An in-Depth Requirements Change Evaluation Process using Functional and Structural Size Measures in the Context of Agile Software Development

Hela Hakim[1], Asma Sellami[2] and Hanêne Ben Abdallah[3]
[1]*Mir@cl Laboratory, University of Sfax, FSEGS, BP 1088. 3018, Sfax, Tunisia*
[2]*Mir@cl Laboratory, University of Sfax, ISIMS, BP 242. 3021, Sfax, Tunisia*
[3]*Higher Colleges of Technology, Dubai, U.A.E.*

Keywords: Requirement Change RC, Functional Change FC, Structural Change SC, Functional Size Measurement FSM, Structural Size Measurement SSM, User Story Description, COSMIC-ISO 19761, Structural Size Measurement Method, Scrum, User Stories US, Agile.

Abstract: The Agile methodology known as Scrum is increasingly used in software development as a response to the challenges of managing frequent requirements changes. However, a number of agile-based projects yield unsatisfactory results mainly because of a lack of a well-defined change evaluation process. In fact, such a process should be set-up early in the Software Life-Cycle (SLC). This paper proposes an in-depth evaluation process for requirements changes affecting either an ongoing sprint or an implemented sprint. This evaluation process involves two levels of details: a functional change level and a structural change level based, respectively, on the COSMIC functional size measurement method –ISO 19761 and the Structural Size Measurement Method. We investigate the use of both COSMIC FSM and SSM methods for rapid and detailed evaluation-based measures of a requirement change request.

1 INTRODUCTION

Compared to other projects, software projects are more difficult to manage due mainly to invisibility, complexity, conformity and changeability of the software products (Fairley, 2009). In particular, at the beginning of the Software Life-Cycle (SLC), requirements are often unclear, ambiguous, and incomplete. Hence, they may change frequently throughout the Software Life-Cycle (SLC). In addition, other reasons may incur requirement changes (*e.g.,* missing functionality, defects corrections, etc.) (Bano, 2012) and (Shalinka et al., 2018). These SLC early changes have lower cost than requirements changes that occur at the later SLC phases (Fairley, 2009).

Overall, researchers have adopted two strategies to reduce the cost of software requirements changes: anticipate changes, and use flexible models more adapted to embrace changes, such as Agile methods (Sellami et al., 2018) (Abdalhamid et al., 2017).

Currently, agile methods (*e.g.*, extreme programming, Scrum, crystal, etc.) are increasingly being adopted in software organizations. Two of the most popular agile methods are eXtreme Programming and Scrum (Hamed et al., 2013) (Dikert, 2016). These are more adapted to the software evolution and encourage an active collaboration between development teams and the product owner. Although Scrum is gaining popularity in comparison with other agile methods, 61% of agile projects end in failure (Gilb, 2018).

This is due mainly to the lack of comprehensive documentations in Scrum (Furtado, 2016), the inappropriate application of Scrum concepts, the limited use of standardized measures, and the poorly change evaluation. For a successful project development, Gilb reported that it is important to use standardized measurement on reviewing development progress, re-evaluating user stories priorities, etc. (Gilb, 2018).

More specifically, requirements are seen at a high level of description in the functional level (black box), while these same requirements are described in detail in the structural level (white box). Consequently, a well-defined change evaluation process based on both functional and structural size of a change is required at any step of the Scrum process, even within an ongoing sprint. In this paper, we will focus on both ongoing and implemented sprints.

Hakim, H., Sellami, A. and Ben Abdallah, H.
An in-Depth Requirements Change Evaluation Process using Functional and Structural Size Measures in the Context of Agile Software Development.
DOI: 10.5220/0009876003610375
In *Proceedings of the 15th International Conference on Software Technologies (ICSOFT 2020)*, pages 361-375
ISBN: 978-989-758-443-5

In practice, changes requested throughout the Scrum process are often evaluated by "expert judgment". The development teams may have the required knowledge about the changed product and the required modifications that must be done with respect to the change request. However, a change evaluation that is based only on the expert judgment is hardly criticized because there is no guarantee of their effectiveness (Abran, 2015).

Thus, we propose an in-depth requirements changes evaluation process that is based on the standardized COSMIC functional size measure (ISO 19761) as well as the structural size measure (Sellami et al., 2015) to achieve more accurate results. An appropriate evaluation of a change request will help the decision makers responding to the change request.

The remainder of this paper is organized as follows: Section 2 first overviews the Scrum process, the COSMIC FSM method (ISO/IEC 19761), and the structural size measurement method. Secondly, it discusses some related works. Section 3 presents our proposed evaluation process that development teams and other stakeholders can use to make appropriate decisions about a change request. Section 4 evaluates our approach. Section 5 discusses several threats to validity. Finally, Section 6 summarizes the presented work and outlines some of its possible extensions.

2 BACKGROUND

2.1 Overview of the Scrum Process

Scrum process is a framework for a complete project management method developed and sustained by Scrum creators: Ken Schwaber and Jeff Sutherland (Ken Schwaber, et al., 2017). It allows the management of the development of complex applications. It involves Scrum Teams and their associated roles, events, artifacts, and rules. Each component within this framework serves a specific purpose and is essential to Scrum's success and usage. It allows a better communication across the development team and the product owner. For a successful Scrum project, the development team must learn how to manage themselves efficiently. In addition, the product owner must be actively involved in every single phase of the software development. Scrum appears to work better with teams of 5 to 9 people, with large projects being typically handled by several scrum teams.

Nevertheless, some companies adapt Scrum for large-scale projects (Dikert et al., 2016). The Scrum process, illustrated in Figure 1, starts with a high-level definition of the project scope (requirements). Scrum uses the product backlog as a list of stories created by the product owners based on their initial requirements that clients/stakeholders/customers describe. Indeed, these stories may increase or decrease in size based on decisions made throughout the software development process. The list of stories is prioritized by the product owner to be used as an iterative input for different sprints (Iterations) (Schwaber et al., 2004). Thus, the active involvement of the product owner is mandatory to explain, elucidate the next iteration that should be implemented and evaluate/test the work done.

