From Story Points to COSMIC Function Points in Agile Software Development – A Six Sigma perspective

Thomas Fehlmann[1], *Luca Santillo*[2]
[1]Euro Project Office AG (Switzerland), [2]Agile Metrics (Italy)

thomas.fehlmann@e-p-o.com, luca.santillo@gmail.com

Abstract:

The spreading of agile methodologies in software development raises the question of how to measure requirements once more, as it happened in 'traditional' software industry development approaches decades ago. The difference is that requirements are not known in advance but detected as User Stories while iterating and enhancing the software product from one agile 'Sprint' to the other.

Some authors – promoting best practices for agile software development – propose Story Points to size User Stories (e.g., Scrum, with Story Cards), yet not combined with base project estimation. Story Points are not standardized, thus leading to eventual misconceptions and quantitative differences among practitioners and domains. The uncertainty implied in such approach can therefore propagate to any estimate based on it, not to mention the difficulty in accurately tracing requirements and their variation over the project and across project iterations.

This work investigates benefits from adopting a standardized Functional Size Measurement (FSM) method, such as COSMIC Function Points, in place of Story Points. Using a Transfer Function (from the Six Sigma practice) that transforms size into effort spent within a particular agile team, defect density prediction can be made using sensitivity analysis.

Keywords

Agile software development, Story Points, COSMIC Function Points, Six Sigma, Transfer Function.

1 The Agile Development Cycle – An Introduction

Software measurement is still a key factor in software estimation, even in modern development methodologies such as Agile Software Development. This work investigates the adoption of a standard measurement method within the agile framework. The agile development cycle selected for the following sections is Scrum, a well-structured methodology with defined processes and ceremonies [16]. The focus on Scrum is for convenience; in fact, the Six Sigma techniques presented here can be used for all agile development approaches that derive from the ideas presented originally in the Agile Manifesto, and made popular by Kent Beck [2].

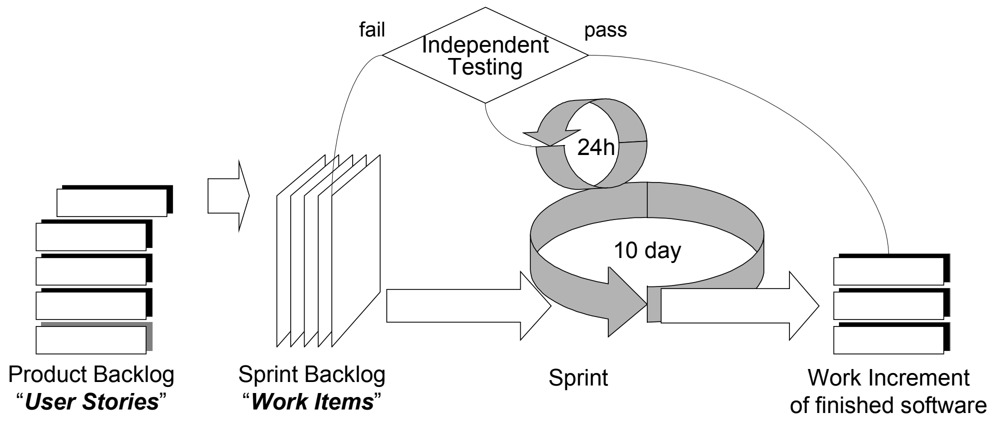

Fig. 1: Scrum Process Simplified

1.1 Scrum Basics

There are three basic roles in Scrum – the *Product Sponsor*, the *Scrum Master*, and the *Scrum Team*. Additionally, two roles are necessary – the *Independent Tester* and the generic *Stakeholder* – as well as three artifacts: the *Product Backlog*, the *Sprint Backlog*, and the *Burn Down Chart*.

The Product Sponsor represents the customer. He manages the Product Backlog, the list of *User Stories* that the Stakeholders want to be implemented by the Scrum Team. The Scrum Master moderates the team; he is representing the software development organization. The Independent Tester and his team perform independent testing for each daily build, to immediately identify pieces of code that need repair. According to the principle of Test-Driven Development (TDD) – i.e., writing test case before any line of code – every User Story has Test Stories and defined Test Cases before being considered for implementation [9].

