the Measurement Manual [1] and the 'Advanced & Related Topics' document [19] for more on sizing at higher levels of granularity and for approximate sizing.

3

THE MAPPING AND MEASUREMENT PHASES

In the Mapping Phase, the FUR of the software to be measured must be mapped to four main concepts of the COSMIC method, namely to events and functional processes, to objects of interest and to data groups. In the Measurement Phase, knowing the functional processes and data groups, the individual data movements of the functional processes must be identified, as the basis for measuring functional size. As the examples treat both the functional processes and data movements together, both Mapping and Measurement phases are considered in this one chapter.

3.1 Applying the COSMIC FSM method in Agile projects

Sizing software produced by Agile projects requires exactly the same knowledge of the principles and rules of the COSMIC method as when using any other project management method. Refer to the Measurement Manual [1] and also any relevant guideline e.g. [20] for sizing the particular application type that is being developed or enhanced.

3.1.1 Sizing User Stories

The design of User Stories is beyond the scope of this guideline but the general advice is that a User Story should define a single functional process. A User Story can then be documented on a single card using a Message Sequence Diagram (MSD) as shown in the example of Figure 3.1.1. Large functional processes with many data movements may well, of course, require rather large cards. The notation of the diagrams in Figure 3.1.1 is as follows:

- The vertical line represents a functional process.
- Horizontal arrows represent data movements, labelled E, X, R or W for Entry, Exit, Read and Write, respectively. Entries and Reads are shown as arrows incoming to the functional process and Exits and Writes as outgoing arrows; they appear in the sequence required, from top to bottom, of the functional process.

If a MSD can be drawn for a functional process, which is the minimum needed to understand the user requirement, then measuring the COSMIC functional size is a trivial task.

Conversely, if there is any difficulty in measuring a COSMIC functional size, then this is almost certainly due to weaknesses (ambiguities or omissions) in the User Stories. Measuring a size in CFP provides an excellent check on the quality of the requirements artefacts [21].

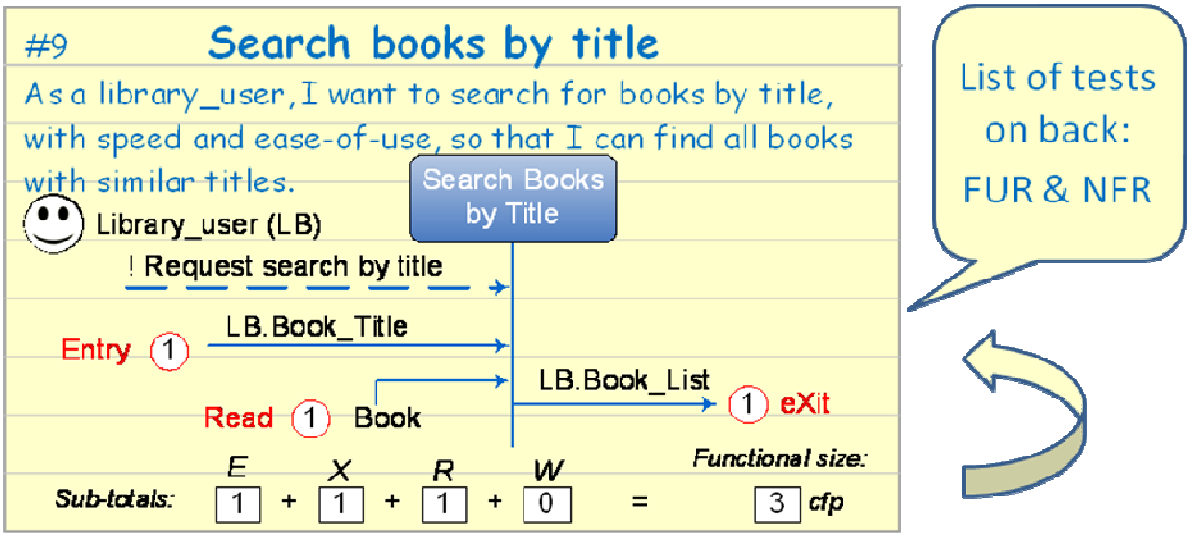

Figure - 3.1.1. Sizing a User Story with the COSMIC Method [22]

The functional size of a User Story could, of course, include changes to a previously-released User Story, to satisfy requirement change requests made by the customer. In such cases, depending on the purpose of the measurement it may be necessary to distinguish two functional sizes for each User Story:

- The 'net new functional size' (or 'product functional size') delivered to the user which excludes any functionality delivered earlier and which was changed or deleted in this iteration
- The developed functional size, representing the work done by the developers, which includes the size of functionality changed and deleted in this iteration at the request of the customer, i.e. it includes re-work[6]

The 'net new functional size' will be needed to be able to produce the 'burndown chart' (see section 1.2.4), whilst the 'developed functional size' will be needed to calculate 're-work' (see section 3.2.). Note that changes to correct defects made by the developers should not be included in either of these two sizes.

