MPED-SCRUM: An Automated Decision-Making Framework Based Measurement for Managing Requirement Change Within the SCRUM Process

Hela Hakim[1], Asma Sellami[2] and Hanêne Ben-Abdallah[3]
[1]University of Sfax, Faculty of Economics and Management of Sfax, MIR@CL Laboratory, Tunisia
[2]University of Sfax, Higher Institute of Computer Science and Multimedia, MIR@CL Laboratory, Tunisia
[3]Higher Colleges of Technology, CIS Faculty, Abu Dhabi, U.A.E.

Keywords: Agile, Prioritizing, Functional Change, Structural Change, Functional Size, Structural Size, COSMIC FSM Method, Structural Size Measurement SSM Method, Scrum, MPED-SCRUM Framework.

Abstract: In Scrum-based projects, delivering precise assessments of requirement changes to stakeholders holds paramount importance for effective software project management. Accurate evaluations empower stakeholders to make informed decisions, preventing costly misunderstandings and fostering shared expectations. The integration of Software Size measurements has significantly contributed to achieving this goal. This paper endeavors to automatically enhance the accuracy of evaluating requirement changes and prioritizing tasks within the Scrum process by leveraging the standardized COSMIC FSM (ISO 19761) along with its extended Structural Size Measurement method. The proposed automated framework, named 'MPED-SCRUM,' stems from the automation of the requirement change evaluation process based on Measuring, Prioritizing, Evaluating and Deciding on requirement change, incorporating both functional and structural change. MPED-SCRUM proves beneficial not only for the Administrator but also for the Scrum Master and Product Owner in effectively managing team members (Module 1). Furthermore, the framework aids both the Scrum Master/Product Owner and development teams in efficiently handling sprint backlogs and user stories (Module 2). Lastly, MPED-SCRUM facilitates the measurement, prioritization, evaluation and decisions making of requirement change requests at two granularity levels – functional and structural. This capability empowers stakeholders to make informed decisions regarding the acceptance, deferral, or denial of a change request (Module 3).

1 INTRODUCTION

Accurate evaluation of requirements changes is a crucial aspect of managing software projects. Agile methodologies, such as Scrum and Kanban, were specifically designed to address the challenge of managing projects with frequent changes.

The 14th Annual State of Agile Report shows that Agile adoption has increased by 33% in response to changing market. 60% of respondents claimed that Agile has assisted their speed to market. Agile has proven to be one of the methodologies capable of solving complex issues and adapting rapidly to business changes in the era of agility and rapid transformation by staying close to the customers (Abran, 2015).

Despite the use of Agile methodology, some problems persist. In software project, the scope is considered as one aspect that can directly affect the budget and timing (Abran, 2015). In fact, the project success is most often based on cost-time tradeoffs. However, the scope seems to be one of the most neglected domains in agile and conventional. Agile is recognized for its rapid improvement as well as its willingness to embrace change.

Every change needs to be carefully evaluated because it can affect the project's time and budget. More clearly the scope is the more tendance to the project success.

This paper aims to study the challenges in evaluating and decision-making on requirements changes automatically based on change functional and structural sizes.

Like levels in agile methodology (in scope particularly), requirement change also have different levels of details. Some authors make the difference between technical change requirements and

Hakim, H., Sellami, A. and Ben-Abdallah, H.
MPED-SCRUM: An Automated Decision-Making Framework Based Measurement for Managing Requirement Change Within the SCRUM Process.
DOI: 10.5220/0012696500003687

In *Proceedings of the 19th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2024)*, pages 571-581
ISBN: 978-989-758-696-5; ISSN: 2184-4895

functional change requirements levels (Haoues et al, 2018) (Stålhane et al., 2014), while we make the difference between functional change and structural change levels in our previous works (Hakim et al., 2020). Indeed, the structural level was proposed by (Sellami et al., 2015) to highlight the control structures of functional requirements/changes. That means that detailed functional requirements can be measured through their control structures when available.