In Scrum, development is time-sliced. Each slice is called a *Sprint*. It has fixed length (typically 30 days) and shall not be extended. The Product Sponsor together with the Scrum Master and the Scrum Team prioritize the Product Backlog. They assign a sizing unit called *Story Point* to each User Story. Story Points are assigned to User Stories using a paired comparison process, without any standardized procedure.

With the help of Story Points, the Scrum Team estimates which User Stories from the Product Backlog they can implement in each Sprint. The User Stories selected for a Sprint are further divided into *Work Items*, driven by Test Cases, by non-functional quality and architectural requirements. These Work Items constitute the Sprint Backlog. User Stories can be split over more than one Sprint.

1.2 Planning the Development

The Product Owner builds the Product Backlog – the list of the client's needs, expressed and/or written in the form of User Stories [16]. Moreover, this list is prioritized according to client necessities:

- the more business value,
- the less costs,
- the higher learnings,
- the lesser risks.

These criteria define the priority of the User Story on the Product Backlog [11].

Before starting a Sprint, the Scrum Team, the Scrum Master and the Product Owner perform the planning in a ceremony called the *Sprint Planning Meeting*. During this meeting, the stories with higher priorities are thoroughly discussed and understood by all involved in the process. In order to stir discussion, the *Story Points* technique the is used. Story points means the team agrees on assigning a sizing number to the User Story; however, story points are not standardized as a metric, and two agile teams assigning story points will not assign the same number.

For prioritizing user stories, *Pairwise Comparison* is typically performed. If validated with the Analytic Hierarchy Process (AHP) [13], it may be regarded as an early estimation technique, although this approach does not provide replicable results or standard measurement values [14].

The Scrum Team then takes these stories and breaks them down into Work Items, representing the activities needed in order to get each story done. These Work Items constitute the Sprint Backlog.

1.3 Planning Sprints

For planning purposes, the question arises – how many Work Items can be done during a time-boxed sprint? To get estimates for the Work Items, agile methodologies propose *Planning Poker*, a ceremony based on some judgment decision technique, such as the Delphi technique – not a method, however, since it's not standardized as well.

The developers are asked for estimates; they write them on some card, and uncover them all together. Discrepancies are resolved in team discussion. This method has the invaluable advantage involving all people and all sort of experiences available in the team for getting estimates. At the same time, developers understand the problem better when discussing estimates and decomposition of User Stories into Work Items.

It is worth noting that in the current agile framework, estimates have nothing to do with Story Points; <u>sizing and estimating is strictly separated</u>, according to

Cohn [5]. This is certainly correct since story points from two different sprint planning meetings are not comparable, and thus no benchmark can be established for studying the relationship between story points and effort estimates.

The decomposition of user stories into work items relies on the developers' ability to understand the product sponsor's business. The main problem when doing this kind of decomposition is that the sponsor must trust both the professional skills and the business domain expertise of developers when judging whether proposed work items are really needed and contribute to business goals, and for sorting out gold–plating and efforts waste. This is another problem with Agile Development that is addressed in [8].

1.4 Tracking Progress

A fundamental problem of tracking development progress is that developers and customers talk about different things when they declare some Work Items as "completed". Usually, a developer does not include application testing into what he thinks is completed; a customer however expects completed items not only tested but installed and integrated as well. Agile methodologies address this problem by daily builds. Every night a working instance of the product is build that can be used immediately for each completed feature. Focus on testing is a precondition for this. Most authors recommend independent testing ([1]); developers enjoy the "green bar" of JUnit for asserting the product after code changes ([9]).

These daily builds – and, the Test-Driven Development (TDD) – give agile development methods a definite advantage over traditional waterfall methods. However, daily builds do not allow predicting of the next milestone when a product release is completed. The reason is that some builds add critical functionality or integration to the product, a big jump forward, while other builds just add bug fixes and small increments. Thus progress in software development is not linearly dependent from builds.

Consequentially, progress is measured in terms of Story Points completed, bug fixes performed, and budget exhaustion rate. None of these metrics are related to how far the product is away from meeting the customer's business goals.

1.5 Validation and Entering Next Cycle

At the end of the Sprint an increment of the product is delivered. Two ceremonies are performed, the *Sprint Review*, where the potentially deliverable product increment shall be accepted by the Product Owner, and the *Sprint Retrospective*, where the team discusses improvements that can be done during the next Sprint. These improvements range from changes in artifacts to better personal collaboration and communication.