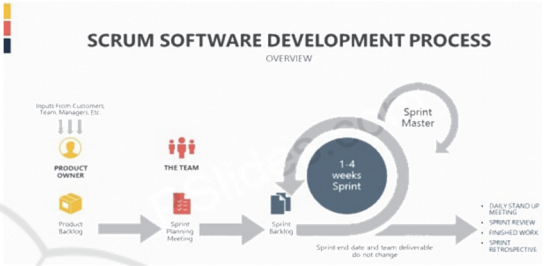

Figure 1: Scrum software development process.

For a single sprint, four types of meetings should be held: 1-sprint planning meeting, daily Scrum, sprint retrospective meeting, and sprint review meeting. The stories to be implemented in a sprint are captured during the planning meeting. They are selected from the product backlog according to their priorities and placed in a sprint backlog. In practice, usually, only the first two or three sprints are identified and planned. Daily stand up meetings are held during the sprint to discuss: what has been done, what to do, and what are the issues (Cohn, 2004) (Ken Schwaber et al., 2017). Each sprint is ended by a sprint retrospective meeting, during which the team reviews the sprint and decides which change will be made, and how they can improve their work in the next sprint.

As we mentioned previously, Scrum uses the user stories to represent the user requirements at different levels of details. A user story is a requirement written in a specific way illustrating the type of user, feature or functionality that the user want to do in order to realize some benefit (Cohn, 2004). The user story description adapted in practice is presented below. This description identifies who will do the user story or find it valuable, What it can be used for, and Why it is valuable or important. *"As a user I want to ... So that ..."*.Typically, development team members use the user story point to determine the effort required for the

accomplishment of a user story compared to other user stories in the same product backlog. Although its popularity, user story point is not a good estimation technique. It has been widely criticized (*cf.,* (Berardi et al., 2011), (Commeyne et al.,2016), (Desharnais etal.,2012), etc.) In fact, user story point is only meaningful for a specific development team and project. Thus, it is crucial to use a standardized measurement method (such as the COSMIC FSM method) as a means for sizing the work product (*e.g.,* the user story). It is also required to use a combination of COSMIC method with structural size measurement method to exhibit a fine-grain measure of requirements. Many situations require a detailed description of requirements to derive estimations that are more accurate. In addition, the study in (Commeyne et al., 2016) reports on the performance of estimation models built using Story Points and COSMIC Function Points. The study showed that the use of COSMIC leads to estimation models with much smaller variances and demonstrates that the use of COSMIC allows objective comparison of productivity across tasks within a Scrum environment. For these reasons, we selected to use not only COSMIC FSM method but also structural size method for sizing respectively functional changes and structural changes, and therefore, provide a real and precise evaluation of the change request.

2.2 COSMIC FSM Method–ISO19761

COSMIC –ISO 19761 considers that a FUR involves a number of functional processes each of which is detailed by a set of sub-processes of two types: data manipulation and data movement. A data manipulation sub-process processes data internally/locally. In contrast, a data movement sub-process moves a data group from/to a functional user (respectively Entry and eXit data movement) or from/to a persistent storage (respectively Read and Write data movement). The software size is measured by counting one CFP (COSMIC Function Point) for each data movement. The size of each functional process is measured separately, and the sizes of all functional processes are added to provide the whole software size.

Compared to other Functional Size Measurement methods, COSMIC is the most straightforward method that measures the size of a change to software (COSMIC, 2020). It defines a functional change 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 Functional Change, referred to as FS(FC), COSMIC attributes one CFP for each changed data movement, irrespectively of the change type (addition, deletion, or modification). FS(FC) is given by the aggregation of the sizes of all the added, deleted and modified data movements. The functional size of the software after a functional change is given as the sum of all added data movements minus the functional size of all removed data movements (COSMIC, 2017).

2.3 Structural Size Measurement Method

The Structural Size Measurement (SSM) is a measurement method proposed by (Sellami et al., 2015) for UML sequence diagrams. It was designed by following the measurement process recommended in (Abran, 2010). The main reason for creating such method is the need of detailed measures to quantify data manipulation.

The proposed Structural Size Measurement is applied to the combined fragments of a sequence diagram to measure its Structural Size (SS). The SS also called control structural size to refer to the structural size of both Conditional Control Structures (CCS) and Iterative Control Structures (ICS), described respectively through the *alt*, *opt* and *loop* constructs. The SS of a sequence diagram is defined at a fine level of granularity (*i.e.,* the size of the flow graph of its control structures).

The use of SS requires the identification of two types of data manipulation depending on the structure type CCS (alt and opt combined fragments in the flow graph) and/or ICS (loop combined fragment in the flow graph).

Each data manipulation is equivalent to 1 CSM (Control Structure Manipulation) unit. The sequence structural size is computed by adding all data manipulations identified for every flow graph.

2.4 Related Work

Researchers and practitioners agree that agile development provides a rapid response methodology to handle requirements change. Thus, many research studies have addressed the issues of managing requirement changes in Scrum process

In Scrum, a change priority in the backlog is the role of the Product Owner. In fact, the product backlog is the Product Owner’s tool. This implies that the elements of the backlog are ordered by priority.

A Product Owner may need to change the order of priority for various reasons among which we cite

the following:

- each time a story is added to the backlog, it must be given higher priority over other elements. It is not necessarily placed last;
- the better knowledge of a story can lead to change its priority;
- the discovery of a bug or a problem can lead to a review of the priorities;
- estimating the cost of developing a story can have an impact on its priority; and
- planning a sprint can change priorities to adjust the scope to team engagement

Change evaluation process followed by a Decision-making process in agile project has received an increased interest in recent years. In fact, the output of one process (change evaluation process) is the input of the second process (decisions-making process). For instance, Drury-Grogan and O'Dwyer explored the decision-making in Scrum process and identified the factors that may influence the decisions made during the sprint planning and daily Scrum meetings (Drury-Grogan et al., 2013).In practice, Scrum teams follow sometimes a three-step process for making-decisions during the sprint planning and daily Scrum meetings: problem identification, solution development, and selection of best alternative. They make decisions in a collaborative manner and they may be influenced by three main factors, according to (Drury-Grogan et al., 2013): Sprint duration, experience and resource availability. However, the final decisions are usually made based on the Scrum team members' experiences. Although expert judgment is much closer to reality, it is often considered as subjective (Abran, 2015).

