# Orchestrating Functional Change Decisions in Scrum Process using COSMIC FSM Method

Asma Sellami[1], Mariem Haoues[2], Nour Borchani[1] and Nadia Bouassida[1]
[1]*Mir@cl Laboratory, University of Sfax, ISIMS, BP 242. 3021, Sfax, Tunisia*
[2]*Mir@cl Laboratory, University of Sfax, FSEGS, BP 1088. 3018, Sfax, Tunisia*

Keywords: Functional Change, User Requirements, COSMIC ISO 19761, Functional Size Measurement, Scrum, User Story Description.

Abstract: Because user requirements change frequently during the software life-cycle, project managers looked at flexible methods that manage changes as more information take place. Today, agile methods such as Scrum are increasingly used in software organizations to avoid the danger of "scope creep" and project delays. Although the scrum method is gaining popularity, a change request is still poorly evaluated within an ongoing sprint. Moreover, there is a lack of a structured change evaluation approach that helps in making an appropriate decision after a Functional Change (FC) request. The main objective of this paper is to propose a decision support tool that assists the decision-makers to decide whether to accept, deny or defer a given FC request. For this purpose, we use the COSMIC Functional Size Measurement (FSM) method to evaluate a FC request. We distinguish between a FC affecting an ongoing sprint and a FC affecting an implemented sprint. Based mainly on the functional size of the proposed FC and the development progress, we provide recommendations to the decision-makers.



## 1 INTRODUCTION

The success of software organizations increasingly depends on how to manage software projects in a dynamic and unpredictable environment, where software requirements are likely to change. In fact, software products are frequently subject to continuous evolution induced by the addition of new functionality or the deletion/modification of existing ones. Multiple causes may be the reason for requirements changes (*e.g.,* missing functionality, defects corrections, etc.) (Bano et al., 2012).

Agile methods are increasingly being adopted in software organizations. They are flexible and can cope with software evolution. In fact, agile methods encourage the communication with the customer/user, who is present during each phase of the software life-cycle. Although agile methods have the reputation for coming up with flexible solutions, many promised software projects have failed to succeed. Gilb considers that 61% of agile projects end up with failure (Gilb, 2018). He also addressed the importance of using measure in reviewing development progress, re-evaluating user stories priorities, etc. This in order to align with the new customer's needs.

Over recent years, many agile methods have been proposed (*e.g.,* extreme programming, scrum, crystal, etc.). Each organization chooses the most appropriate method suitable to its work. Currently, eXtreme Programming and Scrum methods are the most popular (Dikert et al., 2016). In our paper, we selected "Scrum". In the scrum process all the participants (*e.g.,* product owner, development team, scrum master, etc.) should be keenly aware about how *big* is a change request. In fact, a change considered as "small" for the Product Owner (PO) may have significant impact on the development schedule and budget.

In practice, change requests are usually assessed by "experts" that have the required knowledge about the changed product and the process to carry out the required modifications. However, software organizations should avoid as much as possible values that are based on judgment (Abran, 2015), since there is no guarantee of their effectiveness. Thus, it is necessary to use a well defined measurement method to assess a change request.

In this paper, we introduce a new Functional Change (FC) evaluation approach applied in scrum process based on the COSMIC FSM method. Sizing a FC in terms of COSMIC Function Point (CFP) units



Sellami, A., Haoues, M., Borchani, N. and Bouassida, N.
Orchestrating Functional Change Decisions in Scrum Process using COSMIC FSM Method.
DOI: 10.5220/0006853804820493
In *Proceedings of the 13th International Conference on Software Technologies (ICSOFT 2018)*, pages 482-493
ISBN: 978-989-758-320-9

will provide a *real* evaluation of the change request (Haoues et al., 2017). This evaluation will help the decision-makers responding to a change request. To provide for these decisions, we distinguish between a change request that affects an implemented sprint and a change request that affects an ongoing sprint. Nevertheless, in scrum process, changes are not allowed after the sprint starts. However, if a change affects a user story (not yet fully implemented) in an ongoing sprint, this will cause either the sprint stop or time-wasting in implementing a user story that will certainly be changed in the next sprint as requested by the PO. The proposed approach in this paper is supported by the Functional Change Measurement Tool.

The remaining of this paper is organized as follows: Section 2 presents firstly an overview of the scrum process and COSMIC FSM method. Secondly, it discusses some related works. In section 3, we use COSMIC to evaluate a FC request. Section 4 presents the recommendations to answer a FC request. In section 5, we illustrate our approach through the case study "E-Commerce" and present our tool. Section 6 discusses several threats to validity, concludes the presented work and outlines some of its possible extensions.

# 2 BACKGROUND

## 2.1 Overview of the Scrum Process

The scrum process starts with a high-level definition of the project scope. Scrum uses the product backlog as a list of features corresponding to the Functional User Requirements (FUR). This list is prioritized by the PO to be used as an iterative input for different sprints "Iteration" (Schwaber, 2004). Thus, the active involvement of the PO is mandatory to explain, elucidate the next iteration that should be implemented and evaluate the work done. For a single sprint, four types of meetings should be held: sprint planning meeting, daily scrum, sprint retrospective meeting and sprint review meeting. The FUR to be implemented in a sprint are captured during the planning meeting; the FUR to be selected are derived from the Product Backlog and placed in a Sprint Backlog. Daily scrum meetings are held during the sprint to discuss three main questions: what has been done, what are you going to do, and what are the issues (Cohn, 2004). Each sprint is followed by a sprint retrospective meeting during which the development team reviews the sprint and decides which change will be made, and how they can improve their work processes for the next sprint.