1.6 Close the Development Project

The development project closes when the Product Backlog is empty, or the sponsor wants to move the product into maintenance phase or development focus changes from new features to feature enhancements and bug fix requests that might have piled up.

1.7 The Need for Measurements

The stakeholders in agile software development need software metrics that are possibly similar to those used in traditional software development. They need measurements for:

– development effort related to the remaining budget;
– sizing measured accordingly with the "backlog size";
– quality measures, such as user/customer satisfaction.

Without proper measurements, the advantage of using agile development methods against traditional methods remains an unproven claim. Any kind of benchmarking is not possible, and two development teams cannot compare against each other. For mission-critical software, or for competitive software, this is not good enough, and makes sponsor reluctant to take full advantage of agile development methodologies.

Measuring software already during the software development cycle in turn has no relevant disadvantages. It hurts nobody and nothing, especially not in an environment where people are empowered to perform agile software development.

2 Functional Size Measurement Methods – A Brief Overview

Software developers are interested in how much data crosses application boundaries and how this can be combined to produce the expected system response. It does not matter whether such data is persistent or volatile, but it matters how many interactions are needed between system components to produce a response.

2.1 COSMIC Functional Size

The COSMIC measurement method assumes as base functional components – and therefore measures – four sub-types of data movement: Entry, Exit, Read and Write, each of which includes specific associated data manipulation [10]. Data movements can for instance be derived from items documented in sequence diagrams according the RUP Methodology.

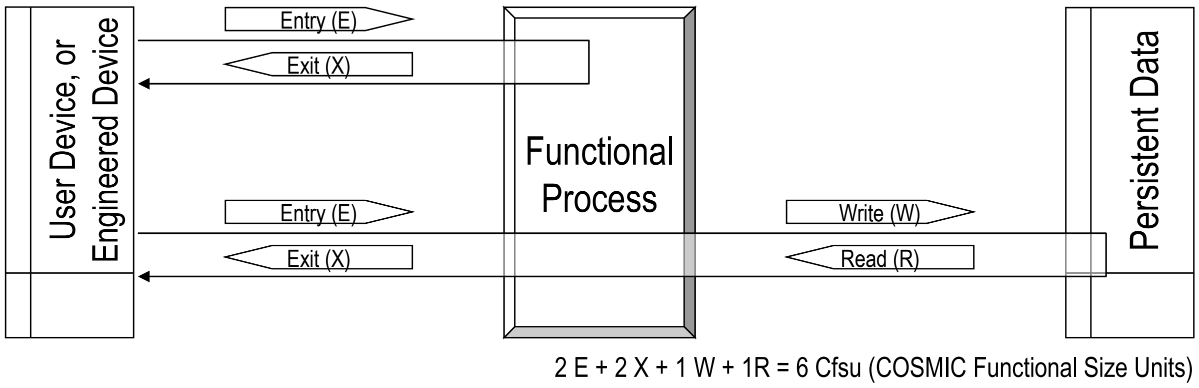
User Device, or Engineered Device
Entry (E)
Exit (X)
Entry (E)
Exit (X)
Functional Process
Write (W)
Read (R)
Persistent Data
2 E + 2 X + 1 W + 1R = 6 Cfsu (COSMIC Functional Size Units)

Fig. 2: Visualization of Data Movements for Counting

A unique sequence of data movements that meets a self-consistent set of *Functional User Requirements* (FUR) is called a *Functional Process*. Measurement principles and rules in the COSMIC method make sure that elementary processes are uniquely defined for any FUR(s) and clearly identified for measurement purposes.

The size of any Functional Process is derived by aggregating the sizes of the Data Movements comprised in it – this results particularly straightforward, since each Data Movement is assigned a size of exactly 1 (COSMIC) Function Point. The size of a portion of software is derived by aggregating the sizes of all the Functional Processes within that portion.

In case of software variations (that is, of changes to its FURs), the size of the change is derived per each functional process by aggregating the Data Movements that are added, changed, and/or deleted in that Functional Process.

2.2 Visualization

Visualization is also available for User Stories, see Fig. 3. The recommended practice is drawing sequence diagrams [3] on the back – or attached – to User Story cards, as pointed out by Rule in a recent note [12]. Sequence diagrams create a common understanding and allow identifying risks with the planned software at an early stage. Such kind of diagrams – either standardized or not – are of great impact in understanding and communicating FUR descriptions. Therefore, they are also very valuable – yet not mandatory – in identifying, analyzing and finally measuring Function Processes according to the COSMIC method. Given the nature of Data Movements within Functional Processes in the COSMIC method, a User Story can be easily regarded as a sequential description of Data Movements for one or more Functional Processes.