An efficient requirement change management is paramount. Such management must be based on an in-depth requirement change evaluation process, which in turn requires a measure-driven process (Hakim et al, 2020). The change evaluation process is composed of a set of activities: (i) measuring the requested change that is expressed in the form of a user story (US) at both functional and structural levels, (ii) prioritizing the measured US, (iii)evaluating the measured US, and finally (iiii) making decisions. In our previous work (Hakim et al, 2020), we proposed an in-depth requirement change evaluation process to support the requirements change at two levels of granularity (functional and structural levels). This paper aims to improve and extends the previous work of (Hakim et al., 2020) by proposing an automated framework named 'MPED-SCRUM'.

The 'MPED-SCRUM' automated framework can be used for managing changes in requirements at different levels of granularity within the Scrum Process. In accordance with our findings (Hakim et al., 2020), this automated framework helps in measuring, prioritizing, and evaluating requirements changes using functional and structural size measurements. It also covers members' management, sprint change management, user stories change management. The users of this framework could be the product owner, the scrum master, and finally the development teams (analyst, designers, developers, testers). Each team member has a different role so that the designers/developers and testers are the most involved users to our framework.

The rest of this paper is organized as follows. Section 2 provides an overview of software size measurement methods and the SCRUM framework. Section 3 discusses the related work. Section 4 provides our proposed automated framework 'MPED-SCRUM' used for managing requirements changes at functional and structural levels, which is based on the in-depth requirement change evaluation process. Finally, Section 5 summarizes the presented work and outlines some of its possible extensions.

2 BACKGROUND

This section describes an overview of the COSMIC FSM method, the SSM method, the Scrum, and finally the prioritizing techniques in the Agile environment.

2.1 COSMIC FSM Method

The Common Software Measurement International Consortium (COSMIC) is an international FSM method designed to be independent of any implementation decisions embedded in the operational artifacts of the software to be measured.

The COSMIC sizing process for measuring the functional requirements size of the software is composed of three phases: the measurement strategy phase, the mapping phase, and the measurement phase (COSMIC v5.0. 2021). The COSMIC sizing is based on measuring functional processes (FP). Each FP is composed of a set of functional sub-processes that may be either a data movement or a data manipulation. There are four types of data movements: Entry (E), eXit (X), Read (R), and Write (W).

- An Entry moves a data group into a FP from a functional user.
- An eXit moves a data group out of a FP to a functional user.
- A Write moves a data group from a FP to persistent storage.
- A Read moves a data group from persistent storage to a FP.

A data group is a set of attributes that describes one object of interest. The COSMIC measurement unit is one data movement of one data group indicated as one CFP (COSMIC Function Point). The size of a functional process is determined by the sum of the data movements it includes. The functional size of a functional process, noted by FS(FP), is given by Equation 1 .

$$FS(FP)=\sum FS(Entries)+\sum FS(eXits)+\sum FS(Reads)+\sum FS(Writes) \quad (1)$$

COSMIC defines a functional change as "any combination of additions of data movements or of modifications or deletions of existing data movements" (COSMIC v5.0, 2021). The size of a functional change presenting in a functional process is the sum of its data movements that have been added, deleted, and modified. The software functional size after the change is the sum of the sizes of all the added data movements minus the size of all the removed data movements.

2.2 The Structural Size Measurement SSM Method

As an extension to the COSMIC FSM method, the Structural Size Measurement (SSM) method is a measurement method essentiality designed to address the challenge of more detailed measures to quantify data manipulation in a software product. In an analogical way to the COSMIC, the process for measuring the SSM method is composed of three phases: Measurement strategy phase, mapping phase, and Measurement phase. The SSM was proposed by (Sellami et al., 2015) for UML sequence diagrams. It was designed by following the measurement process recommended by (Abran, 2010).

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

The SSM defines a Structural change as "any combination of additions of data manipulation or of modifications or deletions of existing data manipulation" (Hakim et al., 2017). The size of a Structural change within a functional process (including structural aspect) is the sum of its data manipulations that have been added, deleted, and modified. The software structural size after the change is the sum of the sizes of all the added data manipulations minus the size of all the removed data manipulations.