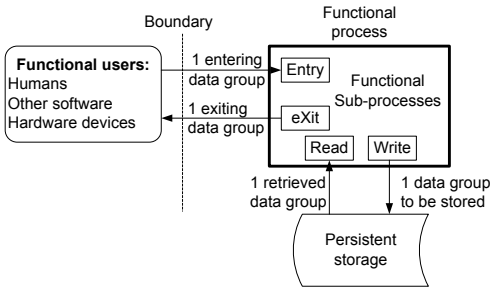


Figure 1: COSMIC data movements (COSMIC, 2017).

User stories represent a high-level of user requirements that typically express the FUR from an external user perspective. A User Story (US) is a brief statement of intent that describes the user demand. Below the US description used in the literature (Cohn, 2004).

**As a** <user type>
**I want to** <feature or functionality>
**so that** <value or expected benefit>

Typically, development team uses the User Story Point (USP) method to determine the complexity of user stories. This method has been widely criticized (*cf.,* (Berardi E., 2011), (Desharnais et al., 2011), (Commeyne et al., 2016), etc.). USP is only meaningful for a specific development team and in a specific project. It could not be used to compare the productivity of the organization with its competitor in the market. In addition, (Commeyne et al., 2016) proved that using COSMIC in agile projects gives a better results in estimating the effort needed to accomplish a US.

## 2.2 COSMIC FSM Method: ISO 19761

COSMIC measure the software functional size from the FURs. FURs represent the *"user practices and procedures that the software must perform"* (COSMIC, 2017). Each FUR involves a number of functional processes. Each Functional Process (FP) consists of a set of functional sub-processes that move or manipulate data. A data movement moves a single data group from/to a user (respectively Entry and eXit data movement) or from/to a persistent storage (respectively Read and Write data movement). One CFP is then assigned per data movement. The functional size of a FP is equal to the number of its data movements. Thus, the software size is equal to the sum of the sizes of all its functional processes.

COSMIC is the only FSM method that measure the size of a change to software. COSMIC defines a Functional Change (FC) as *"any combination of additions of data movements or of modifications or deletions of existing data movements"* (COSMIC,

2017). To measure the Functional Size of a FC, referred to as FS(FC), COSMIC recommends to attribute one CFP for each changed data movement regardless of the change type (addition, deletion, or modification). Thus, the FS(FC) is given by the aggregation of the sizes of all the added, deleted and modified data movements. The FS(FC) is at least equal to 1 CFP with no upper limits (COSMIC, 2017). Whereas, the functional size of the software after a FC is given as the *sum* of all added data movements *minus* the functional size of all removed data movements (COSMIC, 2017).

### 2.3 Related Work

Agile methods embrace change while traditional process avoid changes. To determine the perception of practitioners about product backlog changes, (Alsalemi and Yeoh, 2015) conducted a survey that identified 11 reasons for requirements change. The main reasons are: defect fixing, functionality enhancement, and design improvement. Regarding the change type, the practitioners consider that 56.1% of the requirements changes propose to add new requirements. While, 62.9% of the requirement changes in the product backlog propose to modify existing requirements.

Taking into account the importance of change management in scrum, a number of researchers have addressed the evaluation of requirements change in agile methods. For instance, (Lloyd et al., 2017) addressed the problem of requirements changes during the development process in distributed agile development. In this study, a feature tree approach is proposed with a supporting tool to help managing requirements' changes in distributed agile development. On the other hand, (Stålhane et al., 2014) proposed to analyze the impact of technical changes (*i.e.,* changes affecting non-functional requirements or project requirements and constraints), in particular, safety requirements. The safety requirements have been separated and analyzed after any change that occurs to ensure the validity of safety requirements. Two main questions have been addressed in this study: will the requirement and design affect the safety? and Will the update affect the safety?.

Table 1 summarizes the main proposals that focused on the requirements change in scrum process. We noticed that some studies focused on functional changes (*cf.,* (Alsalemi and Yeoh, 2015)) while other studies focused on technical changes (*cf.,* (Stålhane et al., 2014)). However, changes in these papers have been always considered as new requirements. Thus, no change history is recorded. However, we believe that it is important to evaluate requirements' changes and record changes history. This will certainly help during the software maintenance as well as for new software development. It guarantees a better understanding of the change and directly affects the quality of the software size measurement results (Desharnais et al., 2011). Moreover, change evaluation is usually based on experts' judgment. Whereas, experts' judgment evaluation is less transparent compared to any other technique and depends mainly on the experts' experiences. In addition, there is no way to verify the logic of their evaluation especially when there is no quantified data (*e.g.,* the change size).

## 3 FC EVALUATION IN SCRUM PROCESS

This section illustrates how to evaluate a FC in scrum process using COSMIC FSM method. Thus, we propose a detailed US description that provides the required information to apply COSMIC. Then, we present measurement formulas to measure the functional size of software products using US format. Later, we describe our method that can be used in evaluating a FC.

### 3.1 Detailed US Description

In scrum, user requirements are represented in user stories format. However, user stories represent the user requirements at a high level of details (Desharnais et al., 2011). Moreover, there is no standard representation of user stories. Hence, different user stories description formats have been proposed. For instance, (Verwijs, 2016) distinguished between two ways to break down user stories: horizontal and vertical. The horizontal breakdown divided the user stories by the type of work that is needed or the layers or components that are involved (Taibi et al., 2017). Whereas, the vertical breakdown decomposed the user stories in *"a way that smaller items still result in working, demonstrable, software"* (Verwijs, 2016). (Desharnais et al., 2011) used COSMIC to assess the quality of the US documentation. They proposed a documentation quality rating scale including five values. The evaluation of the user stories documentation is based on whether or not it includes information that can be used to identify COSMIC concepts. They concluded that guarantying the US quality may guaranty the measurement results quality as well.