3.1.2 Accounting for non-functional requirements (NFR)

Hitherto, this guideline has assumed (see 2.1.1) that NFR do not need to be accounted for separately. In an environment where all projects are subject to very similar NFR's, it can be safe to take a functional size of the software as the measure of work-output at whatever level (User Story, iteration, release or project). Effort data will include effort spent on developing/enhancing NFR as well as meeting functional requirements. Data collected this way will then be consistent for all purposes of performance measurement, estimating, etc. NFR are then effectively ignored.

Other approaches may be needed when NFR vary in their impact on projects and need to be taken into account more explicitly. There is potentially an enormous variety of NFR, which can require different treatments. The following are two examples of possible approaches.

- Some Agile teams assign specific NFR to one or more specific User Stories and assign a 'size' in USP to each User Story. As we have seen, this is really a way of estimating effort, not size. Many types of requirements that appear initially as NFR actually evolve into functional requirements, e.g. for maintainability or usability, and can be easily sized in units of COSMIC Function Points (CFP),

[6] Note that these definitions of 'delivered size' and of 'developed size' differ from those given in the Measurement Manual [1], section 2.2.2. The latter definitions include aspects which could be used to refine the above definitions further.

just as for any other functional requirement [23]. Other types of NFR, e.g. to use a particular technology for the system cannot be 'sized' by any method; they are a constraint. They must be taken into account in the choice of productivity (or velocity) to be used to convert size to effort.

- Some Agile teams will increase the 'size' measured in USP of one or more specific User Stories to take account of an NFR. Again this is effectively a way of accounting for the additional effort resulting from the NFR. The equivalent if the User Stories are sized in units of CFP is to use a lower-than-average productivity (or velocity) figure for these User Stories to account for the additional effort resulting from the NFR. The knowledge or experience required to judge the additional effort is the same whichever approach is used.

The general point is that a team must consider NFR very carefully in many cases even before starting an Agile process. Failure to establish a mechanism to account for NFR will almost certainly result in project control mechanisms, such as shown in Figs. 1.3 or 3.2.2 giving a false impression of progress.

3.1.3 Backlog maintenance or measuring size as the project progresses

In the SCRUM approach, the 'product owner' (= customer) should continuously refine the backlog, defining the higher-priority requirements in more detail and, jointly with the team, updating their estimates. Transposing to COSMIC, the backlog items should correspond – at least for the higher-priority section of it – to single functional processes (see Section 3.1.1). Any functional process in the backlog would be assigned its *measured* size expressed in CFP.

EXAMPLE 1. Figure 3.1.3 shows a typical product backlog table before starting work on an iteration. Note that it might be preferable to distinguish 'priority' and 'iteration' since they may not coincide.

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

Figure 3.1.3 - Product backlog at early stage

In order to consistently maintain the backlog and properly account for the associated effort, an Agile team must agree a 'done' definition, i.e. its definition of what must be accomplished for a User Story to be considered as completed and deployed into production. The 'done' definition is also meant as an agreed level of quality between the team and the customer. As a practical criterion to denote an item in the backlog as "done" – where these items are expected to be listed at the level of functional processes – it can be assumed that "a functional process can be considered 'done' if at least one acceptance test for each alternative path has passed with success".

The team could, however, implement multiple (incremental) versions of the same functional processes and put them in the backlog as separate items.

EXAMPLE 2. Suppose a functional process with a size of 20 CFP. It might appear in the backlog as two distinct items: FPXv1 with a size of 12 CFP, and FPXv2 with a size of 8 CFP; FPXv1 does not implement some data movements (for example when some external interfaces are stubbed) or does not manage conditions for alternative paths, etc.

3.2 Measuring and monitoring the size of changes to software and 're-work'

3.2.1 Measuring changes and re-work

Almost inevitably in an Agile project – and intrinsic to the learning philosophy of Agile – some functionality that is considered as 'done' will need to be modified, or scrapped altogether and re-developed in later iterations. Some modifications to 'done' or implemented software may also be necessary if a large chunk of required functionality must be spread over several increments or planned to be implemented over more than one iteration. All this is referred to as 're-work'.

The COSMIC method is perfectly suited to measuring changes to functionality that has already been considered as 'done,' and thus to the measurement of re-work. It is valuable to track the re-work percentage as a project proceeds, since a high value (or higher than planned value) may indicate unstable requirements and inefficiency that needs team attention.