It is less transparent compared to any other techniques and depends mainly on the experts' skills. Consequently, it is important to provide quantitative values (output of change evaluation process) as measurement results (derived from the use of both COSMIC functional and structural size measurement methods). These measurement results should both be accurate, and cover the functional and structural levels.

In addition, there are many studies that addressed the issues of managing requirements changes not only in Scrum process but also in other areas of development (such as distributed agile) and its exploitation beyond its traditional use.

For instance, in agility many types of problems have been identified. In (Lloyd et al., 2017), the authors addressed the problem of requirements changes during the software development in distributed agile development. They proposed 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 change requests. They focused on the safety requirements. Regarding the use of functionality measures in agile project, (Commeyne et al, 2016) proved that the use of ISO standards to measure the size of agile projects is mandatory. This study demonstrated the reliability of COSMIC in estimating the size and therefore, the effort required to accomplish the defined requirements. (Sellami et al., 2018) proposed a COSMIC-based tool for evaluating functional changes within the Scrum process. This tool assists the decision-makers to decide whether to accept, deny or defer a given functional change request.

These findings are summarized in Table 1.

Table 1: Summary of the proposals, focusing on changes in Scrum.

Study	Focus	Type	Findings
(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
(Alsalemi and Yeoh, 2015)	Product backlog change management and requirement traceability	Survey	Lack of requirement change traceability
(Stålhane et al., 2014)	Impact of technical changes in safety requirements	Exploratory	A supporting tool that ensures the validity of safety
(Sellami et al., 2018)	Evaluation of functional changes in Scrum process using COSMIC	Exploratory	Quantify FC request to make appropriate decisions
Our study	In-Depth Requirements Changes Evaluation Process based on functional and structural size methods	Exploratory	Quantify RC at functional and structural level to help in making accurate decisions

From Table 1, we noticed that some studies focused on functional changes (*cf.,* (Lloyd, 2017)), while other studies focused on technical changes (*cf.,* (Stålhane et al., 2014). However, changes in these works have been always considered as new requirements. In addition, none of the previous

studies used in-depth requirements changes evaluation process. This is due to the lack of detailed measurements and poorly defined scope (or requirements change requests). However, it is important to evaluate requirements' changes at two levels of details and provide useful and accurate information for the right audience (*e.g.,* the product owner or the development team).This will certainly help during the software maintenance as well as for new software development.

In practice, usually, Scrum teams do not allow changes in the middle of iteration (sprint), since developers may already have preceded the implementation. In fact, practitioners consider that changes during an ongoing sprint may introduce defects. However, other authors (Sellami et al., 2018) believe that some changes must be authorized during an ongoing sprint. For example, a change request that proposes the deletion of a user story selected in the current sprint must be authorized. Since it is useless to implement a user story that will be deleted in the next sprint. Nevertheless, changes introduced during an ongoing sprint need prioritization. In this paper, we believe that an in-depth change evaluation within the Scrum process will increase its flexibility.

3 AN IN-DEPTH EVALUATION PROCESS OF REQUIREMENTS CHANGES

This section describes an in-depth requirements changes evaluation process. We will focus on how to evaluate a requirement change request that occurs within the Scrum process and that it may affect functional requirements and-or structural requirements. In Scrum, requirements are represented in a user story format and epics. This format does not represent all the COSMIC FSM concepts (*e.g.*, data movement's identification) nor the SSM concepts (*e.g.*, data manipulations). Thus, we propose a detailed description of a user story that provides the information required to apply both COSMIC (COSMIC, 2020) and structural size method (Sellami et al., 2015). Thereafter, we propose a number of algorithms to provide size-driven prioritizations of user stories (when requirements changes occur).We also propose a set of measurement formulas to be used for sizing requirements as described in the user stories formats at different levels of details (*i.e.,* functional and structural levels).

The six generic steps described herein provide a structured in-depth evaluation process that development teams use to evaluate the status of requirements changes. These steps are presented below.

1. Refining and grooming the User Story description
2. Mapping of the US concepts with measurement concepts
3. Sizing requirements from the US description
4. Prioritizing US in the product/sprint backlog
5. Evaluating the status of requirements changes requests
6. Facilitating 'good' decision-making

Each of the change evaluation steps is described in the following sections.

3.1 Step 1: Refining and Grooming the User Story Description

Refining the user story description sets the stage for applying software size measurement methods. Clearly, well-defined requirements changes avoid ambiguity and misinterpretations. It can be evaluated with a high level of accuracy. In contrast, unclear requirements changes are prone to different interpretations and judgments. They can be evaluated approximately. In (Ken Schwaber et al., 2017), a set of rules are established to provide the meaning of a "good" User Story. It should be: i) Independent: there is no independency from other User Stories in the backlog (it is sufficient on its own to avoid dependencies with other User Stories. Because any dependency generates planning and testing issues.) ii) Negotiable: it is a support for discussion with a view to improving the initial need (It can be modified and refined until it is integrated into an iteration or Sprint). iii) Valuable: the creation of one User Story must perform service to the user (It only makes sense if it brings business value).iv) Estimable: it must be well-defined to be easily estimable. v) Small: it must be achievable on a sprint, either sufficiently small or cut so that it can be deployed on a single sprint and minimize the tunnel effects over several sprints. And, vi) testable: it must tell a story from which the acceptability criteria and the tests must flow clearly to facilitate its validation.

Different templates are proposed to determine what the users will need the software for (Sellami et al., 2018). However, there is no standardized representation of a user story. Some rules are proposed to define what Is a 'good' user story, but it is defined at a high level of details (Desharnais et al.,

2011). This is not sufficient for an in-depth change evaluation.

Thus, we propose a detailed description of a user story that expands the previous template (Sellami et al., 2018), and represents all the information required to apply not only the COSMIC FSM method but also the structural size measurement method (see Table 3). Some of the detailed US statements that need to be described includethe following:

- **<If(condition)><Then>**

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

It describes the first type of alternative scenario in a detailed description of a user story. This typically means that there is an option and the corresponding concept in the SSM method is the 'opt' combined fragment.

- **<If(condition)>**

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

<else>

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

It describes the second type of alternative scenario in the detailed description of a user story. That means that there are two alternatives (*i.e.,* two scenarios), and the corresponding concept in the SSM method is the 'alt' combined fragment.