In our study, we detail a US in a way that represents all the required information to apply COSMIC FSM method. Consequently, the refinement between the existing description of user stories in section 2.1

Table 1: Summary of the proposals focused on requirements change in scrum process.

| Study | Focus | Type | Findings |
|---|---|---|---|
| (Alsalemi and Yeoh, 2015) | Product backlog change management and requirement traceability | Survey | Lack of requirement change traceability |
| (Lloyd et al., 2017) | Requirements change management in distributed agile development | Experimental | A supporting tool |
| (Commeyne et al., 2016) | Evaluation of teams' productivity using COSMIC | Experimental | COSMIC is more reliable in estimating models with much smaller variances |
| (Stålhane et al., 2014) | Impact of technical changes in safety requirements | Exploratory | A supporting tool that ensures the validity of safety |
| This Study | Evaluation of functional changes in scrum process using COSMIC | Exploratory | Quantify FC request to help the decision-makers |

**/*description*/**

**As a** <UserType> **I want to** <Action> <Object> **so that** <value or expected benefit> <NFR>

**/*Scenario description*/**

<User/System> <ActionType: Entry, Read, Write, eXit, Expletive> <DataTransferred> <Action>

Figure 2: Detailed user story description format.

and COSMIC method leads to the new description that we proposed in Figure 2, where:

- <UserType> is the user of the US referred to as the functional user in COSMIC.
- <Action> and <Object> are used to replace the concept "feature or functionality" in the US description provided in section 2.1. In fact, the "feature or functionality" is a combination of an action and an object that the action will be applied on. These two concepts are important in analyzing and finding similarities between different user stories. This distinction is required when measuring the functional size of a US. For example, considering the US "as customer I can log-in to my account". The <Action> in this US is then "log-in". And the <Object> is "customer".
- <value or expected benefit>: It is used to characterize the successful ending of the US. This information is used to distinguish between two functional processes.
- <NFR> describes the non-functional requirements.

<value or expected benefit> and <NFR> are optional.

Two different user stories could not have the same combination of <UserType>, <Action>, <Object>, and <value or expected benefit>. Although this description represents more details compared to the old one, it do not represent the functional sub-processes. Hence, moving to the scenario description is required. At this level, we will be able to identify the sub-processes. We distinguish the following concepts:

- <User/System>: three main types are identified such as external actor, system and data storage. External actor could be a human actor (*e.g.,* administrator, client, etc.) or an external system in a direct relation with the software to be measured. The system that represents the software to be measured. Finally, the data storage that allows the retrieving and saving of data (*e.g.,* databases, XML files, memory cards, etc.).
- <ActionType>: the action that will be applied on a certain data group is restricted to a number of known verbs (*e.g.,* select, ask, read, etc). These verbs could be classified into four main corpses: entry actions, read actions, write actions, and exit actions. These corpses are provided in the Appendix. These actions represent the data movements in each US.
- <DataTransfered>: represents the data that have been transferred in each functional sub-process. They can be either input or output data. This depends on the <ActionType> and the <User/System>. <DataTransfered> in COSMIC corresponds to the "data group" concept.
- <Action>: gives a summary of the user story purpose.

As an example, lets consider the following user story (US1):

**As an** <Administrator> **I want to** <add> <a Customer>

The priority of US1 is equal to P1. The importance of US1 is Essential. And the status of US1 is "done". The functional size of US1 is equal to 4 CFP. The detailed measurement results are provided in Table 2.

Table 2: Detailed measurement results of the functional size of US1.

| Functional User | US Detailed Description | Data Group | Objects of interest | Data Movement Type | CFP |
|---|---|---|---|---|---|
| | The administrator asks to add a customer | - | - | - | - |
| | The system displays the interface | - | - | - | - |
| Administrator | The administrator enters the data of the customer | customer data | customer | Entry | 1 |
| - | The system checks the existing of the client | customer data | customer | Read | 1 |
| - | The system inserts the data | customer data | customer | Write | 1 |
| Administrator | the system displays error/ confirmation message to the administrator | error message | error | eXit | 1 |
| **FS(US1) = 4 CFP** | | | | | |

## 3.2 Prioritizing User Stories

In scrum, user stories are prioritized as requested by the PO (Ambler, 2014). The development team starts the implementation with the more prior US. Whereas, PO does not have enough knowledge about the implementation details. For instance, it is not possible to buy a product without login-on as a customer. Therefore, we propose to balance the US for PO and development team perspectives according to two parameters: importance (developers perception) and priority (PO perception). The priority of user stories is defined by the PO. The US that must be implemented first is the more prior one (*i.e.,* P1 is more prior than P2, P2 is more prior than P3, etc.). The importance of a US can be Essential or Desirable. Essential user stories represent the basic user needs (Fairley, 2009). Desirable user stories add values to the software (Fairley, 2009). For two user stories with the same priority, the developers start with the Essential one. If two user stories have the same priority and the same importance, the developers will start with the US having the minimum functional size. For example, a US with importance = Desirable and priority = P1 (*i.e.,* (D, P1)) indicates that this US is Desirable for the developers but very prior to the PO. Hence, it may be selected after all the user stories with importance = Essential and priority = P1 (*i.e.,* (E, P1)).