The COSMIC rules for measuring changes are given in the Measurement Manual [1], section 4.3 and 4.4. In the Measurement Manual, these rules are described in the context of required changes to existing software, as in an enhancement project or in maintenance activities. These same rules can be applied to changes made during the normal course of an Agile project. The rules are therefore described again here as they can be applied in the context of an Agile project.

The COSMIC method allows us to distinguish the functional size developed (Size $_{DEV}$), corresponding to the effort spent by the developers on all the work and re-work that they do, versus the functional size of the product (Size $_{PROD}$), which measures the cumulative net size of the product delivered to the customer.

The calculation for the functional size developed (Size $_{DEV}$), summed at the level of an increment, iteration, release or project is as follows:

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

$$\text{Size}_{REWORK} = \text{Size}_{CHG} + \text{Size}_{DEL}$$

where DEV = Developed, Add = Added, CHG = Changed (or Modified) and DEL = Deleted. Figure 3.2.1. shows an example of the data collection for this calculation

The functional size of the product after a change (Size $_{PROD\ AFT}$) is:

$$\text{Size}_{PROD\ AFT} = \text{Size}_{PROD\ BEF} + \text{Size}_{ADD} - \text{Size}_{DEL}$$

where Size $_{PROD\ BEF}$ is the size of the product before the change.

For the rules on measuring changed (or modified) functionality, see the Measurement Manual [1], section 4.4.1.

The rework percentage after a change is expressed as:

$$\text{Rework (\%)} = 100 \times \text{Size}_{REWORK} / \text{Size}_{PROD\ AFT}$$

where Size $_{REWORK}$ and Size $_{PROD\ AFT}$ are the cumulative values at the point when the re-work calculation is required. The rework percentage should be updated and reported at each iteration. Obviously, the rework percentage will be higher in those cases where the requirements are unclear and/or unstable.

The effort to remove defects, which contribute neither to Size $_{DEV}$ nor to Size $_{PROD}$, is an extra cost for the developers. But rework due to changes to functional processes already implemented, requested by the user, should be a cost for the customer.

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 3.2.1 - Product backlog with explicit size changes to functional processes

EXAMPLE 1. In the example of Figure 3.2.1, suppose that for the third iteration the customer requires changes to functional processes F.P. 7, F.P. 8 and F.P. 4, with higher priority than some implementations initially planned. The backlog would be as in Figure 3.2.1:

Changes to F.P. 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 F.P. 7 increases by 1 CFP.

Changes to F.P. 8 consist of the modification of 3 data movements already implemented; the overall size of F.P. 8 remains unchanged.

Changes to F.P. 4 consist of the modification of 2 data movements and the cancellation of 1 data movement already implemented; the overall size of F.P. 4 decreases by 1 CFP.

In this example, the three net changes (+1, 0, -1) would not change the functional size of the product (Size $_{PROD}$), but only change the functional size developed (Size $_{DEV}$).

For iteration 3 of the example, we would obtain:

$$\text{Size}_{DEV} = 21 + 3 + 3 + 3 = 30 \text{ CFP},$$

Before the start of Iteration 3, the size of the product (Size $_{PROD\,BEF}$) is 56 CFP. After completion of Iteration 3,

$$\text{Size}_{PROD\,AFT} = 56 + 21 - 2 = 75 \text{ CFP}$$

$$\text{Size}_{REWORK} = 3 + 3 + 3 = 9 \text{ CFP}$$

The re-work percentage on completion of iteration 3 is 100 x 9 / 75 = 12%

EXAMPLE 2. Considering the User Story example as shown in Figure 3.1.1, which is “As a library user, I want to search for books by title, with speed and ease of use, so that I can find all books with similar titles”. The customer may have defined another User Story, with a lower priority such as “As a library user, I want to search for books by author, with speed and ease of use, so that I can find all books of the same author”. This second User Story may be developed in a different iteration from that in which the first User Story is developed. And there is a high probability that both User Stories can be implemented in

the same functional process: “Search books”. In that case, the Entry movement by which search criteria are entered by the library user will be modified when the second User Story is developed. This change will be measured as an increase of one data movement in the Size $_{DEV}$, but no change in the Size $_{PROD}$.