- **<If(condition)>**

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

<elseif>

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

<**else**>

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

It describes the third type of alternative scenario in the detailed description of a user story. This means that there are more than two alternatives (*i.e.,* more than two scenarios), and the corresponding concept in the SSM method is also the 'alt' combined fragment.

- **<Loop(condition)[n]>**

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

<**end loop**>

It describes the iterative scenario in the detailed description of a user story. This means that there are *n* iterations to be executed (*i.e.,* iterative scenarios), and the corresponding concept in the SSM method is the 'loop' combined fragment.

3.2 Step 2: Mapping of the User Story Concepts with the Measurement Concepts

Step 1 produces a detailed description of a user story (*i.e.,* a new refined format of a US). This description serves as the basis for sizing requirements/ requirements changes at different levels of details. For this reason, it is important to map the US concepts with those of COSMIC and structural size methods.

Table 2 presents a mapping of US concepts (as described in step 1) with those of COSMIC functional and Structural size methods.

Table 2: Mapping of US concepts with measurements methods' concepts.

User Story	**COSMIC FSM Concepts**	**SSM Concepts**
<UserType>	Functional User or System or Persistent storage	Not Applied
<Object>	Object of interest	Not Applied
<Action>	Data movement	Not Applied
<value or expected benefit>:	Optional	Not Applied
<NFR>	Optional	Not Applied
<User/System>	Functional user, System, and Persistent storage.	Not Applied
<DataTransfered>	Data Group	Not Applied
<ActionType>	Data movements (**E**ntry,**R**ead, **W**rite,e**X**it)	Not Applied
<Action>	Data movement	Not Applied
<If (condition)> <Then >	Not Applied	***Opt***combined fragment(one alternative in the flow graph)
<If (condition)> <else >	Not Applied	***Alt***combined fragment (two alternatives in the flow graph)
<If (condition)> <else if> <else >	Not Applied	***Alt***combined fragment (more than two alternatives in the flow graph)
<Loop (condition)[*n*]> <end loop>	Not Applied	***Loop***Combined fragment(***n*** iterations in the flow graph)

3.3 Step 3: Sizing Requirements from the User Story Description

Once the requirements/ requirements changes are clearly identified and described in the US format, software size measurements can be applied. Thus, sizing requirements or sizing requirements changes can be determined according to a set of measurement formulas that are based on the refined US format.

Using measurement formulas will facilitate the measurement process. Based on the established formulas, requirements functional size with their corresponding structural size is easily generated. Note that the requirements size (either requirements functional size or requirements structural size) derived from the product backlog are different from the requirements size derived from the increment product. Because, most often changes occur during the Scrum process (Schwaber et al, 2017). Hence new functionality may appear with or without a structural requirement, while others may be modified or deleted. The product/sprint backlog/USfunctional size and respectively the product/sprint backlog/USstructural size are determined using COSMIC functional size measurement method and respectively the structural size method. Of course, the Product backlog size depends on the size of all sprints initially identified. The functional size, respectively, the structural size of the product backlog or the increment product is the sum of all the functional sizes, respectively; the structural sizes of all the sprints it includes (see Equation 1 and Equation 1').

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

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

Where:

- FS(P)is the functional size of the product backlog or the increment product.
- SS(P)is the structural size of the product backlog or the increment product.
- FS(S_i) is the functional size of sprint*i*.
- SS(S_i) is the structural size of sprint*i*.
- *n* is the number of sprints initially identified in the case when sizing the product backlog or the number of implemented sprints in the case when sizing the increment product.

The functional size, respectively, the structural size of a sprint is the sum of all the functional sizes, respectively, the structural sizes of all the user stories (US) it includes (see Equation2 and Equation2').

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

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

Where:

- FS(S_i) is the functional size of sprint*i*($1 \le i \le n$).
- SS(S_i)is the structural size of sprint*i*($1 \le i \le n$).
- FS(US_{ij})is the functional size of the US_j in S_i.
- SS(US_{ij})is the structural size of the US_j in S_i.
- *m* is the number of user stories in sprint S_i.

Note that $FS(US_{ij})$is the sum of all the functional sizes of its actions(see Equation 3).The$SS(US_{ij})$ is the sum of all the Structural sizes of its alternatives (conditional and iterative) (see Equation 3').

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

$$SS(US_{ij}) = \sum_{k=1}^{r} SS(Alt_{ijk}) \quad (3')$$

where

- FS(US_{ij}) is the functional size of the US_jin S_i.
- SS(US_{ij}) is the structural size of the US_jin S_i.
- FS(Act_{ijk})is the functional size of action Act_{ijk} in US_{ij}($1 \le i \le n$and$1 \le j \le m$).
- SS(Alt_{ijk}) is the structural size of alternative Alt_{ijk}in US_{ij}($1 \le i \le n$and$1 \le j \le m$).
- *p* is the number of actions in user story*j*.
- r is the number of Alternatives in user story *j*.

3.4 Step 4: Prioritizing User Stories in the Product/Sprint Backlog

Prioritizing US that clearly states the product owners' expectations is essential to software project success.

In Scrum, user stories are prioritized as requested by the product owner. However, the product owner perspective may not have enough knowledge about the implementation details. Hence, ordering user stories based only on priority is not sufficient. In fact, the developer's view is also important in the user stories prioritization. Taking into account the developer's perspective is important to maximize the business value released at the end of every sprint.

Therefore, it is important to adopt comprehensive product owner/development team perspectives according to three parameters: US importance, US priority, and a combination of a US functional size with its structural size. The priority of user stories is defined by the product owner. For example, P(US1)[1] is more prior than P(US2), and P(US2) is more prior than P(US3), etc.

The importance of a user story is defined by the development teams and it can be Essential or Desirable. For example, User stories importance (noted by I(US1) and I(US2))in the same cluster of classes (the same data base, service, etc.) address the

[1] P(US1) that is put the US 1 on top priority.

same importance(*i.e.,* I(US1)= I(US2)[2]then I(US1) and I(US2) have the same importance(regardless of there are Essential or desirable).

The US functional size is determined according to COSMIC FSM method, while the US structural size is determined using the structured size measurement method.