Figure 3 is used to help prioritizing user stories in the product backlog/sprint. Hence, it can be used when selecting user stories from the product backlog and when re-organizing user stories in an on-going sprint after an approved FC. The importance/priority of the changed user stories is compared to the importance/priority of the user stories initially in the sprint. For example, suppose that a FC proposes the addition of a US with importance = Essential and priority = P1. If this FC is accepted, the added US must be implemented before the user stories initially selected in the sprint with a priority lowest than P1 (*i.e.,* P2, ...Pn).

On the other hand, developers identify the status of a US that can be used to control the development progress of a US. Thus, the status of a US can be:

- **New** is the status of a user story in the product backlog.
- **To do** is the status of a user story assigned to an on-going sprint.
- **Doing** is the status of a user story currently being implemented.
- **To test** is the status of a user story ready for testing.
- **Done** is the status of a user story tested with success.

In our FC evaluation approach, we consider that only two status are sufficient (done and undone). Thus, a US with New, To do or Doing status is not fully implemented. Whereas, a US with Done or To test status is fully implemented. Hence, we keep these two US status: undone and done as given in Figure 4. Where an undone user story is not fully implemented. And, a done user story if fully implemented.

## 3.3 Measuring the Software Functional Size from its US Descriptions

This section proposes measurement formulas that can be used to measure the software Functional Size (FS) based on the description of its user stories. Note that the FS of the product backlog can be different from the FS of the increment product. In fact, changes always happen in the scrum process. Hence, new

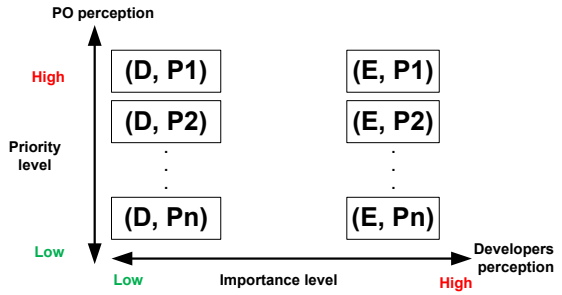


Figure 3: Prioritizing user stories in product backlog/sprint.


Figure 4: Adapted user stories status in scrum process.

functionality may appear, while others may be modified or deleted. The functional size of the product backlog is given by measuring the sizes of all sprints initially identified. While, the functional size of the increment product depends on the functional size of the implemented sprints.

The functional size of the product backlog or the increment product is equal to the sum of the functional sizes of *all* the sprints it includes (see Equation 1).

$$FS(P) = \sum_{i=1}^{n} FS(S_i) \quad (1)$$

Where:

- FS(P) is the functional size of the product backlog or the increment product.
- FS($S_i$) is the functional size of sprint $i$.
- $n$ is the number of sprints initially identified in the case of product backlog size measurement or the number of implemented sprints in the case of increment product size measurement.

The functional size of a sprint is equal to the sum of the functional sizes of all the user stories it includes (see Equation 2).

$$FS(S_i) = \sum_{j=1}^{m} FS(US_{ij}) \quad (2)$$

Where:

- FS($S_i$) is the functional size of sprint $i$ ($1 \leq i \leq n$).
- FS($US_{ij}$) is the functional size of the user story $j$ in $S_i$.
- $n$ is the number of user stories in sprint $S_i$.

The functional size of a user story is equal to the sum of the functional sizes of its actions (see Equation 3).

$$FS(US_{ij}) = \sum_{k=1}^{p} FS(Act_{ijk}) \quad (3)$$

Where:

- FS($US_{ij}$) is the functional size of the user story $j$ in $S_i$.
- FS($Act_{ijk}$) is the functional size of action $Act_{ijk}$ in $US_{ij}$ ($1 \leq i \leq n$ and $1 \leq j \leq m$).
- $p$ is the number of actions in user story $j$.

## 3.4 Research Method

Accepting or rejecting a FC request depends on two main factors: the FS of the changed US and the US status (done or undone). In fact, the FS(FC) gives a *real* evaluation of the FC (Haoues et al., 2017). A change in an ongoing sprint, in an implemented sprint or in the product backlog may be handled with or without extra cost/time. Consequently, every FC needs to be evaluated. In our study, we evaluate a FC proposed in an ongoing or an implemented sprint. This evaluation is used later to provide recommendations to the decision-makers to accept, defer or deny a FC request.

In Figure 5, we present the different phases of this study. In scrum, a FC request is proposed by the PO or the development team. First, it should be expressed in terms of US format to identify the changed US (noted by USa). This later has a current status which is either done (*i.e.,* the change is in an implemented sprint or an ongoing sprint) or an undone user story (*i.e.,* the change is in an ongoing sprint). In the case of an ongoing sprint, we identify the attributes of the sprint where the change occur (*e.g.,* size, start date, etc.) and measure respectively the FS(FC), the FS(USa) and the functional sizes of all the undone user stories in the same sprint. In the case of an implemented sprint, we measure the FS(FC) and the FS(USa). These measures are used to evaluate the FC and provides guidelines to orchestrate decision making (*i.e.,* accept, deny or defer a FC request).