2.3 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 (Schwaber et al., 2004). 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. Using Scrum a better communication across the development team and the product owner was observed. 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 (Schwaber et al., 2004). Scrum appears to work better with teams of 5 to 9 people, with large projects being typically handled by several scrum teams (Schwaber et al., 2004).

Nevertheless, some companies adapt Scrum for large- scale projects (Dikert et al., 2016). The Scrum process 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 the initial requirements as described by the stakeholders and customers. The number of stories may increase or decrease 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 (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.

3 RELATED WORK

Researchers and practitioners agree that agile development provides a rapid response methodology to handle requirements changes (Abran, 2015). Thus, many research studies have addressed the issues of managing requirement changes in the Scrum process.

In Scrum, a change priority in the backlog is the role of the Product Owner. While the product owner selects what items are included in the product backlog, the development team has the final say on how the items are executed in the sprint backlog. This implies that the items of the backlog are ordered by priority. The issue of prioritizing and managing items (or changes) and their automation has received an increased interest in recent years. This section presents the works related to both exploratory and experimentally change management processes.

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 collection, and selection of the best alternative. Decisions are often made in a collaborative manner that may be influenced by three main factors, according to (Drury-Grogan et al., 2013): Sprint duration, experience, and resource availability.

Table 1: Summary of the exploratory and experimentally proposals, focusing on managing changes in Scrum.

Study	Focus	Findings
(Commeyne et al.,2016)	Evaluation of teams' productivity using COSMIC	COSMIC is more reliable in estimating models with much smaller variances
(Alsalemi and Yeoh, 2015)	Product backlog change management And requirement traceability	Lack of requirement change traceability
(Sellami et al., 2018)	Evaluation of functional changes in Scrum process using COSMIC	Quantify FC request to make appropriate decisions
(Stålhane et al., 2014)	Impact of technical changes in safety requirements	A supporting tool that ensures the validity of safety
(Lloyd et al., 2017)	Requirements change management in distributed agile development	A supporting tool
(Sellami et al., 2018)	Orchestrating Functional Change Decisions in scrum Process using COSMIC	Tools to Quantify FC request to make appropriate decisions
(Hakim et al., 2020)	In-Depth Requirements Changes Evaluation Process based on functional and Structural size methods	Quantify RC at functional and structural levels to help in making accurate decisions
Our study	An automated framework based on functional and structural sizes to manage requirements' changes at functional and structural levels within the SCRUM	Automating the Process of managing changes in SCRUM using functional and structural measures

However, the final decisions are usually made based on judgment according to the team members' experiences. Although experts' 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 use an objective change evaluation process that is based on the standardized COSMIC FSM method and its extension SSM method. Measurement results should be accurate, and cover the functional and structural levels sizes.

In addition, there are many studies that addressed the issues of managing requirements changes automatically 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. To provide with an accurate evaluation of changes, we proposed an in- depth Requirements Change Evaluation Process Using Functional and Structural Size Measures in the Context of Agile Software Development (Hakim et al., 2020). This process takes into account the different levels of requirements/ changes that impact the success of their management and the whole project. It means that, when changes are evaluated at different levels of details (functional and structural levels), appropriate decisions can be made by stakeholders.

The findings proposed by (Commeyne et al.,2016), (Alsalemi and Yeoh, 2015), (Sellami et al., 2018), and (Hakim et al., 2020)) are exploratory researches, while the findings proposed by (Lloyd et

al., 2017), (Stålhane et al., 2014), (Sellami et al., 2018), and our study) are experimental researches (See Table 1).

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 a 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. Indeed (Sellami et al., 2018) extended their work and proposes tools to support the change management using the COSMIC method at functional level. In this paper, we believe that an automated framework to support requirement change requirements at functional and structural levels respectively based on COSMIC and SSM measurement methods within the Scrum process will be a very interesting challenge.