Use of Algorithm 1 provides a basis for prioritizing which user stories to select in the product backlog. The Algorithm 1 can also be used to reorganize user stories in an on-going sprint when requirements changes occur.

Algorithm 1: Prioritizing user stories.

Aim: Prioritizing user stories taking into account inputs defined as below.

Inputs: P(US) the priority of a User Story (US), I(US) the importance of a US, FS(US) the functional size of a US, SS(US) the structural size of aUS.

Outputs: User stories are organized by taking into account their priorities, importance and then their functional sizes and structural sizes.

BEGIN

1. If I(USi) == I(USj) && P(USi)!= P(USj) then
   a. Select the more prior user story (US);
2. Else if I(USi) != I(USj) && P(USi) != P(USj) || P(USi) == P(USj) then
   a. Select the most important (Essential) US
3. Else if I(USi) == I(USj) && P(USi) == P(USj) && FS(USi) != FS(USj) && SS(USi) == SS(USj) then
   a. Select the user story with minimum functional size;
4. Else if I(USi) == I(USj) & P(USi) == P(USj) & FS(USi) == FS(USj) & SS(USi) != SS(USj) then
   a. Select the user story with minimum Structural size;
5. Else if I(USi) == I(USj) && P(USi) == P(USj) && FS(USi) != FS(USj) && SS(USi) != SS(USj) then
   a. Select the user story with minimum Structural size and minimum functional size
6. Else
   a. Select the user story that requires less demand on resources (time or budget)

END

On the other hand, developers identify the status of a user story that can be used to control the development progress. Thus, the status of a user story 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
- **In Progress** is the status of a user story currently being implemented?
- **To Verify** is the status of a user story ready for testing.
- **Done** is the status of a user story tested with success in the customer environment.

When changes occur throughout the Scrum process, it may affect US independently of its status. Thus, we keep only two main status “done” and “undone” (including “new”, “to do”, “in progress”, and “to verify”).

3.5 Step 5: Evaluating Requirements Changes Requests Status

This step is the basis core of the RC evaluation process. It focuses on how to evaluate the status of a Requirements Changes request. RC request may occur within an Ongoing Sprint or an Implemented Sprint. Each identified RC is evaluated separately (*i.e.,* when only FC occurs, or when only SC occurs, or when both FC and SC occur) as depicted in Figure 2. RC may be handled with or without extra cost/time. Consequently, every RC needs to be evaluated based on the Functional Change (FC) and its corresponding Structural Change (SC).

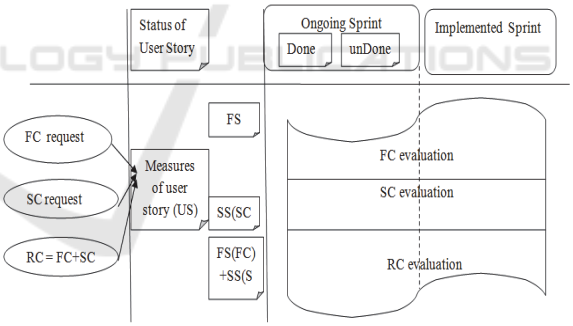

Figure 2: Evaluating requirement changes described in a US Format.

Functional and structural change measurements represent a means for guiding the evaluation process so that the right decision can be made. This evaluation is used later to provide recommendations to the decision-makers to accept, defer or deny a RC request. It depends on three main factors: the FS of the changed US, the SS of the changed US and the US status (done or undone).

Note that a RC request (either FC or SC or both) can be requested by the PO or the development team. First, it should be described in the US refinement format (see section 3.1) to identify the

[2] I(US1) that is put the US 1 on top importance

US affected by the change (referred to as *USc*). The *USc* should have a current status that can be "done" (*i.e.,* this change occurs in an implemented sprint) or "undone"(*i.e.,* this change occurs in an ongoing sprint). In the case of an ongoing sprint, we identify the attributes of the sprint where the change occurs (*e.g.,* the sprint size, start date), and then we measure the FS(FC), the FS(USc), and the functional sizes of all the undone user stories, respectively, the SS(SC), the SS(USc), and the Structural 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(USc), respectively, the SS (SC) and the SS(USc).

3.5.1 Evaluating RC Status in an Ongoing Sprint

When a RC occurs in an ongoing sprint, it may contain both "done" and "undone" user stories. This RC request can be a FC request or SC request or both of them (FC and SC). Thus, we propose to evaluate each RC request separately (*i.e.,* the cases when FC occurs, SS occurs, or both of them).

a) The case when FC occurs

When the status of USc (the user story affected by a change) is either "undone" or "done", we propose to compare the FS(FC) to the functional size of all the undone user stories (*USundone*), respectively, to the functional size of all the done user stories(*USdone*) in the same sprint. Hence, different baselines will be used to evaluate the status of the FC (see Table 3).

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

Table 3: Evaluating a FC request when USc status = un-done/done.

Low	Moderate	High
$FS(FC) = 1CFP$	$2CFP \leq FS(FC) \leq FS(USundone/USdone)$	$FS(FC) > FS(USundone/USdone)$

b) The case when SC occurs

In an analogical way, if the status of the changed user story (USc) is either "undone" or "done", we propose to compare the SS(SC) to the Structural size of all the undone user stories respectively, to the structural size of all the done user stories in the same sprint. Different baselines will be used here to evaluate the status of the SC (see Table 4).

A "High" SC is a change with a structural size greater than the total structural sizes of undone/done user stories in the same sprint. This change will increase the complexity of the code and the costs of the project, since it will have a potential impact on the software development progress. However, a "Low" SC has a structural size of 1 CSM. This change can be handled without any impact on the software development progress. Whereas, a "Moderate" SC is a change with structural size lowest than the structural size of undone/done user stories in the same sprint. It will produce few structural changes. It means that there is few alternative scenario, no complexity in the code, and no impact on the software development progress.

Table 4: Evaluating a SC request when USc status = un-done/done.

Low	Moderate	High
$SS(SC) = 1CSM$	$2CSM \leq SS(SC) \leq SS(USundone/USdone)$	$SS(SC) > SS(USundone/USdone)$

c) The Case when FC and SC occurs simultaneously

When the status of the changed user story (USc) is "undone" or "done", we propose to compare at the same time SS(SC) to the Structural size of all the undone/done user stories, respectively, we compare the FS(FC) to the functional size of all the undone user stories in the same sprint. Table5 shows the RC status evaluation.