### 3.4.1 FC Evaluation in an Ongoing Sprint

At the moment when a FC appears, an ongoing sprint may contain both done and undone user stories. Thus, if the status of the changed user story (USa) is undone, our method proposes to compare the FS(FC) to the functional size of all the undone user stories in the

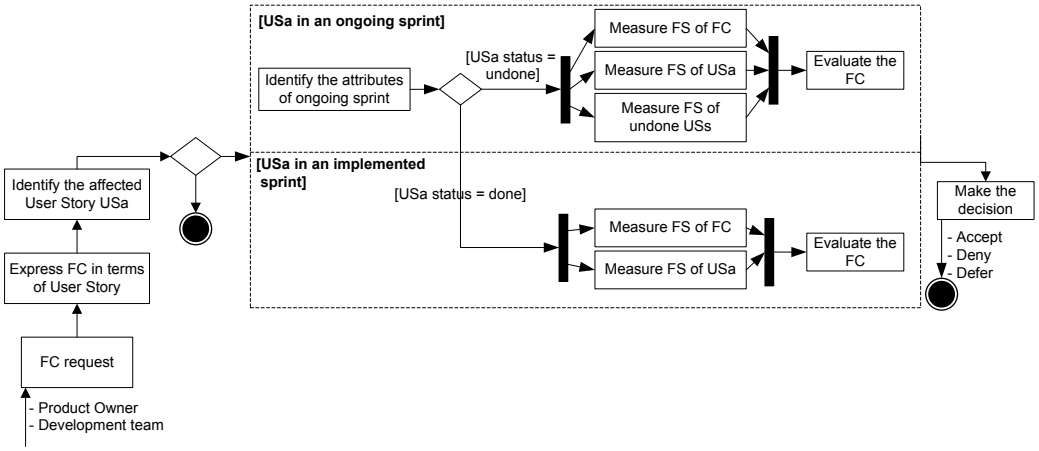


Figure 5: Phases for the FC request evaluation in the scrum process.

Table 3: Evaluating a FC request where USa status = undone.

| Low | Moderate | High |
|---|---|---|
| 1 CFP | 2 CFP $\leq$ FS(FC) $\leq$ FS(US undone) | FS(FC) $>$ FS(US undone) |

Table 4: Evaluating a FC request where USa status = done.

| Low | Moderate | High |
|---|---|---|
| 1 CFP | 2 CFP $\leq$ FS(FC) $\leq$ FS(USa) | FS(FC) $>$ FS(USa) |

same sprint. Hence, different baselines will be used to control the status of the FC (see Table 3). Whereas, if the status of the USa is done, we compare the FS(FC) to the functional size of USa.

A "High" FC is a change with a functional size bigger than the total functional sizes of undone user stories in the same sprint. It will have a potential impact on the software development progress. However, the functional size of a "Low" FC is equal to 1 CFP. This change can be handled without any impact on the software development progress. Whereas, a "Moderate" FC is a change with functional size lowest than the functional size of undone user stories in the same sprint. It will produce few changes in the software development progress.

#### 3.4.2 FC Evaluation in an Implemented Sprint

An implemented sprint in the increment product include a number of done user stories. Hence, a FC affecting a done US means that the work that has been already done must be changed. Thus, an additional time and effort may be required to handle the change.

A "High" FC is a change with a functional size bigger than the functional size of the changed user story in the implemented sprint (USa). An important effort may be required to implement this change. However, the functional size of a "Low" FC is equal to 1 CFP. This change can be handled without any required effort. However, a "Moderate" FC is a change with functional size less than the functional size of the user story affected by the change in the implemented sprint. A little effort may be required to implement a "Moderate" change.

## 4 DECIDING ON A FC REQUEST

Software functional size can be used not only for effort/cost estimations but also for making-decisions (*e.g.,* budget decision, portfolio decision, etc.) (Abran, 2010). In this section, we provide some recommendations for the decision-makers (*cf.,* product owner, development team and the scrum master) that can be used to help in making decision regarding a FC request. Recall that a FC may affect either an ongoing sprint or an implemented sprint. These decisions are as follows:

- Accept the FC request which means implement the FC in the current sprint.
- Deny the FC request which is made only if the FC proposes a new software (re-start the development from the beginning).
- Defer the FC request to the next sprint means accept the FC and implement it in the next sprint not in the current one.

### 4.1 FC in an Ongoing Sprint

Algorithm 1 provides recommendations that can be used in making decisions when the FC is a modifica-

tion and affects a US in an ongoing sprint. These recommendations are based mainly on the comparison between the FS(FC), the functional size of all undone user stories in the current sprint (noted by FS(US undone) in Algorithm 1), and the functional size of the changed user story (noted by FS(USa) in Algorithm 1). For instance, if the FS(FC) is greater than the total sizes of all the undone user stories in the current sprint, we suggest to defer the FC. Hence, the changed user story is deleted from the current sprint and added after modification with respect to the change to the next sprint. In the case where the FS(FC) is less than the total sizes of all the undone user stories in the current sprint, it is required to compare the FS(FC) to the FS(USa) before the change (noted by FS(USa)i in Algorithm 1). Thus, if the FS(FC) is greater than the FS(USa)i, we suggest to defer the FC, and the USa after the change (noted by (USa)f in Algorithm 1) is added to the next sprint. Whereas, if the FS(FC) is less than the FS(USa)i, the decision will be made based on the impact of the change on the FS(USa)i. In the case where a FC proposes the addition of a user story without changing any user story in the sprint, a comparison must be done between the FS(FC) and the functional sizes of all the undone user stories in the sprint. Hence, if the FS(FC) is less than the FS(US undone), the FC is accepted. Otherwise, the FC is deferred to the next sprint. A deletion FC request do not have any affect on the development progress.