4 AUTOMATED FRAMEWORK FOR MANAGING REQUIREMENTS CHANGES

To manage requirements changes at both functional and structural levels we propose an automated framework 'MPED-SCRUM' which is simple and efficient. This framework can be used to manage members profiles, manage sprints (Add, modify), manage user stories (add, delete, modify), evaluate changes through an in-depth requirement change evaluation process based on functional and structural size measurement methods.

Figure 1 presents our proposed automated framework 'MPED-SCRUM' at a high level of abstraction. MPED- SCRUM can be used by an Admin, Product Owner, Scrum Master, and the development teams (including analyst, designer, developers, and testers).

Figure 1: The proposed automated framework.

This framework is composed of five parts that are categorized into three modules as follows.

Module1: Members Management

1. Admin Authentication
2. Members Authentication Module2: Sprint /User stories change Management
3. Sprint Management
4. User stories Management Module3: Requirement Change Evaluation
5. Measuring, Prioritizing; Evaluating, and Deciding on requirement change

4.1 Module1: Members Management

Once the roles of each member are defined, authentication and authorization are required for a minimum level of security, while dealing with access to any sensitive data. The lack of security typically results in conflict issues, often legal claims between the stakeholders and product owner, and ultimately, a dissatisfied customer. The UML use case diagram defines the behavior, conditions, and constraints the first module (see Figure 2).

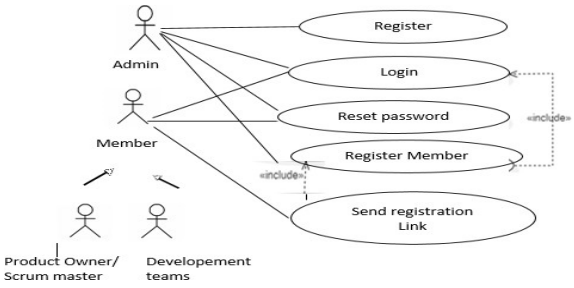

Figure 2: Members management (Register) use cases.

The users of this module are:

- Admin: is the person who connects through the MPED-SCRUM account.
- Scrum Master/product Owner: is a person having a limited access to the application components.
- Development Teams: is a group of persons composed of analysts, developers and testers having a very limited access to the application components.

Note that all the participants (Scrum Master/product Owner and the development team) are a generation of the member actor.

Authentication is the process of validating the identity of a registered user attempting to get access to the application. Besides, MPED-SCRUM tool offers the following functionality: the user who registers directly from the home page, he is expected to be an Admin. After authentication, the Admin can manage information about each team member (developers, analysts, designers, Project Managers/Product Owner, Testers, etc.) in the member component.

- **Admin Authentication**

From the home page, user can register by filling the Register form (See Figure 3).

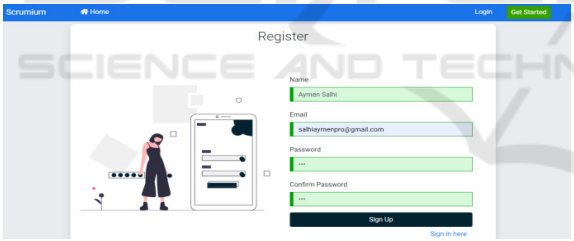

Figure 3: Admin Registration form.

The user will be redirected to the Admin profile (See Figure 4).

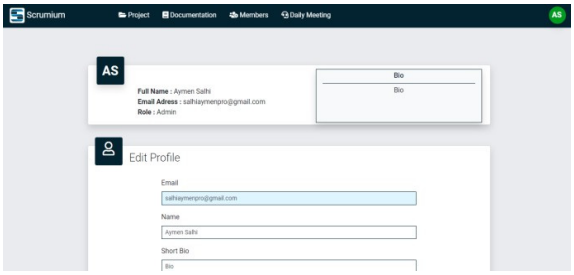

Figure 4: Admin Profile view.