3.2.2 Monitoring planned, developed and produced sizes

When monitoring the different sizes (planned, developed, and produced), a chart such as the one shown in Figure 3.2.2 is more useful than the burndown chart shown in Figure 1.3:

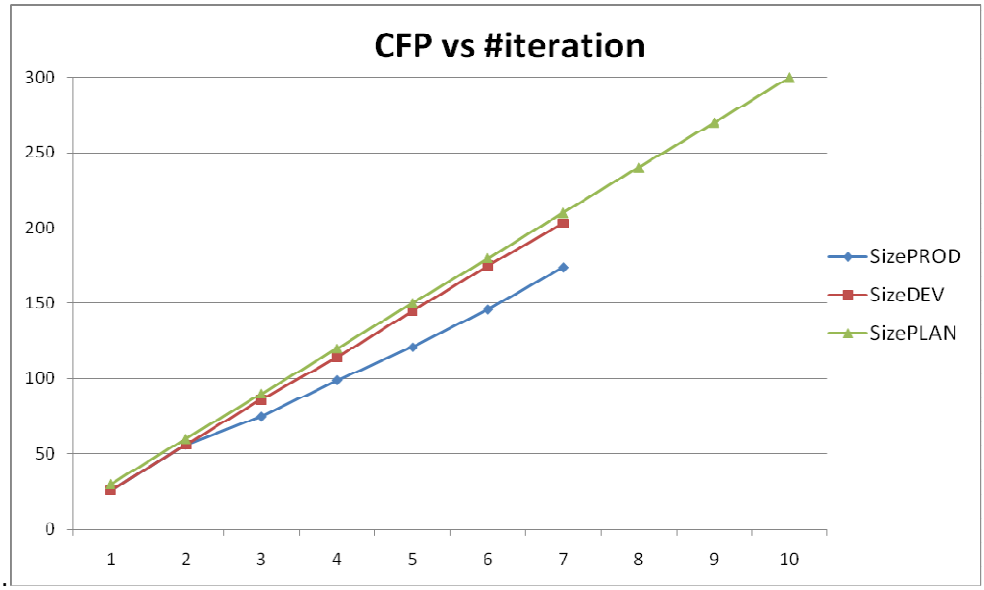

Figure 3.2.2 - Progress of product size (y-axis) over number of iterations (x-axis)

4

USING FUNCTIONAL SIZE RESULTS AT THE PROJECT LEVEL

4.1 Initial project estimation and budgeting

At a very early stage of an Agile project, the scope of the software to be developed can be defined as a list of User Stories or via any other functional requirements format that has sufficient information to obtain a list of required functionalities. With historical data on the average size of a functional process and with software process velocity or unit cost data for a given business domain and a given technology, a first estimate can be made of the software project effort.

At this point, the functional size is not “measured” but “estimated” using the estimated number of functional processes multiplied by the average size of a functional process. The estimated number of functional processes may be built up from the number actually identified plus a contingency allowance for those not yet identified.

This estimated functional size – that can be called the “initial (functional) size” – could be compared with the actual size – or “final (functional) size” – at the project end. Recording the initial and final functional sizes for all projects in a historical database will enable the average size of a FP and the software process unit cost to be refined over time.

EXAMPLE. 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/work-hour (WH) (or, assuming a conversion factor of 8 work-hours per day, and 19 work days per month, 6.08 CFP/work-month).

Thus, we would have an overall effort estimate of ca. 8000 WH. (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.)

Adopting an iterative lifecycle (with iterations of 4 weeks long) and establishing a staff of 5 FTE (full time equivalent), the project will need 10 iterations, with an average velocity of 32 CFP per iteration. The initial project estimates and performance assumptions are summarized in Figure 4.1.1.

Issue	Unit of measure	Value
Overall Effort	Work-hour	8000
Total Planned Size	CFP	320
Avg. Productivity	CFP/WH	0.04
Avg. Velocity	CFP per iteration	32
Project duration	Weeks	40

Figure 4.1.1 - Initial planned performance data

For more on estimating, see [16], [24].

4.2 Iteration planning and project re-estimation

The customer of an Agile project can stop a project at the end of any iteration for the following reasons:

- Most of the needs have been implemented successfully and the return on investment does not justify spending any more effort on remaining items;
- The forecast of cost to completion exceeds the available budget or business value.

In order to provide sufficient and good-quality information for such decisions to be made, an Agile team usually uses its velocity trend to estimate the number of iterations required to deliver the remaining backlog items. For an Agile team to monitor its velocity in CFP per iteration, the net new delivered functional size must be measured at the end of every iteration, accounting for each data movement that was added, modified, and deleted during the iteration, exactly as if each iteration represents an evolution project. Iteration planning is then an appropriate moment to measure the functional size of the backlog items that are being considered for the next iteration. With a known velocity, the team can then estimate the required effort, calculated as the size divided by the velocity in e.g. CFP/day) This can be compared with an available effort, calculated as the team capacity for that iteration, i.e. number of full-time equivalent (FTE) team members multiplied by the number of working days of the iteration. Team members can then look at major differences to understand what is missing before making any commitment on the iteration scope.