### 4.2 FC in an Implemented Sprint

To carry out a FC in an implemented sprint, it is required to discuss the change with the PO. In fact, changing an implemented sprint means re-work (*i.e.,* redoing a work that already has been done as requested by the PO). In the case of an implemented sprint, we suggest to deny the change. However, in order to satisfy the PO needs, we provide some analysis that allow the PO to determine the importance of the change such as a comparison between the time spent in implementing the changed user story and estimate the required time to re-develop the user story after the change. These analysis are provided in Algorithm 2. Hence, this algorithm do not provide recommendations to answer a FC request but it provides some warnings to remember the PO about the importance of the FC, whether it is really needed or not.

## 5 CASE STUDY AND TOOL

In this section, we demonstrate the application of our approach on a case study “E-commerce” web site.

Algorithm 1: Deciding on a FC in an ongoing sprint.

**Aim** : Deciding on a FC in an ongoing sprint
**Require:** FS(FC), FS(US undone), and FS(USa).
**Ensure** : Recommendations

```
1  begin
2    if FS(FC) > FS (US undone) then
3      defer the FC to the next sprint;
4      delete (USa)i from ongoing sprint /*
         (USa)i is US before the FC */
5      add (USa)f to next sprint /* (USa)f is
         US after the FC */
6    else if FS(FC) ≤ FS(US undone) then
7      if FS(FC) > FS(USa)i then
8        defer the FC to the next sprint;
9        delete (USa)i from current sprint;
10       add (USa)f to next sprint;
11     else if FS(FC) ≤ FS(USa) then
12       if FS(USa)f > FS(USa)i then
13         /* the FS(USa) after the
              change is greater than the
              FS(USa) before the change
              */ if remainingtime (USa)f ≤
              requiredtime & teamprogress
              = early then
14           accept the FC;
15           delete (USa)i from the
               current sprint;
16           add (USa)f to the current
               sprint;
17         else
18           defer the FC;
19           delete (USa)i from the
               current sprint;
20           add (USa)f to the next
               sprint;
21       else if FS(USa)f < FS(USa)i then
22         /* the FS(USa) after the
              change is lower than the
              FS(USa) before the change
              */
23         accept the FC;
24         delete (USa)i from the current
             sprint;
25         add (USa)f to the current
             sprint;
26   else if FS(FC) = 1 then
27     accept the FC;
28     delete (USa)i from the current sprint;
29     add (USa)f to the current sprint;
```

Algorithm 2: Deciding on a FC in an implemented sprint.

**Aim** : Deciding on a FC in an implemented sprint
**Require:** FS(FC), FS(USa), USa priority, priorities P, FS(USs), devtime, USa real DevTime
**Ensure** : Warnings

```
1  begin
2      FC percentage = FS(FC) * 100 div FS(USa)
3      Avr DevTime = ∑ devtime div ∑ FS(USs)
4      if USa importance = Essential then
5          alert (you are going to change an Essential User Story with + FC percentage) ;
6      else if USa priority < P(n div 2) then
7          alert (This FC could highly impact other user stories as it was implemented in an early phase) ;
8      else if Avr DevTime < USa real DevTime then
9          alert (this US took more time in development then the average time needed to accomplish a US with the same functional size. It may contain extra data manipulation) ;
```

## 5.1 Case Study

The case study "E-commerce" is developed by a team of engineers. The web site allows the customer to buy computer equipments on-line. This product has been delivered after six sprints each one lasts for two weeks. It includes initially ten user stories. The initial detailed measurement results are given in Table 5.

The development team defined the first sprint backlog by choosing the user stories organized according to their importance/priority. US5, US6, and US7 have been implemented during the sprint S1. All the user stories have the importance = Essential and the priority = P1. In sprint S2, and based on the importance/priority of user stories in the product backlog, US4, US8, US9 and US10 have been chosen to be implemented in S2. As mentioned in Table 5, US4 is with importance = Essential and priority = P2. Whereas, US8, US9 and US10 are with importance = Essential and priority = P3. Hence, the development team starts by US4.

By the end of the implementation of US4, the PO proposes to add (US 11, US 12, and US 13). Where: the FS(US 11) = 4 CFP, FS(US 12) = 3 CFP and FS(US 13) = 3 CFP. The importance/priority for all the added user stories is (Essential, P2). Note that the status of the user stories initially in the sprint are: US4 status = done and US8 status = US9 status = US 10 status = undone. The question here is whether to accept the FC and implement it in the current sprint (S2) or defer the FC to the next sprint (S3). In this case, the FC affects an ongoing sprint and proposes the addition of three user stories without changing any user story in the sprint. The total functional size of undone user stories is equal to 12 CFP. While, the FS(FC) is equal to 10 CFP. We need to classify the user stories (initially in S2 and new ones) according to their importance/priority (see Figure 3). For this purpose, we compare the importance/priority of the new user stories (US11, US12 and US13) with the user stories already in S2 (US4, US8, US9, and US10). Hence, the user stories will be organized as follow: US4, US11, US12, US13, US8, US9 and US10. The user stories coming from the FC appear in the beginning of the list. Thus, by applying Algorithm 1, we make the following recommendations.