Once the user 'Admin' is authenticated, he has the right to access to all the application components. Authorization here is about deciding whether this user is permitted to perform a given action on a specific resource.

- **Member Authentication**

Recall that a project member may be a Scrum Master/Product Owner or a development teams (analyst, designer, developers, and testers). Member's authentication starting by sending a registration form via email. To achieve this goal, we used MailTrap to test the emails that admin want to send it to the others members. The member just needs to enter the member email and click "Send Registration Link" button (See Figure 5 and Figure 6).

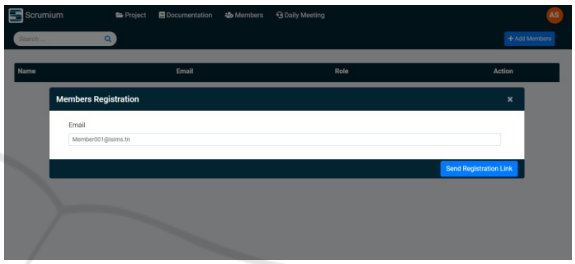

Figure 5: The member email.

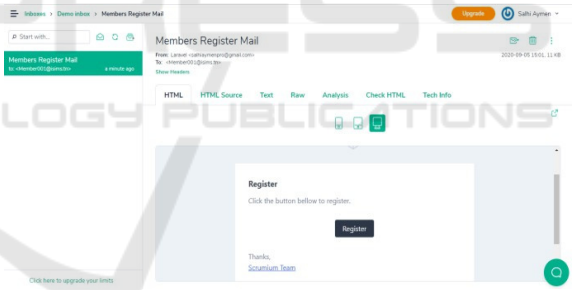

Figure 6: The send Registration link.

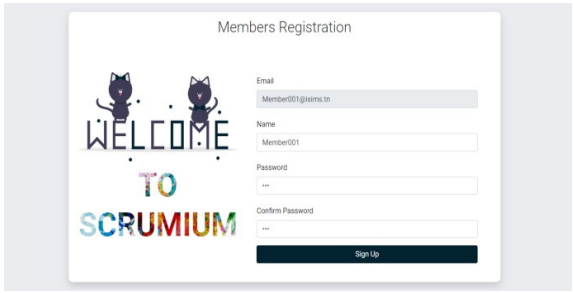

Figure 7: The Member registration Form.

Once the Member clicks the Register button, a registration form will be opened (Figure 8).

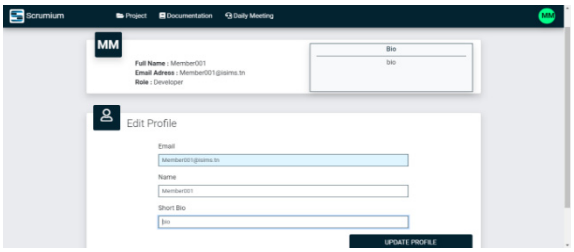

Figure 8: The Member Profile View.

Thereafter, the user will be registered as a team member (analyst, designer, developer, testers), and the Admin can change his role later to Scrum Master /Product Owner or another role (See Figures 7 and 8).

After the member's registration, the admin will get the team members list, and he/she can manage his/her members easily (See Figure 9).

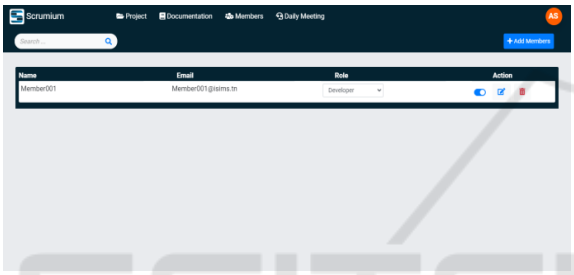

Figure 9: The members list example.

4.2 Module 2: Sprint /User Stories Change Management

In this second module, we focus on implementing the sprint management (Add/Modify the sprint backlog) and the user story management (Add the user stories at functional and structural levels, add/modify/delete users stories, and create the sprint backlog)

Figure 10 presents the use cases of this module and its corresponding actors.