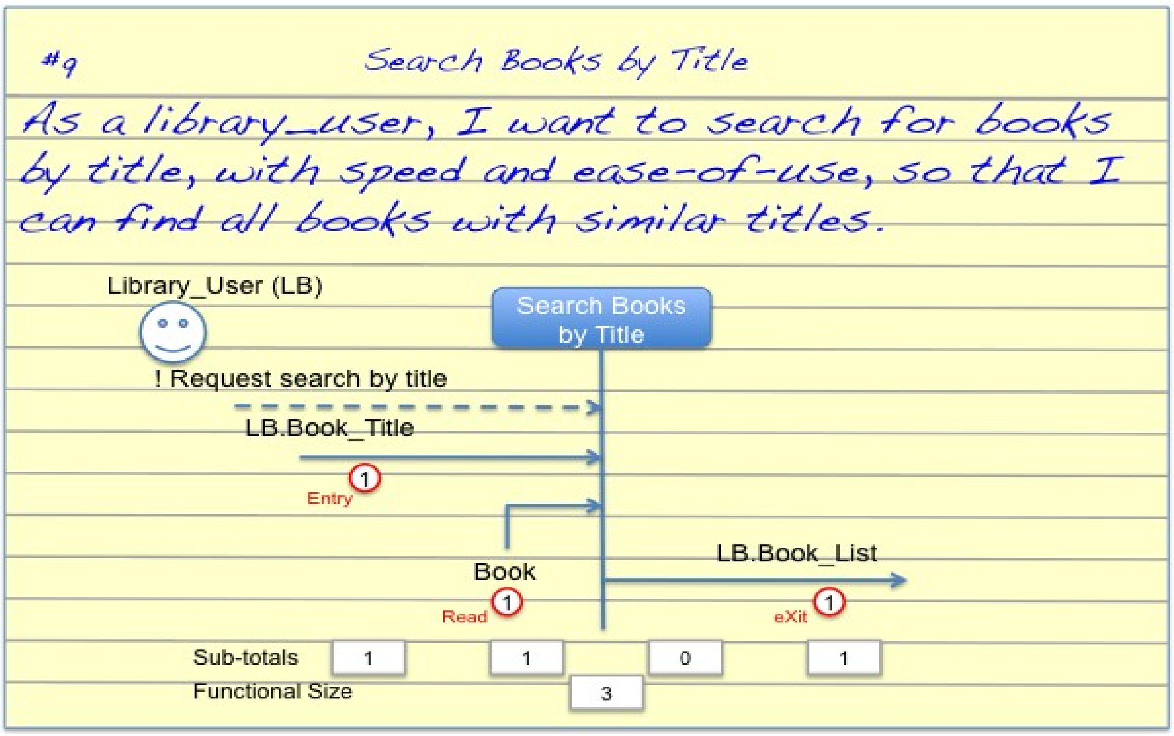
#9
Search Books by Title
As a library_user, I want to search for books by title, with speed and ease-of-use, so that I can find all books with similar titles.
Library_User (LB)
Search Books by Title
! Request search by title
LB.Book_Title
1
Entry
Book
1
Read
LB.Book_List
1
eXit
Sub-totals
1
1
0
1
Functional Size
3

Fig. 3: Sample User Story with FSM according to the COSMIC method [12].

2.3 How to use FSM in Agile?

In practice, the change required to integrate functional size measurement into User Stories used in agile methodologies is to replace the usual template for writing User Story Cards.

Normally [16] the template for User Stories reads:
As a … [**stakeholder role**] …
I want to … [**perform an action**] …
[With some frequency and/or quality characteristic] …
So that … [**description of value or benefit achieved**].

When adapting it to COSMIC, it reads:
As a … [**functional user**] …
I want to … [**respond to an event**] …
[With some frequency and/or quality characteristic] …
So that … [**useful output, or outcome, produced**].

With just these few adaptation, agile teams adopt a standardized and comparable size measurement method that yield identical results across time, across teams, across projects, and across organizations. Developers, product owners, and their customers thus have a low-cost, effective tool to use when estimating, tracking progress, and assessing value-for-money.