Table 5: Detailed Product Backlog.

| User Stories | FS(US) | Status | Importance | Priority |
|---|---|---|---|---|
| US 1 | 3 CFP | New | Desirable | P3 |
| US 2 | 3 CFP | New | Desirable | P3 |
| US 3 | 3 CFP | New | Desirable | P3 |
| US 4 | 3 CFP | New | Essential | P2 |
| US 5 | 4 CFP | New | Essential | P1 |
| US 6 | 6 CFP | New | Essential | P1 |
| US 7 | 6 CFP | New | Essential | P1 |
| US 8 | 3 CFP | New | Essential | P3 |
| US 9 | 3 CFP | New | Essential | P3 |
| US 10 | 6 CFP | New | Essential | P2 |

- Accept US 11, US 12, and US 13.
- Defer US 8, US 9, and US 10 to the next sprint S3.

## 5.2 Tool

Algorithm 3 is used to measure the FS of a given US. An action is characterized as a data movement if it includes a verb presented in one of the corpses in the Appendix.

Figure 6 gives some snapshots of the proposed tool in this paper. The productivity of the development team during the sprints is given in Interface 1. As it is displayed in this interface, the productivity of the team decreases to 6 CFP/hour in sprint S4. On the other hand, the number of done user stories during the past sprints is equal to 80% of the initially defined user stories. It remains only 20% of user stories to be implemented during the next sprints.

Interface 2 is used for making recommendations after a FC request. For instance, after finishing the implementation of all user stories (from US1 to US16)[1], the PO suggests a new FC that proposes to delete US10. US10 has been already implemented in sprint S3. The functional size of US10 is equal to 6 CFP and the implementation of US10 has been done in seven days. Moreover, this user story presents a fundamental requirements. According to Table 4, this FC is a "Medium" change. The FS(FC) is exactly equal to the FS(US10) which is 6 CFP. Thus, by applying Algorithm 1 we suggest to accept this FC.

Algorithm 3: Measuring the FS(US).

**Aim** : Measure the functional size of a user story
**Require:** US is the user story to be measured.
**Ensure** : SizeUS is the functional size of US.

```
1  begin
2      Size US = 0;
3      if User/System = External User & action
         = corpusEntry & datagroup != Empty
         then
4          data movement type := entry;
5          SizeUS := SizeUS + 1;
6      else if User/System = System & action =
         corpuseXit & datagroup != Empty then
7          data movement type := exit;
8          SizeUS := SizeUS + 1;
9      else if User/System = System & action =
         corpusWrite & datagroup != Empty
         then
10         data movement type := write;
11         SizeUS := SizeUS + 1;
12     else if User/System = System & action =
         corpusRead & datagroup != Empty then
13         data movement type := read;
14         SizeUS := SizeUS + 1;
```

# 6 CONCLUSION AND THREATS TO VALIDITY

This work proposed a decision support tool for measuring the Functional Size of User Stories in terms of CFP units using COSMIC FSM method. In addition, functional changes are measured and evaluated so that the decision-makers will be guided to decide which FC request should be accepted, deferred or denied. The proposed approach in this paper has been illustrated through the case study "E-commerce". However, the validity of the above results are subject to two types of threats (internal and external) (Wohlin et al., 2000):

- The internal validity threats are related to four issues. The first issue affecting the internal validity of our approach is its dependence on a detailed description of the user story; such details may not always be available. Thus, for further work, we consider that approximate/rapid FC evaluation is required especially for an urgent FC request. The second issue is related to the productivity of the development team. In fact, two functional processes with exactly the same functional size do not require always the same development time. Moreover, the rapidity of the development team at the beginning of the sprint and the end of the sprint are not the same (this depends on the development team skills). Thus, for further work, we will use the structural size measurement proposed by (Sellami et al., 2015) for more precise evaluation (*i.e.,* to take into account the data manipulations). The third issue is related to the evaluation of the FC which is based only on the FS(FC) without taking into account the FC type (delete, addition or modification). However, we consider that this factor is important in the evaluation of a FC request. Finally, in this study we did not take into account the relationship between the user stories. In fact, a FC affecting a use story may lead to an impact on the functional size of other use stories. For further work, we will focus on the relationships between user stories and change propagation.
- The external validity threats deal with the possibility to generalize the results of this study to other case studies including the usability of the proposed tool and the decision algorithms. The first issue is the limited number of case studies used to test the proposed tool. In fact, only one case study has been used: the 'E-commerce". Thus, testing the proposed tool and algorithms in an industrial environment is required in order to get the feedback from the practitioners.

[1]US15, US6, and US 6 have been implemented in S1. US4, UC11, UC12 and UC13 have been implemented in S2. US8, US9 and US10 have been implemented in S3. US14 and US15 have been implemented in S4. US1, US2, and US3 have been implemented in S5. Finally US16 has been implemented in S6.