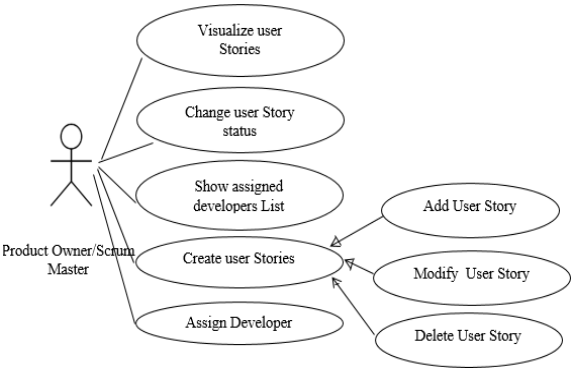

Figure 10: Sprint/users stories change management use cases Diagram.

- **Sprint/ User Stories Change Management**

Recall that the "user story format" of this work is based on a detailed textual description deduced from our previous work (Hakim et al., 2020).

The detailed textual description of a user story highlights both the functional and structural aspects of a feature.

The functional aspect of a feature is presented in Figure 11.

Regarding the User stories format, an additional form fields are required. Figure 11 presents the global format of the US.

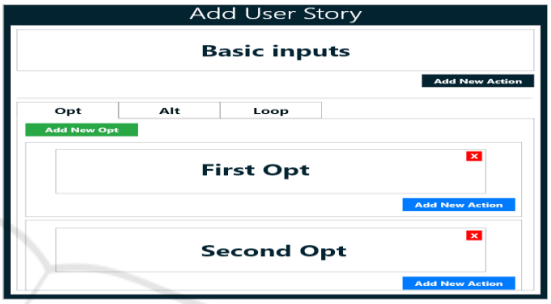

Figure 11: The global format of the US.

To add some parts to this form dynamically, we used Angular 7 FormArray API, such as creating nested forms. Figure 12 presents the structure of the data model form.

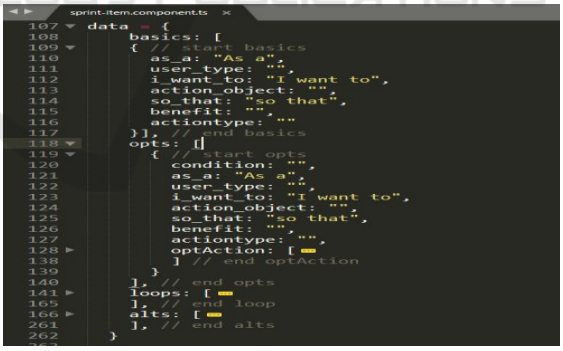

Figure 12: The structure of the data model form.

4.3 Module 3: Requirement Change Evaluation

In this third module, we focus on the main part of our automated framework in which we implement the Requirement Change Evaluation process. This process is based on implementing the **M**easurement, **P**rioritizing and **E**valuation and **D**eciding on RC within scrum (That is why it's called the MPED-SCRUM).

Figure 13 presents the different use cases of this module. Here, the actor is the development team (analysts, designers, developers, and testers).

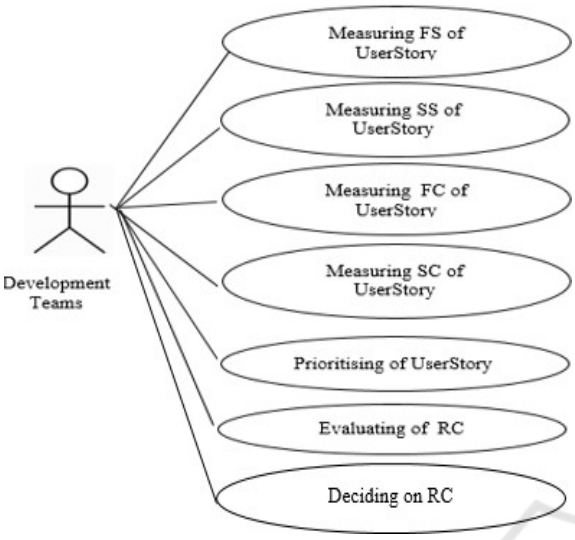

Figure 13: Requirement Change Evaluation Use Case Diagram.