A "High" RC is a change with a structural size bigger than the total structural sizes of undone user stories and having a functional size bigger than the total functional sizes of undone/done user stories in the same sprint. It will have a potential impact on the software development progress. However, the RC is considered as "Low" if it has a functional size of 1 CFP and a structural size of 1 CSM. This change can be handled without any impact on the software development progress. Whereas, a "Moderate" RC (summarizing both a SC and a FC) is a change with structural size lowest than the structural size of undone/done user stories and having a functional size lowest than the functional size of undone/done user stories in the same sprint.

Table 5: Evaluating RC (FC and SC) request when USc status = undone/done.

Low	Moderate	High
$SS(SC) = 1CSM$&& $FS(FC)=1$ CFP	$2\ CFP < FS(FC) < FS(USundone/done)$&& $2CSM \leq SS(SC) \leq SS(USundone/USdone)$	$FS(FC) > FS(USundone/done)$ && $SS(SC) > SS(USundone/USdone)$

3.5.2 Evaluating RC Status in an Implemented Sprint

An implemented sprint in the increment product includes a number of done user stories. In this study, we assume that a RC request (including FC request and-or SC request) affecting a "done" US means that the work that has already been done must be changed depending on the change request types (FC or SC or both). Thus, an additional time and effort may be required to take into account this change.

As it is described in section 3.5, requirements change request evaluation is done separately, depending on the nature of the requirement change (functional change or structural change or both of them) in an Implemented Sprint.

a) The case when FC occurs

A "High" FC is a change with a functional size bigger than the functional size of the user story changed in the implemented sprint (USc). An important effort may be required to implement this change. However, a "Low" FC has a functional size of 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.

b) The case when SC occurs

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

c) The case when both FC and SC occur simultaneously

A "High" RC (*i.e.*, FC and SC) is a change with a structural size bigger than the structural size of the user story affected by the change, respectively; its functional size is bigger than the functional size of the user story affected by the change in the implemented sprint (USc). An important effort may be required to implement this change. However, a "low" RC is a change with a "Low" SC having a structural size of 1 CSM, respectively a FC of 1 CFP. This change can be handled without any required effort. However, a "Moderate" RC involves FC having functional size less than the functional size of the user story affected by the change, and the SC having structural size less than the structural size of the user story affected by the change in the implemented sprint. A little effort may be required to implement a "Moderate" change.

3.6 Step 6: Facilitating 'Good' Decision-Making

In this step, we focus on how to facilitate change decisions. We believe that good decision making depends not only on the accuracy of requirements, but also on the use of the right measures. Thus, important decisions are made regarding many criteria including software size measurements. In fact, software size can be used for many exploitations, such as effort/cost estimations, quality, etc. (Abran, 2015).

In addition to these considerations, software size measurement could be viewed from different perspectives during the evaluation process. And therefore, quickly and specific recommendations that decision-makers (*cf.,* product owner, development team and the Scrum master) can apply to contribute to the software project success. Recall that a RC may affect either an ongoing sprint or an implemented sprint. Thus decisions can be made to:

- accept the RC request (FC and-or SC Requests) that is implement the RC in the current sprint.
- deny the RC request (FC and-or SC Requests) that is if the RC proposes a new software, then restarting the development from the beginning.
- defer the RC request to the next sprint (FC and-or SC Requests)that is accept the RC and implement it in the next sprint not in the current one.

3.6.1 Deciding on RC Requests That Occur in an Ongoing Sprint

When the RC request (FC and-or SC) is a modification and affects a US in an ongoing sprint, we propose Algorithm 2that provides recommendations to help in making decisions. These recommendations are based mainly on:

- The comparison between the FS(FC), the functional size of all undone user stories in the current sprint referred to as *FS(USundone)* in Algorithm 2, and the functional size of the changed user story referred to as *FS(USc)* in Algorithm 2.
- The comparison between the SS(SC), the Structural size of all undone user stories in the current sprint referred to as *SS(USundone)* in Algorithm 2, and the structural size of the changed user story referred to as *SS(USc)*in Algorithm 2).

For the purpose of demonstrations, we assume that the FC and SC are both included in the same user stories and in the same sprint. For instance, if the FS(FC) is greater than the total functional size of all the undone user stories. And, if the SS(SC) is greater than the total sizes of all the undone user stories in this same current sprint, we recommend to defer both FC and SC. Then, the user story changed (after the FC and the SC) is deleted from the current sprint and added after modification with respect to the change to the next sprint.

In the case when the FS(FC) is less than the total functional size of all the undone user stories. And, if the SS(SC) is less than the total size of all the undone user stories in the same current sprint, it is required to compare the FS(FC) to the FS(USc) before the change (referred to as FS(USc)i in Algorithm 2), respectively, we compare the SS(SC) to the SS(USc) before the change (referred to as SS(USc)i in Algorithm 2).

Thus, if the FS(FC) is greater than the FS(USc)i. And, if the SS(SC) is greater than the SS(USc), we recommend to defer the FC, the SC and the USc after the change (referred to as (USc)f in Algorithm 2) to the next sprint.

On the other hand, if the FS(FC) is less than the FS(USc)i and if the SS(SC) is less than the SS(USc)i, the decision will be made based on the impact of the change on the FS(USc)i, respectively, the impact of the change on the SS(USc)i.

In the case when 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 size of all the undone user stories.

In the same way, we compare the SS(SC) to the structural size of all the undone user stories in the sprint. Hence, if the FS(FC) is less than the FS(*USundone*) and the SS(SC) is less than the SS(*USundone*), then both the FC and the SC should be accepted. Otherwise, the FC and the SC is deferred to the next sprint.

A deletion of FC request and a SC request do not have any influence on the development progress.

Algorithm 2: Deciding on a RC based FC and SC in an ongoing sprint.

Aim: Deciding on a FC and SC in an ongoing sprint
Require: FS(FC), SS(SC), FS(USundone), SS(USundone), FS(USc), and SS(USc).