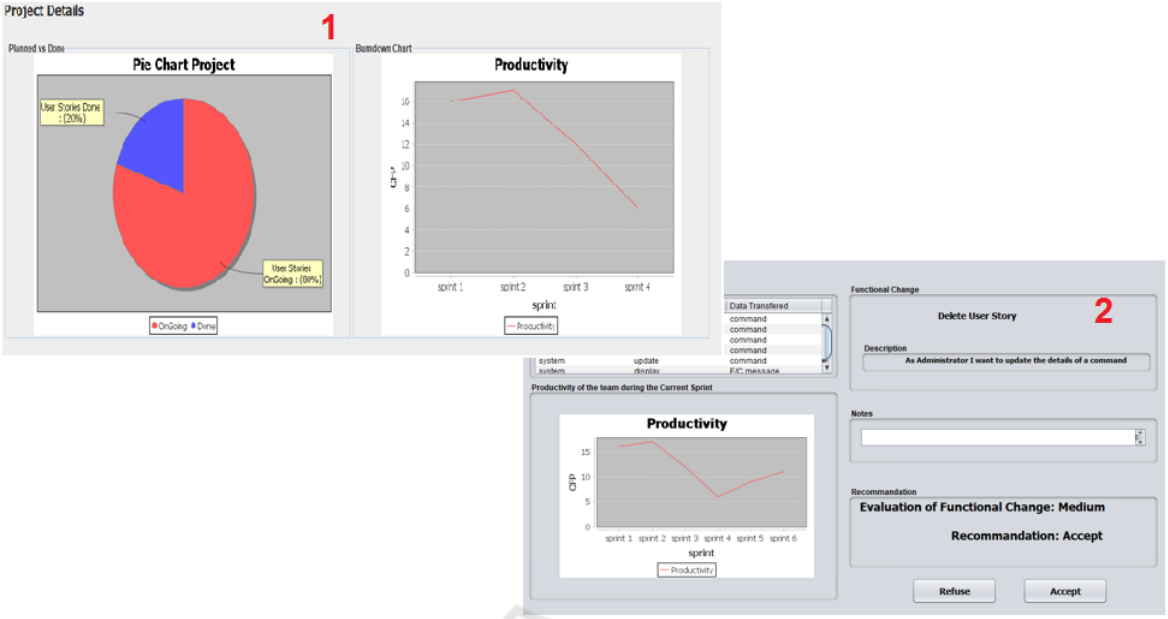


Figure 6: Snapshots of tool with the case study "E-commerce".

## REFERENCES


Abran, A. (2010). *Software Metrics and Software Metrology*. IEEE Computer Society.

Abran, A. (2015). *Software Project Estimation: The Fundamentals for Providing High Quality Information to Decision Makers*. Wiley-IEEE Computer Society Pr, 1st edition.

Alsalemi, A. M. and Yeoh, E. T. (2015). A survey on product backlog change management and requirement traceability in agile (scrum). In *the 9th Malaysian Software Engineering Conference (MySEC)*, pages 189–194.

Ambler, S. W. (2014). *User Stories: An Agile Introduction*.

Bano, M., Imtiaz, S., Ikram, N., Niazi, M., and Usman, M. (2012). Causes of requirement change - a systematic literature review. In *EASE 2012*.

Berardi E., Buglione L., S. L. S. C. T. S. (2011). Guideline for the use of cosmic fsm to manage agile projects, v1.0.

Cohn, M. (2004). *User Stories Applied: For Agile Software Development*. Addison-Wesley Professional.

Commeyne, C., Abran, A., and Djouab, R. (2016). *Effort Estimation with Story Points and COSMIC Function Points: An Industry Case Study*.

COSMIC (2017). *The COSMIC Functional Size Measurement Method, Version 4.0.2, Measurement Manual*.

Desharnais, J. M., Kocaturk, B., and Abran, A. (2011). Using the cosmic method to evaluate the quality of the documentation of agile user stories. In *2011 Joint Conference of the 21st International Workshop on Software Measurement and the 6th International Conference on Software Process and Product Measurement*, pages 269–272.

Dikert, K., Paasivaara, M., and Lassenius, C. (2016). Challenges and success factors for large-scale agile transformations. *Journal of Systems and Software*, 119(C):87–108.

Fairley, R. E. (2009). *Managing and Leading Software Projects*. Wiley-IEEE Computer Society Pr.

Gilb, T. (2018). Why agile product development systematically fails, and what to do about it!

Haoues, M., Sellami, A., and Ben-Abdallah, H. (2017). Functional change impact analysis in use cases: An approach based on COSMIC functional size measurement. *Science of Computer Programming, Special Issue on Advances in Software Measurement*, 135:88–104.

Lloyd, D., Moawad, R., and Kadry, M. (2017). A supporting tool for requirements change management in distributed agile development. *Future Computing and Informatics Journal*, 2(1):1–9.

Schwaber, K. (2004). *Agile Project Management with Scrum (Developer Best Practices)*. Microsoft Press; 1 edition.

Sellami, A., Hakim, H., Abran, A., and Ben-Abdallah, H. (2015). A measurement method for sizing the structure of uml sequence diagrams. *Information & Software Technology*, 59:222–232.

Stålhane, T., Hanssen, G. K., Myklebust, T., and Haugset, B. (2014). Agile change impact analysis of safety critical software. In Bondavalli, A., Ceccarelli, A., and Ortmeier, F., editors, *Computer Safety, Reliability, and Security*, pages 444–454.

Taibi, D., Lenarduzzi, V., Ahmad, M. O., and Liukkunen, K. (2017). Comparing communication effort within

the scrum, scrum with kanban, xp, and banana development processes. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, EASE'17.

Verwijs, C. (2016). 10 useful strategies for breaking down large user stories (and a cheatsheet).

Wohlin, C., Runeson, P., Hst, M., Ohlsson, M. C., Regnell, B., and Wessln, A. (2000). *Experimentation in Software Engineering: An Introduction*. Kluwer Academic Publishers.

# APPENDIX

**Entry Corpus:** These verbs express Entry data-movements: Assign, change, choose, click, create, edit, give, input, modify, provide, re-enter, select, submit, type, update.

**Exit Corpus:** These verbs express eXit data-movements: display, edit, list, output, post, present, print, return, send, Show update, view.

**Read Corpus:** These verbs express read data-movements: find, get, obtain, post, read, recognize retrieve, Validate, Verify.

**Write Corpus:** These verbs express write data-movements: add, archive, change, create, define, delete, edit, insert, record, register, remove, save, store, Update.