4.3.1 Measuring Users Stories

Figure 14 presents a screenshot on measuring users stories (with refinement formats) at functional level and structural level

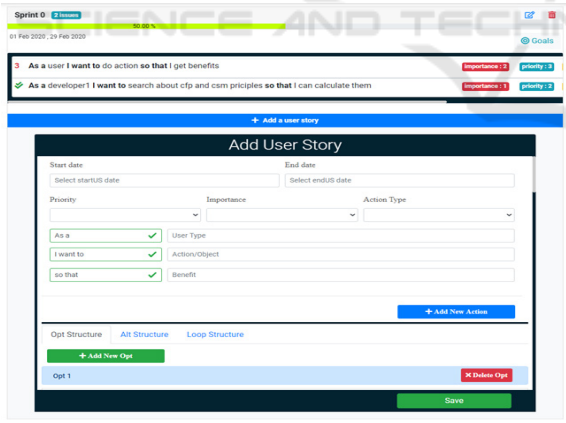

Figure 14: User Story Add Form and resultList.

4.3.2 Prioritizing Users Stories

Automating the user story size at different levels of granularity (functional and structural levels) is a critical task of the evaluation part. Thus, using MPED-SCRUM, the users story size can be generated when a requirement change occurs. An automated prioritization process can allow members to easily manage, monitor, and update priorities as user stories are completed, modified, deleted, or new users stories are added (i.e., when requirements changes are requested and can be expressed in the form of user stories at functional and structural levels).Algorithm 1 presents how to prioritize the requirements changes using both the COSMIC functional and structural size measurement methods. Each Functional change is evaluated using the COSMIC FSM method, while each Structural change is evaluated using SSM method.

Four basic values (Priority, Importance, CFP, and CSM) are used for running 'prioritizing user stories' algorithm, and therefore implementing the decision-making.

Aim: Prioritizing user stories taking into account the following inputs: P(US), I(US), FS(US),and SS(US)
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 a US.

Outputs:

User stories are organized by taking into account their priorities, importance, and their functional and structural sizes (Hakim et al, 2020).

If *P(USi) != P(USj)* **then**
Select the more prior user story (US);
Else if *P(USi) == P(USj) & I(USi) != I(USj)* **then** Select the most important (Essential) US ;
Else if *P(USi) == P(USj) & I(USi) == I(USj) & FS(USi) != FS(USj)* **then**
Select the user story with minimum functional size;
Else if *P(USi) == P(USj) & I(USi) == I(USj) & FS(USi) == FS(USj) & SS(USi) != SS(USj)* **then**
Select the user story with minimum Structural size;
Else
Select the user story that requires less demand on resources (time or budget);
End

Algorithm 1: Prioritizing user stories.

4.3.3 Evaluating the Requirement Change

The evaluation process focuses on how to evaluate the status of a requirement change request. Table 4, 5, and 6 present the FC, the SC, and RC (both of FC and

SC) requests evaluations, respectively. This evaluation helps therefore in making appropriate decisions about whether to accept, defer, or deny a change request. Algorithm 2 presented below is the proposed algorithm to make such decision.

Table 2: Evaluating a FC request when USc status = undone/done (Hakim et al, 2020).

Low	Moderate	High
FS(FC)=1CFP	2CFP≤FS(FC)≤FS(USundone/USdone)	FS(FC)>FS(USundone/USdone)

Table 3: Evaluating a SC request when USc status = undone/done. (Hakim et al, 2020).

Low	Moderate	High
SS(SC)=1CSM	2CSM≤SS(SC)≤SS(USundone/USdone)	SS(SC)> SS(USundone/USdone)

Table 4: Evaluating RC (FC and SC) request when USc status = undone/done (Hakim et al, 2020).