Also, if an Agile team has measured the functional size of all items in the product backlog and the velocity assumption from early estimates turns out to be quite different from their actual velocity trend (either higher or lower), the team can re-estimate the effort needed to clear the remaining project backlog and plan any actions needed with the customer, as appropriate.

EXAMPLE. Continuing with the previous example, the initialization of the product burndown chart would look like the longer line ("Planned") in Figure 4.2.2, corresponding to the initial performance baseline assumed in the table of Figure 4.2.1.

After the first two iterations we might have the situation shown by the effective line in Figure 4.2.2: in the first iteration the team has delivered 26 CFP, and in the second 30 CFP.

Functional Area	Functional Process	Size (CFP)	Priority/Iteration	Done
F.A. 1	F.P. 1	8	1	1
	F.P. 2	10	1	1
	F.P. 3	8	1	1
	F.P. 4	6	1	2
High detail, functional size measured!	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	
	F.P. 10	6	3	
	F.P. 11	10	3	
F.A. 2	F.P. 1	8	3	
	F.P. 2	8	3	
Less detail, estimated the number of f.p. with an average size	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 4.2.1 - Product backlog at a later stage

The Product burndown chart would then look like Figure 4.2.2.

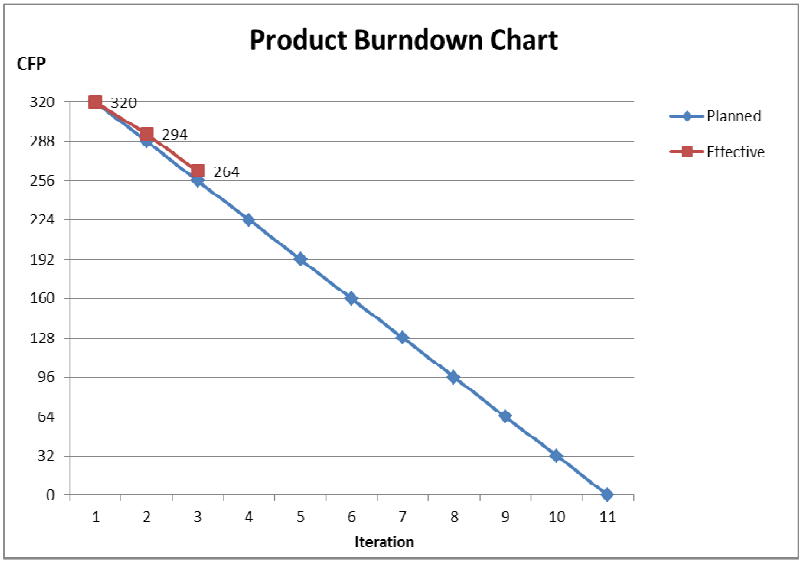

Figure 4.2.2 – Burndown chart as initially planned and at a later stage

It will be seen that a burndown chart can be used for re-planning and possible adjustment as the project progresses. For the situation shown in Figure 4.2.2, the options to be considered could include:

- The team has learnt from the first two iterations and judges that it can increase productivity and still deliver the originally planned functionality in 10 iterations;
- To deliver all the required functionality will take more effort and time than originally estimated;
- Subject to customer agreement, the same team could deliver reduced functionality in 10 iterations, i.e. the functionality is 'de-scoped'. This option could result in a new performance baseline as shown in Figure 4.2.3.

Issue	Unit of measure	Initial value	Updated value	Delta
Overall effort	Work-hour (WH)	8000	8000	---
Total planned size	CFP	320	300	-20
Average productivity	CFP/WH	0.04	0.0375	- 0.0025
Average velocity	CFP per iteration	32	30	-2

Figure 4.2.3 - Updated project performance data

For an example of the use of COSMIC to estimate user stories, see [25].

4.3 Measuring progress of the project

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

$$PoC = Size_{done} / Size_{total}$$

where $Size_{done}$ represents the size delivered ('done') up to now (i.e. at the i^{th} iteration) and $Size_{total}$ is the 'total planned size.

The total planned size may evolve as a result of changes and more detail becoming available about the requirements. This often happens in Agile projects. Actual performance data should be continually updated, controlling the evolution of the software project scope (total planned size) and the actual average productivity. If the total planned size increases, but the average productivity remains the same, the project budget will need to be increased.

Apart from possible 'scope creep' effects, the economic goal for the project would be to keep (or improve) the average productivity of the most recent performance baseline agreed between all the stakeholders.

EXAMPLE. In the updated example performance baseline of Figure 4.2.3, the percentage of completion, after the 2nd iteration would be: PoC = 56/300 = 18.7%.

4.4 Earned value analysis with COSMIC

The parameters of earned value analysis can also be easily calculated from the data collected.