3 FSM and Agile Development Integration – A Six Sigma Perspective

3.1 Functional Sizing in Six Sigma

Six Sigma orientates products and services towards measurable customer values. Functional Size is an example of (quantification of) such customer value; thus FSM should be regarded as an essential constituent of every 'Six Sigma for Software' initiative [7].

To make processes measurable, Six Sigma establishes translation mechanisms between process controls and process responses. Such translations are called *Transfer Functions*.

Transfer functions translate controls $\underline{\boldsymbol{x}}$, the parameter vector that engineers use to run the process, into the responses $\underline{\boldsymbol{y}}$ that yield the business value. Process response corresponds to the views of customers; process controls to the view of engineers. – Similar transfer functions exist between the process views of users and administrators, or of developers and testers.

If the controls are able to limit the variations of the responses within tolerance range, controls are *capable*. Selecting the capable controls for the responses wanted is the Six Sigma practice called *Critical Parameter Management* [4].

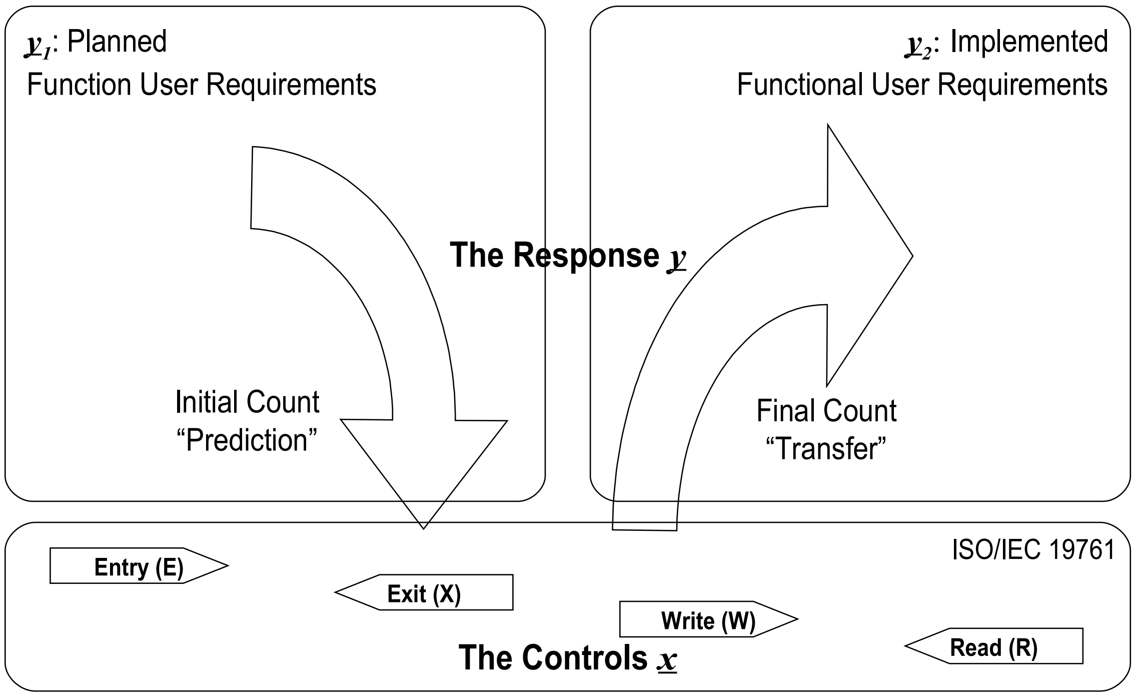
$\underline{y}_1$: Planned
Function User Requirements
$\underline{y}_2$: Implemented
Functional User Requirements
The Response $\underline{y}$
Initial Count
"Prediction"
Final Count
"Transfer"
ISO/IEC 19761
Entry (E)
Exit (X)
Write (W)
Read (R)
The Controls $\underline{x}$

Fig. 4: FSM as Six Sigma Prediction and Transfer Function

Implementing functionality is a sample transfer function: it translates a (measurement) model of the FURs (e.g., the COSMIC software model) into the relating set of implemented functional processes and their data movements by developing software (Fig. 4).

Ideally, a transfer function is also accompanied by a *Prediction Function*. The prediction function predicts which controls are needed to make the response uniquely defined and variations within tolerance.