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

4.3.4 Deciding on the Requirement Change

The assessment of software size at different level of granularity serves not only for effort/cost estimations but also for decision-making, such as budgetary and portfolio decisions (Abran, 2010). In this section, we present in algor ithm 2 below a set of actions for decision-makers (e.g., product owner/ scrum master, development team) to aid in making decisions regarding a Functional Change (FC) request respectively a Structural Change (SC). It is essential to note that an FC respectively an SC may affect an ongoing sprint or one that has already been implemented. The decisions are as follows:

- Accept the FC request and SC request, signifying the implementation of the RC in the current sprint.
- Deny the FC request and SC request, an action taken only if the RC proposes a new software, necessitating a restart of development from the beginning.
- Defer the FC request and SC request to the next sprint, implying acceptance of the RC and its implementation in the subsequent sprint rather than the current one.

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
If FS(FC)>FS(USundone) &&SS(SC)>SS(USundone)**then**
Defer the FC to the next sprint;
Defer the SS to the next sprint;
Delete (USc)i from the ongoing sprint; Add (USc)f to the next sprint;
Else if FS(FC) <FS(USundone)&&SS(SC) < SS(USundone) **then**
If FS(FC)>FS(USc)i && SS(SC)>SS(USc)i **then**
Defer the FC to the next sprint;
Defer the SC to the next sprint;
Delete (USc)i from the current sprint;
Add (USc)f to the next sprint;
Else if FS(FC)<FS(USc)&& SS(SC)<SS(USc) **then**
If FS(USc)f>FS(USc)i && SS(USc)f>SS(USc)i **then**
If Remainingtime(USc)f <requiredtime&&teamprogress= early **then**
Accept the FC;
Accept the SC;
Delete(USc)i from the current sprint;
Add(USc)f to the current sprint;
Else
Defer the FC;
Defer the SC; Delete (USc)i
Add (USc)f to the next sprint;
Else if FS(USc)f<FS(USc)i &&SS(USc)f<SS(USc)i then
Accept the FC; Accept the SC;
Delete(USc)i from the current sprint; Add (USc)f to the current sprint;
Else if FS(FC) == 1 CFP &&SS(SC) == 1 CSM then
Accept the FC;
Accept the SC;
Delete (USc)i from the current sprint;
Add (USc)f to the current sprint;
End
Algorithm 2: Deciding on a RC.

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

5 CONCLUSION

In this study, we investigated the problem of automatically managing and evaluating changes

within the SCRUM process. The automated MPED-SCRUM framework includes three modules based on the in-depth change evaluation process that support the requirement change evaluation at functional and structural levels. The change evaluation process is based on measuring the size of requirements changes requests that are expressed in the form of user stories at both functional and structural levels. The knowledge of the change size helps in prioritizing and evaluating changes, and finally making the right decision about accept, deny or delete such changes. It can be used by Scrum master, development teams to meet clients' changes requests at different levels of details. For further works, we deploy this framework particularly the second module as an API to be integrated into the most famous automated Framework like JIRA or any other developed solutions for software organizations. Futures works will be focused in using the artificial intelligence as the Smartest automated tool for Decision Makers.

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

Alsalemi, 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)*, 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.

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

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 on Software Measurement and the 6th International Conference 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 andLeadingSoftwarePro-jects*.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 measurement. *Science of Computer Programming, Special IssueonAdvancesinSoftwareMeasurement*,135:88–104.

Hakim, H,.Sellami, A., and Ben-Abdallah, H. (2020). An in- Depth Requirements Change Evaluation Process using Functional and Structural Size Measures in the Context of Agile Software Development. In ICSOFT (pp. 361-375).

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.

Download the official Scrum GuideLloyd, D., Moawad, R., and Kadry, M. (2017). A supporting tool for requirements change management in dis- tribute agile development. Future Computing and Informatics 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 structure of UML sequence diagrams. Information & Software 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)