Recalling the 'percentage of completion' (PoC) factor, defined in section 4.3, the BCWP (budgeted cost of work performed) can be defined as:

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

where BAC (budget at completion) is the overall budget of the project.

The BCWS (budgeted cost of work scheduled) can be expressed as:

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

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 formulae (see [26] for an in-depth discussion on this topic, except that Story Points are to be replaced by COSMIC Function Points):

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

CPI = actual Average productivity / baseline Average productivity

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

where ACWP (actual cost of work performed) is the effort spent for $Size_{done}$.

In our example, the initial average productivity would be 0.04 CFP/WH (whereas the values from Figure 4.2.3 are taken as the latest agreed and planned performance) and the actual average productivity would be the actual value measured, at any time during the project, as $Size_{done}$ /ACWP.

The CPI in this example is therefore 0.0375 / 0.04 = 0.9375 for the work done in the first two iterations. CPI values lower than 1 indicate more costs than expected, while SPI values lower than 1 indicate more effort than expected.

4.5 Process improvement monitoring

Agile teams apply continuous process improvement through a practice called "retrospective" that they perform at the end of each iteration. The aim of a retrospective is to modify the current process – people behaviour, tools, deliverables, etc. – to increase the team's efficiency at delivering functioning software. An Agile team could monitor its velocity trend in CFP delivered per iteration to better understand the efficiency of their retrospective sessions for helping to improve their process.

At the organizational level, process owners would want to monitor their process unit-cost trend using the same base measures for all projects, Agile or not. Since non-Agile projects are not likely to be applying iterative development, it is sufficient to measure sizes for the functionality delivered at the end of Agile projects, as is done for non-Agile projects according to normal COSMIC rules. Therefore, whether the project developed new software or enhanced existing software, the performance measures can be compared, regardless of the process or technology used.

REFERENCES

[1] COSMIC, 'COSMIC FSM Method v3.0.1 – Measurement Manual (The COSMIC Implementation Guide for ISO/IEC 19761: 2003)', May 2009. URL: www.cosmicon.com

[2] COSMIC, 'COSMIC FSM Method v3.0.1 – Documentation Overview and Glossary of Terms', May 2009. URL: www.cosmicon.com

[3] Agile Manifesto, 2001, URL: www.agilemanifesto.org

[4] Principles behind the Agile Manifesto, URL: http://agilemanifesto.org/principles.html

[5] Beck K. & Andres C., Extreme Programming Explained. Embrace Change, 2/e, Addison-Wesley, 2005, ISBN 0-321-27865-8

[6] Beck K. & Fowler M., Planning Extreme Programming, Addison-Wesley, 2000, ISBN 0201710919

[7] Cockburn A., Crystal Clear. A Human-Powered Methodology for Small Teams, Addison-Wesley, 2005, ISBN 0-201-69947-8

[8] Beck K., Test Driven Development by Example, Addison-Wesley, 2002, ISBN 978-0321146533

[9] Evans E., Domain-Driven Design: Tackling Complexity in the Heart of Software, Addison-Wesley, 2003, ISBN 978-0321125217

[10] Schwaber K., Agile Project Management with SCRUM, Microsoft Press, 2003, ISBN 0-7356-1993-X

[11] De Luca J., Feature Driven Development (FDD) processes, v1.3, Nebulon Pty. Ltd., 2004, URL: http://www.nebulon.com/articles/fdd/download/fddprocessesA4.pdf

[12] DSDM Consortium, DSDM Public Version 4.2, 2007, URL: http://www.dsdm.org/version4/2/public/

[13] Cohn M., User Stories Applied for Agile Software Development, Addison-Wesley, 2004, ISBN 978-0321205681

[14] Cohn M., Agile Estimating and Planning, Prentice Hall, 2005, ISBN 0131479415

[15] Abran A., Software metrics and software metrology, IEEE Computer Society, Wiley, 2010, ISBN 978-0-470-59720-0

[16] Buglione L. & Abran A., Improving Estimations in Agile Projects: issues and avenues, Proceedings of the 4th Software Measurement European Forum (SMEF 2007), Rome (Italy), May 9-11 2007, ISBN 9-788870-909425, pp.265-274, URL: http://www.dpo.it/smef2007/papers/day2/212.pdf

[17] Standish Group, CHAOS Report, April 2009, URL: http://www1.standishgroup.com/newsroom/chaos_2009.php

[18] International Software Benchmarking Standards Group, www.isbsg.org

[19] COSMIC, 'COSMIC FSM Method v3.0 Advanced & Related Topics', December 2007. URL: www.cosmicon.com.

[20] COSMIC, 'COSMIC FSM Method v3.0 – Guideline for Sizing Business Application Software, v1.1', May 2008. URL: www.cosmicon.com .