The COSMIC method principles and rules are an example of prediction function; they define the controls, i.e., they identify the data movements in the COSMIC software model that in turn control the functional size being implemented. Between the FURs initially sized ($\underline{y}_1$), and the FURs implemented ($\underline{y}_2$), there might differences – within tolerance.

Story points are not a prediction function, since they don't identify the controls needed for a transfer function – in Six Sigma terms, the difference between predicted size and implemented size is unpredictable, thus out of tolerance; hence story points are not a measurement method.

3.2 Sensitivity Analysis for Velocity Prediction

Using a Measurement Function as the Transfer Function, that transforms the sizing controls $\underline{x}$ (COSMIC Data Movements) into implemented FURs $\underline{y}_2$, a *Sensitivity Analysis* can be performed. The sensitivity analysis is the first derivative of the Transfer Function against time.

In agile terminology, this derivative is called *Velocity*. The velocity prediction is not a number; it spans a range of possible velocities that the Agile development process possibly will produce at the end. The smaller the backlogs are, the narrower the range becomes. However, there is another very important aspect of velocity prediction that yields value to the project team: the velocity range is an indicator for the actual *Duration Risk* involved in the project: the higher the derivative the higher the duration risk.

3.3 Sensitivity Analysis for Uncertainty Prediction

The transfer function $\underline{x} \rightarrow \underline{y}_2$ does not only provide the duration risk, it yields *Uncertainty Prediction* as well, which is a fundamental factor in any software estimation model due to the high rates it can easily achieve [15]. The derivative against change in the FURs is an indication for the implementation risk.

Six Sigma is a statistical method that works with linear vector spaces. When calculating uncertainty, the linear vector space of controls are the FSM counts. It has four dimensions, the four sub-types of data movements being its four degrees of freedom. The linear vector spaces of the response contains the formulation of a FUR, be it in natural language or in a some formal requirement definition language; its degree of freedom typically has higher dimension than four.

In order to compute uncertainty, it is not actually required to change the FURs. It is sufficient to modify the number or the total size of functional processes (or equivalently, the total size of data movements) and calculate the effect it has for the process response using the transfer function $\underline{x} \rightarrow \underline{y}_2$.

These events are measurable as vectors in the linear vector space of the FURs and thus have a canonical distance defined, the length of the vector difference [6]. As a help for imagination, consider variations of FURs expressed in natural language that vary slightly in wording. Such variations are small and their vector representation are almost identical. However, when replacing fundamental glossary within on FUR, it's meaning and thus the amount or the size of functional processes required to meet the FUR changes “a bit”.

Such variations, and their rates, are measurable in Six Sigma, thus adding to FSM methods such as COSMIC a huge new dimension for process analysis – yet to be fully investigated.

3.4 Story Boards Visualization

Visualization allows the development team to detect issues and risks simply by experience with “how it should look like”. To achieve this, and at the same time to outweigh the missing discussions in the Planning Poker, we use Sequence Diagrams as a visualization instrument. In practice, the idea is to copy the User Stories and place them on a big wall (Fig. 5). Since the size of the User Story cards is not suitable for wall placement, we capture them electronically and size them accordingly such that the team has always an overall overview of the system.

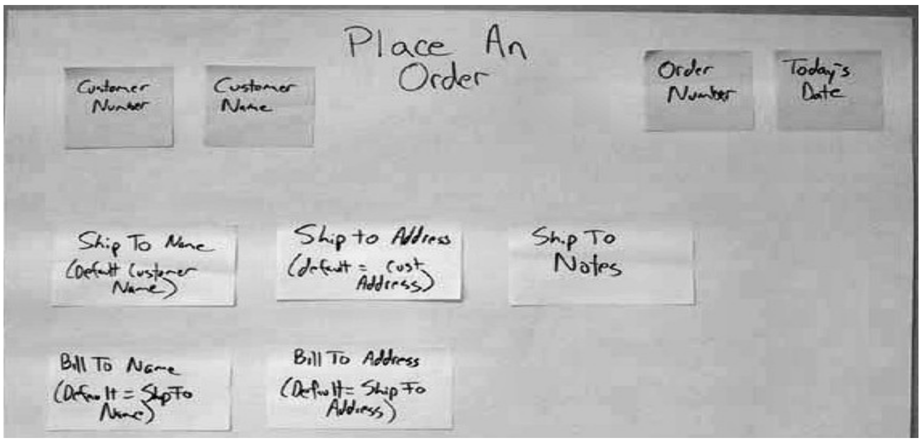
Place An Order
Customer Number
Customer Name
Order Number
Today's Date
Ship To Name (Default Customer Name)
Ship to Address (default = Cust Address)
Ship To Notes
Bill To Name (Default = Ship To Name)
Bill To Address (Default = Ship To Address)