BEGIN

1. **if** FS(FC)>FS(USundone) &&SS(SC)>SS(USundone) then
    a. Defer the FC to the next sprint;
    b. Defer the SS to the next sprint;
    c. Delete (USc)i from the ongoing sprint;
    d. Add (USc)f to the next sprint;
2. **else if** FS(FC) <FS(USundone)&&SS(SC) < SS(USundone) then
3. **if** FS(FC)>FS(USc)i && SS(SC)>SS(USc)i then
    a. Defer the FC to the next sprint;
    b. Defer the SC to the next sprint;
    c. Delete (USc)i from the current sprint;
    d. Add (USc)f to the next sprint;
4. **else if**FS(FC)<FS(USc)&& SS(SC)<SS(USc) then
5. **if** FS(USc)f>FS(USc)i && SS(USc)f>SS(USc)i then
6. **If**Remainingtime(USc)f <requiredtime&&teamprogress= early then
    a. Accept the FC;
    b. Accept the SC;
    c. Delete(USc)i from the current sprint;
    d. Add(USc)f to the current sprint;
7. **else**
    a. Defer the FC;
    b. Defer the SC;
    c. Delete (USc)i
    d. Add (USc)f to the next sprint;
8. **else if** FS(USc)f<FS(USc)i && SS(USc)f<SS(USc)i then
    a. Accept the FC;
    b. Accept the SC;
    c. Delete(USc)i from the current sprint;
    d. Add (USc)f to the current sprint;
9. **else if** FS(FC) == 1 CFP &&SS(SC) == 1 CSM then
    a. Accept the FC;
    b. Accept the SC;
    c. Delete (USc)i from the current sprint;
    d. Add (USc)f to the current sprint;
10. **END**

3.6.2 Deciding on RC Requests That Occur in an Implemented Sprint

When one or more user stories are *already* developed in a sprint, any change may involve further discussion with the product owner. After all, the product owner will validate the final product's acceptance test results.

In this case, we suggest to deny the requirements change (whether FC or SC or both of them). However, the Product Owner' needs must be taken to handle his/her requested changes. For this end, we provide some analysis that allows the Product Owner to determine the importance of the change request, such as a comparison among the time spent to implement the changed user story (noted *T*) and the time predicted to re-develop the user story after the change (noted *Tpre*). We propose for this end the analysis below:

If(I(Usa)is essential and T<Tpre) analyze 1:"the US is essential".

If (P(USa) is less than the priorities of the number of all US n divided by2 (P(n/2))and (T<Tpre) analyze 2:"the US is prior".

Else analyze 3: "The US is complicated and must be re-developed".

These analysis does not offer decision-makers with an answer on a RC request, but it highlights when it is required to discuss with the Product Owner the importance of the RC(whether it is really needed or not.)

4 CASE STUDY

The case study "IT-commerce" is developed by a team of engineers. The web site allows the customer to buy IT equipment on-line. This web site has been delivered after six sprints each one lasts for two weeks. It includes initially ten user stories. We provide the detailed measurement results that are given in Table 6.

The development team defined the first sprint S1 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 noted by "E" 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 6, US4 is with importance = "E" and priority = P2. Whereas, US8, US9 and US10 are with importance = Essential and priority = P3. Hence, the development team starts by US4.

After the implementation of US4 (*i.e.,* the US4 status is done), the PO proposes to add US11, US12, and US13(See Table 7). Where:

FS(US11) = 4 CFP, FS(US12) = 3 CFP,FS(US13) = 3 CFP. And, SS(US11) =2 CSM, SS(US13)= 3CSM and the importance/priority for all the added user stories is (Essential/P3). Note that the status of the user stories initially in the sprint is: US4 status = done and US8 status = US9 status = US 10 status = undone. (Colored in red in the table 7)

The question here is whether to accept the RC that may include FC and-or SC (in other words, implement RC in the current sprint S2 or defer the (FC and-or SC) to the next sprint S3).To do this, we must apply our proposed an in-depth process.

First, we apply the step1, 2 and 3 respectively (refinement and grooming step, mapping step and sizing step) Table 6 lists these results.

Second, we **apply the step 4** in which user stories are prioritized based not only on their priority and importance but also on their functional and structural sizes. For this end, based on the proposed Algorithm 1, the results of step 4 are provided in Table 7.Hence, the user stories will be organized as follows: US9, US12, US11, US8, US13, and US10.

After that, we apply step5 to evaluate the RC status. Recall that the RC (FC and-or SC)in this case occur in an ongoing sprint and propose the addition of three user stories (US11,US12,US13) without changing any user story in the sprint This RC is considered as a moderate change (see Table 5)

Finally, **we apply the step 6**: The total functional size of undone user stories has the value of 12 CFP (US8, US9, and US10). While, the FC has a functional size of 10 CFP (US11, US12, and US13). The total structural size of undone user stories has the value of to 4 CSM (US8, US9, and US10). While the SC has a structural size of 5 CSM (US11, US12, US13)

Thus, after applying Algorithm 2, we make the following decisions:

- ✓ Accept US 9(FC and SC), US 12 (FC and SC), and US 11(FC and SC),
- ✓ Defer US 8(FC and SC), US 13 (FC and SC), and US 10 (FC and SC), to the next sprintS3.

Table 6: Product Backlog initially organized.

	ToDo LIST					
Sprint	User Stories	FS(US) CFP	SS(US) CSM	Status	I (US)	P(US)
S_n	US1	3	2	New	D	P3
	US 2	3	2	New	D	P3
	US 3	3	0	New	D	P3
S2	US 4	3	1	New	E	P2
S1	US 5	4	2	New	E	P1
	US 6	6	2	New	E	P1
	US 7	6	2	New	E	P1
S2	US 8	3	2	New	E	P3
	US 9	3	1	New	E	P3
	US 10	6	0	New	E	P3

Table 7: Product Backlog after change.