[21] Desharnais J.M., Kocaturk B., Abran A., 'Using the COSMIC Method to Evaluate the Quality of the Documentation of Agile User Stories. IWSM/Mensura Conference, Nara, Japan November 2011.

[22] Rule, G., "Sizing User Stories with the COSMIC FSM", April 2010, http://www.smsexemplar.com/wp-content/uploads/20100408-COSMICstories-article-v0c1.pdf

[23] (Example of several papers) Al-Sarayreh, Khalid T.; Alain Abran and Juan J. Cuadrado-Gallego, "A Standards-based Model for the Specification and Measurement of Maintainability Requirements", In *Proceedings of the* 22nd International Conference on Software Engineering

and Knowledge Engineering (SEKE 2010), Redwood City, California, USA, July 2010, *(ISBN 1-891706-26-8).*

[24] COSMIC, 'COSMIC FSM Method v3.0.1 - Guideline for Project Estimating using COSMIC Sizing', (under development, 2011), URL: www.cosmicon.com

[25] Desharnais J.M., Kocaturk B, Buglione L., 'Using the COSMIC method to estimate Agile user stories', Profes 2011 Proceedings, Valoir Workshop, pp. 68-73, June 2011

[26] Buglione L., Meglio Agili o Veloci? Alcune riflessioni sulle stime nei progetti XP, XPM.it, February 2007, URL: http://www.xpm.it

APPENDIX A - FOR AGILISTS – 'WHY COSMIC?'

This Appendix is included for people who are familiar with Agile methods but who have no knowledge of the COSMIC Functional Size Measurement (FSM) method. Its purpose is to explain why they should consider adopting the COSMIC method of measuring software size instead of using User Story Points (USP) in Agile processes.

A.1 Why measure using the COSMIC method rather than User Story Points?

A number expressed in USP is supposed to be a measure of the *size* of a User Story. In practice, however, it is effectively a unit of development *effort*, agreed by each Agile team for its own purposes of helping estimate and control project progress. Provided the team stays together across all the iterations and releases of a project and even for several projects, it may well be an adequate measure for these purposes.

However, USP's are not on an objective, standardized measurement scale. Typically, each Agile team decides which USP value is best for their "average size" User Story. They then assign a USP value to each User Story, reflecting its size and complexity relative to what they consider their "average" User Story. Each Agile team therefore tends to have its own unique measurement scale.

In contrast, the internationally-standardized COSMIC method of measuring a 'functional size', i.e. a size based only on what the software is required to do, in units of COSMIC Function Points (CFP), is 100% objective and repeatable. Using the COSMIC method on Agile projects will result in software size measures that can be compared across projects at any level from User Story up to the size of the whole delivered software.

The COSMIC method is also extremely easy to apply to measuring a User Story, which typically corresponds to one or a very few 'functional processes', in COSMIC terminology. Of the various existing 'functional size measurement' methods, it happens that the concepts of the COSMIC method fit perfectly with those of Agile methods. Measuring a size in CFP should take no more effort than in units of USP. In addition, the measurement process provides a quality-check on the User Story by helping to detect ambiguities and omissions.

By recording effort in units of work-hours and establishing 'velocity' in units of CFP/work-hour (rather than USP/iteration or whatever), all the usual Agile practices can be performed for e.g. managing the backlog, planning iterations, etc right up to estimating the whole project effort.

'Agilists' are therefore strongly recommended to consider using standard COSMIC size measurements in CFP instead of USP if any of the following needs arise.

- Where there is a need to enhance learning and/or to share experience across Agile project teams for purposes such as performance measurement, planning and controlling project progress, estimating, measuring process improvement, or for software portfolio governance.
- Where there are needs in software supply contracts to use an industry-standard software size measurement method for unit pricing, etc

A.2 How to find out more about the COSMIC method

If the above sounds interesting to an Agilist, he/she may either consult a person who is already experienced in COSMIC size measurement, or can study the method personally. It is simple and easy to learn. All COSMIC method documentation is available for free download at www.cosmicon.com.

We suggest starting with the ‘Method’ Overview’ document and then moving on to the ‘Measurement Manual’ [1] (the latter is available in ten languages).

APPENDIX B - FREQUENT RESISTANCE BEHAVIOURS FROM AGILE TEAM MEMBERS TO FUNCTIONAL SIZE MEASUREMENT

In the current state of practice, Agile teams tend to strictly observe the practices advocated in their respective Agile Software Development or Agile Product Management methods, almost as a matter of faith. Introducing functional size measurement may therefore meet some resistance. This Appendix describes some approaches to handle the resistance

B.1 Current practices are satisfactory for the Agile team

Agile teams that are familiar and successful with estimating and sizing their projects with USP are likely to resist changing their sizing and estimating methodology. The fact is that they can still use USP within their projects if they feel comfortable with it. There is value in the process of deciding on the USP that stimulates rich discussions and collaboration among team members. The planning and estimating exercise by the team must be seen as a means to reach common understanding of features and functionalities to develop and enhance.