Fig. 5: An example of User Stories in practice

The advantage of the proposed approach is that it works as well for distributed development teams; it is not necessary to meet physically for planning pokers. Measurements such as actual sizing is done 'off-line' by the project office; knowing the numbers in real-time are not really relevant for the development team. But with the agile process becoming measurable with COSMIC, both sponsors and Scrum Masters (or Project Managers) have control over the process and an early warning system as in any traditional, well-cared software development project.

3.5 Communication in the SW Project

The COSMIC measurement method in connection with structured diagrams (such as the example in Fig. 4) help agile teams to better cooperate internally, and communicate with management and sponsors – the structure diagrams would easily adapt to the evolving scope and can accommodate changes from sprint to sprint.

4 Conclusions

Merging a standard measurement method such as COSMIC with Statistical Process Control such as Six Sigma within the Agile Development framework yields several benefits and advantages.

While traditional software development methods and project cycles would typically (incorrectly) expect functional size to remain stable during a project (unless a rigorous scope management approach is undertaken), in Agile projects, Scope Managers have the ability to continually measure risk exposure, using a replicable and homogeneous measurement method (COSMIC) with statistical tools (Six Sigma). (In fact, Scope Managers should therefore become part of every agile project team, complementing the traditional role of a project office.)

The adoption of a standardized measurement method in a Six Sigma prospective does therefore provide a tool for early estimation, adaptation to measurable requirements change over iterations, and uncertainty/risk assessment (and possibly reduction) in the overall estimation process, especially in the Agile development framework.

References

[1] Aguanno, K. (ed.), Managing Agile Projects, Multi-Media Publications, Lakefield, Ontario, CA (2004)

[2] Beck, K., eXtreme Programming explained, Addison-Wesley, Boston (2000)

[3] Bell, D., UML basics: The sequence diagram – introduction. In: IBM Developer Works (2009), http://www.ibm.com/developerworks/rational/library/3101.html.

[4] Creveling, C.M., Slutsky, J.L., Antis, D., Design for Six Sigma, Prentice Hall, New Jersey (2003)

[5] Cohn, M., Agile Estimating and Planning, Prentice Hall, New Jersey (2005)

[6] Fehlmann, Th., The Impact of Linear Algebra on QFD. In: International Journal of Quality & Reliability Management, Vol. 21 No. 9, pp. 83--96, Emerald, Bradford, UK (2004)

[7] Fehlmann, Th., Statistical Process Control for Software Development – Six Sigma for Software revisited. In: EuroSPI^2 2006 Industrial Proceedings, University of Joensuu, Joensuu, Finland (2006)

[8] Fehlmann, Th., Six Sigma for Agile Software Product Development, Procs. PROFES 2010, Limerick

[9] Fowler, M., JUnit Test Infected: Programmers Love Writing Tests, found 30-Apr-2010, http://junit.sourceforge.net/doc/testinfected/testing.htm

[10] Lesterhuis, A., Symons, C. (eds.), The COSMIC Functional Size Measurement Method, Measurement Manual, Version 3.0.1 (2009), http://www.cosmicon.com/

[11] Heitor Roriz Filho, Can Scrum Support Six Sigma? In: Scrum Alliance (2009), http://www.scrumalliance.org/articles/161-can-scrum-support-six-sigma

[12] Rule, P.G., Sizing Agile 'Stories' using COSMIC, 2009, www.cosmicon.com

[13] Saaty, Th.L., Decision-making with the AHP: Why is the principal Eigenvector necessary. In: European Journal of Operational Research vol. 145, pp. 85--91, Elsevier Science B.V. (2003)

[14] Santillo, L., Early & Quick COSMIC-FFP Analysis using Analytic Hierarchy Process, Procs. IWSM-Metrikon 2000, Königs Wusterhausen (Berlin)

[15] Santillo, L., Error Propagation in Software Estimation, Procs. IWSM–Metrikon–Mensura 2006, Potsdam (2006)

[16] Schwaber, K., Beedle, M., Agile Software Development with Scrum, Prentice Hall (2008)