	ToDo LIST					
Sprint	User Stories	FS(US) CFP	SS(US) CSM	Status	I(US)	P(US)
S_n	US1	3	2	New	D	P3
	US 2	3	2	New	D	P3
	US 3	3	0	New	D	P3
S2	US 4	3	1	Done	E	P2
S1	US 5	4	2	done	E	P1
	US 6	6	2	done	E	P1
	US 7	6	2	done	E	P1
S2	US 8	3	2	undone	E	P3
	US 9	3	1	undone	E	P3
	US 10	6	0	undone	E	P3
S2	US11	4	2	New	E	P3
	US12	3	0	New	E	P3
	US13	3	3	New	E	P3

5 THREATS TO VALIDITY

The proposed process in this paper has been illustrated through the case study "IT-commerce". However, the validity of the above results is 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 process 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 RC evaluation is required especially for an urgent RC request. The second issue is related to the productivity of the development team. In fact, two functional processes with exactly the same functional size or the same structural 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 other criteria. 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 (deletion, addition or modification). However, we consi-der 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 and the structural size of other use stories. For

Table 8: Summary of the detailed user story refinement description format.

User story US(Sellami et al., 2018)	**/*description*/** **As a** <UserType>**I want to** <Action><Object>**so that** <value or expected benefit><NFR>	**/*Scenario nominal description*/** <User/System><ActionType: Entry, Read, Write, eXit, Expletive> <DataTransferred><Action>	
User story with refinement different alternatives scenario	**/*description*/** **US** **/*Scenario alternative description 1*/** **<If (condition)>** **<Then >** <User/System><ActionType: Entry, Read, Write, eXit,Expletive> <DataTransferred><Action>	**/*description*/** **US** **/*Scenario alternative description 2*/** **<If (condition)>** <User/System><ActionType: Entry, Read, Write, eXit, Expletive><DataTransferred> <Action> **<else >** <User/System><ActionType: Entry, Read, Write, eXit, Expletive><DataTransferred ><Action>	**/*Scenario alternative description 3*/** **<If (condition)>** <User/System><ActionType: Entry, Read, Write, eXit, Expletive><DataTransferred><Action> **<else if>** <User/System><ActionType: Entry, Read, Write, eXit, Expletive><DataTransferred><Action> **<else >** <User/System><ActionType: Entry, Read, Write,eXit, Expletive><DataTransferred><Action>
User story with refinement iterative scenario	**/*description*/** **US** **/*Scenario iterative description 4*/** **<Loop (condition)[n]>** <User/System><ActionType: Entry, Read, Write, eXit, Expletive><DataTransferred><Action> **<end loop>**		

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 process and the making decision algorithms. The first issue is the limited number of case studies used to test the proposed process. In fact, only one case study has been used: the 'IT-commerce". Thus, testing the proposed process and algorithms in an industrial environment is required in order to get the feedback from the practitioners.

6 CONCLUSION AND FUTURE WORK

This work proposed an in depth requirement change evaluation process based on the use of US functional and structural size measurement methods. Thus, user stories are expressed in terms of CFP units using the standard COSMIC FSM method, and respectively, in terms of CSM units using the structural size measurement method. In addition, Requirement Changes are measured and evaluated so that the decision-makers will be guided to decide which RC request should be accepted, deferred or denied.

Finally, other works will focus necessary on the automation of this in-depth requirement change evaluation process and a tool will be so promoted to help making the right decision that can be used by the PO in the Scrum method. This tool may be developed as an API to be integrated into JIRA or any other developed solutions for software organization

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.

Abdalhamid, S. and Mishra, A., 2017. Adopting of agile methods in software development organizations: systematic mapping. *TEM Journal*, *6*(4), p.817

Al Salemi, A. M. and Yeoh, E. T. (2015). A survey on product backlog change management and require- ment traceability in agile (Scrum). In *the 9th Malay- sian Software Engineering Conference (MySEC)*, pa- ges 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*.

COSMIC (2020). *The COSMIC Functional Size Measurement Method, Version 5.0,Announcement of Version 5.0 of the COSMIC Measurement Manual – March 31, 2020*

Drury-Grogan, M., O'Dwyer, O.: An investigation of the decision-making processin agile teams. *Int. J. Inf. Technol. Decis. Mak.* ***12****(6),* 1097–1120 (2013)

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 *2011Joint Conference of the 21st International Workshop onSoftwareMeasurementandthe6thInternationalConference on Software Process and Product Measurement*, pages269–272.

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 SocietyPr.

Furtado, F., Zisman, A.: Trace++ (2016): a traceability approach to support transitioningto agile software engineering. In: *The 24th International Requirements Engineering Conference (RE), pp. 66–75.*

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 measu- rement. *Science of Computer Programming, Special IssueonAdvancesinSoftwareMeasurement*,135:88– 104.

Hamed, A.M.M and Abushama, H.Popular Agile Approaches *in Software Development: Review and Analysis. Computing, Electrical and Electronics Engineering (ICCEEE), 2013 International Conference on (2013), pp. 160-166.*

Ken Schwaber and Jeff Sutherland, *TheScrum Guide™ The Definitive Guide to Scrum: The Rules of the Game November 2017.*

Lloyd, D., Moawad, R., and Kadry, M. (2017). A supporting tool for requirements change management in distributedagiledevelopment.*FutureComputingandInformatics Journal*, 2(1):1–9.

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

Sellami, A., Hakim, H., Abran, A., and Ben-Abdallah, H. (2015). A measurement method for sizing the struc- ture of UML sequence diagrams. *Information & Soft ware Technology, 59:222–232.*

Sellami, A., Haoues, M., Borchani, N., & Bouassida, N. *(2018, July).* Guiding the Functional Change Decisions in Agile Project: An Empirical Evaluation. *In International Conference on Software Technologies (pp. 327-348). Springer, Cham.*

Sellami, A., Haoues, M., Borchani, N., & Bouassida, N. Orchestrating Functional Change Decisions in Scrum Process using COSMIC FSM Method. *ICSOFT 2018: 516-527*

Sellami, A., Haoues, M., Borchani, N., & Bouassida, N.Towards an Assessment Tool for Controlling Functional Changes in Scrum Process. *IWSM-Mensura 2018: 34-47*

Shalinka Jayatilleke, Richard Lai, *A systematic review of requirements change management , Information and Software Technology 93 (2018) 163–185*

Stålhane,T.,Hanssen,G.K.,Myklebust,T.,andHaugset,B. (2014). Agile change impact analysis of safety cri- tical software. *In Bondavalli, A., Ceccarelli, A., and Ortmeier,F.,editors,ComputerSafety,Reliability,and Security, pages444–454.*

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 Publisher.