FSM may, however, be required at the organization level, especially when it is a common practice within their industry sector to have their software process performance regularly benchmarked by independent experts. All that is required is that someone measures functional sizes, which requires very little effort at the User Story level. This could be done by someone outside the team to avoid disturbing them too much, who would also obtain the actual effort to contribute to the process performance database. Comparing a measured functional size against a team's consensus of the USP should also be instructive for the team. However, it may be more acceptable for someone in the team to do the measurements.

B.2 The word "productivity" is taboo

When an Agile team is told that measuring functional size of their project serves for calculating "their" productivity, they may feel judged and uncomfortable with the idea – though this reaction is not unique to Agile teams. Comparing functional size and effort to obtain a value of productivity is not a measure of individual performance but rather a measure of process performance and must very clearly be presented as such by the management. Agile teams do measure their project velocity, but that value, when measured in USP per iteration, is only meaningful to them and cannot be used for benchmarking or estimating other projects in the organization. It helps to explain to the team that measuring the functional size of their software deliverables contributes to establish an organizational unit cost that serves for benchmarking with other similar organizations, establishing estimation models for use by members of all teams to better estimate future projects, and to objectively monitor process improvement derived from their improvement actions resulting from their retrospectives.

B.3 Lack of motivation ("What's in it for me?")

Any team member may ask why spend precious project effort to measure functional size if it is not improving project delivery and quality.

It should be explained that first, effort to perform FSM is negligible once the learning curve is absorbed and that the expected (or actual) functionality is understood. Second, measuring the functional size contributes to identify ambiguities about data groups and data movements, which is of value to any team member having to code required data manipulations and movements (e.g. capture, display/print, send, store, retrieve, receive, calculate, transform, etc.). Third, FSM results can contribute to enhance

a reliable and more precise estimation model – assuming that effort data is collected and used – that the team could use for initial project estimations. This is an aspect of Agile methods that is known to be weak. And finally, knowing the average unit cost of their project may initiate different approaches to the iteration retrospective in order to ensure an increasing effectiveness.

In summary, motivation of Agile team members to apply FSM could be addressed with an awareness session or some training on the chosen FSM method.

APPENDIX C - COSMIC CHANGE REQUEST AND COMMENT PROCEDURE

The COSMIC Measurement Practices Committee (MPC) is very eager to receive feedback, comments and, if needed, change requests for this guideline. This appendix sets out how to communicate with the COSMIC MPC.

All communications to the COSMIC MPC should be sent by e-mail to the following address:

mpc-chair@cosmicon.com

Informal general feedback and comments

Informal comments and/or feedback concerning the guideline, such as any difficulties of understanding or applying the COSMIC method, suggestions for general improvement, etc should be sent by e-mail to the above address. Messages will be logged and will generally be acknowledged within two weeks of receipt. The MPC cannot guarantee to action on general comments.

Formal change requests

Where the reader of the guideline believes there is an error in the text, a need for clarification, or that some text needs enhancing, a formal change request ('CR') may be submitted. Formal CR's will be logged and acknowledged within two weeks of receipt. Each CR will then be allocated a serial number and it will be circulated to members of the COSMIC MPC, a world wide group of experts in the COSMIC method. Their normal review cycle takes a minimum of one month and may take longer if the CR proves difficult to resolve. The outcome of the review may be that the CR will be accepted, or rejected, or 'held pending further discussion' (in the latter case, for example if there is a dependency on another CR), and the outcome will be communicated back to the Submitter as soon as practicable.

A formal CR will be accepted only if it is documented with all the following information.

1 Name, position and organization of the person submitting the CR.

2 Contact details for the person submitting the CR.

3 Date of submission.

4 General statement of the purpose of the CR (e.g. 'need to improve text…').

5 Actual text that needs changing, replacing or deleting (or clear reference thereto).

6 Proposed additional or replacement text.

7 Full explanation of why the change is necessary.

A form for submitting a CR is available from the www.cosmicon.com site.

The decision of the COSMIC MPC on the outcome of a CR review and, if accepted, on which version of the business application guideline the CR will be applied to, is final.

Questions on the application of the COSMIC method

The COSMIC MPC regrets that it is unable to answer questions related to the use or application of the COSMIC method. Commercial organisations exist that can provide training and consultancy or tool support for the method. Please consult the www.cosmicon.com web-site for further detail.