A Framework for Requirements Prioritization Process in Agile Software Development

Khaled AbdElazim[1], **Ramadan Moawad**[2] **and Essam Elfakharany**[3]

[1] College of Computing and Information Technology, Arab Academy for Science, Technology and Maritime Transport University (AASTMT), Cairo, Egypt
[2] Faculty of Computers and Information Technology, Future University in Egypt, Cairo, Egypt
[3] College of Computing and Information Technology, Arab Academy for Science, Technology and Maritime Transport University (AASTMT), Cairo, Egypt

resalasoft@gmail.com, ramdan.mowad@fue.edu.eg and essam.fakharany@gmail.com

Abstract. Requirements engineering is a crucial phase of software engineering, and requirements prioritization is an essential stage of requirements engineering particularly in agile software development. Requirements prioritization goals at eliciting which requirements of software need to be covered in a particular release. The key point is which requirement will be selected in the next iteration and which one will be delayed to other iterations for minimizing risk during development and meeting stakeholders' needs. There are many existing techniques for requirement prioritization, but most of these techniques do not cover continuous growth and change of requirements or cover requirements dependencies. So, most of these prioritization techniques need to be more continuous, scalable, implemented merely and integrated with software development life cycle and not work separately. This paper introduces a framework to prioritize requirements in agile software development. This framework tries to find solutions for the challenges facing this prioritization process such as how to make this prioritization continuous and scalable and how to deal with rapidly requirement changes and its dependencies.

1. Introduction

Software systems have become the backbone of most business operations; however, business requirements are rapidly changing, and it is impossible to fulfill the requirements at once. Agile techniques are used to lessen the negativity of these problems by using applying user's necessities regularly and iteratively through deciding on a number of requirements after being prioritized to be implemented in iterations. This may be accomplished with the aid of appearing continuous requirements prioritization. Software product management use Requirement prioritization for identifying which candidate requirements of software ought to be selected in a selected release and decrease the risk for the duration of developing. Requirement prioritization has many advantages regarding all software development lifecycle style. Prioritization helps the implementation of software with special needs of stakeholders [1]. Additionally, the challenges associated with software system development like restricted resources, restrained budget, and insufficiently expert programmers among others create requirements prioritization essential. This can lead to better planning of software releases because not all the needs can be applied in one version because of some of the issues [2].

Requirement prioritization has a vital role in developing software as it improves its planning, budget control, and scheduling. Consequently, requirements prioritization means determining what's between a pool of requirements to be applied first and the order of implementation. Moreover, we need to develop software based on prioritized requirements to have a lower probability of being rejected. Stakeholders need to compare prioritized requirements so that they can decide on their vitality through await scale which is finally used to compute the ranks [3]. The acceptability level of software systems is mostly determined by how well the developed system has met or satisfied the specified requirements. The critical success elements for attaining outstanding software systems are eliciting and prioritizing the adequate requirements and scheduling appropriate releases with correct functions. Otherwise, when indistinct or imprecise necessities are carried out, the ensuing system will fall short of person or stakeholder's expectations. Many software improvement initiatives have large potential requirements that may be almost not possible to deliver inside the expected time frame and finances [4].

Requirement prioritization approaches are; first, we define which requirements in the project and which of them are delay or are ignored. Second, the elements that are obtained to be part of the plan have to include their other dependent requirements. Third, the captured requirements have to be prioritized according to their importance.

After determining requirements, we need to repeat this cycle entirely or partially depending on new requirements or changes in the old ones. So, requirements prioritization need to be continuous. Continuous prioritization of requirements into agile methodologies has a significant impact on value delivery of software. Managing continuous changing through requirement prioritization is necessary. Therefore, we will propose a framework for ongoing and scalable requirement prioritization in agile software development.

This paper will be organized as follows: - The first section is an introduction, and the second section is a review of related works that contain most common and modern techniques and its challenges. The third section is the proposed framework for continuous and scalable requirements prioritization. The fourth section contains architecture of a supporting tool to help for using this framework, and the fifth section includes a conclusion and future work.

2. Related work

We have reviewed most common and modern techniques in this area of research that exists in the literature [5]. Most existing techniques of requirement prioritization lack scalability, dependability, continuous prioritization, rank update, feedback handling and full implementation for methods or algorithms. In agile development, we must deal with rapidly changing requirements, so we need a requirement prioritization process to be continuous. Most existing techniques do not take feedback from finished iteration to aid for enhancing and avoiding weakness point in next iteration. It's far worth to be aware that, a number of the present techniques are updated rank manually. For example, methods include round the group prioritization, cumulative voting, ping pong balls approach, win-win, multi-voting system, dot voting system and top ten prioritization techniques. These strategies function efficaciously for restricted scale tasks with the best ten to thirty requirements throughout five to six stakeholders [6]. Many requirements prioritization techniques utilize a ranking process to prioritize candidate requirements. The ranking process is usually executed by assigning weights across requirement based on pre-defined criteria, such as the value of the condition perceived by relevant stakeholders or the cost of implementing each requirement [7]. From the literature view; AHP suffers scalability problem as the most authors concluded. This is because AHP executes rating by way of introducing the method that is described through an improvement of the relative priorities among every two of requirements. This will no longer be possible as the wide variety of requirements grows. It also does no longer assist the evolution of the requirements or rank updates but provides efficient and reliable results [8,9]. Also, from this research, it is discovered that maximum current machine learning techniques be afflicted by rank updates hassle. Most of these techniques that be afflicted by this lower are case base ranking [10]; interactive genetic algorithm prioritization method [4]; Binary

search tree [8] and EVOLVE [11]. Besides, the current methods don't have prioritized classifying requirements according to their respective ranks [8]. Released a few studies where specific prioritization strategies have been experimentally evaluated. From their research have a look at, they stated that the maximum of the prioritization strategies aside from AHP and bubble sorts produce unreliable or deceptive outcomes even as AHP and bubble sorts had been additional time losing.

Then the researchers placed that; techniques like binary search tree, spanning tree, hierarchy AHP, priority groups provide unsure results and those techniques are challenging to achieve. [9]. Were also of the opinion that, some methods like value intelligent prioritization requirement triage and fuzzy logic based techniques are also a waste of time and are additionally error-prone due to their dependence on professionals. Moreover, other methods as Planning game suffers from rank updates problem but has a better variance of numerical computation. Wieger's technique and requirement triage are relatively respected and adopted by practitioners; however, in the event of requirements evolution, those techniques do not support rank updates as well.

In brief, the restrictions of current prioritization techniques can be explained as follows:

1. Loss of scalability: scalability problems afflict methods like pairwise comparisons, bubble sort, and AHP because requirements are as compared based on possible pairs causing n (n _ 1)/2 comparisons [8]. So once the wide variety of elements in a list is doubled, the other method best wants double time and effort for prioritization at the same time the methods AHP, bubble sort and pairwise will require four times. This leads to lack of scalability and consuming time.

2. Dependency requirements: It is a vital attribute that means some requirements depend on one another to work. These requirements which are mutually dependent can finally be combined as one requirement since if one is not there the other will not be implemented. In this way, prioritizing will lead us to redundant and erroneous results. Addressing that attribute among the authors of research prioritizations is scarce. However, it is stated by the author of work [12], that these dependencies can be determined through mapping post and preconditions from all the requirements, but this depends on the contents of every element. Therefore, proper technique of prioritization should consider or consider factors dependences before initiating the process.

3. Continuous requirements change: Continuous requirements prioritization in agile software development is mandatory because the requirements rapidly vary so we must deal with these changes continuously.

4. Rank updates: rank update defined as anytime prioritization; that is, the ability of a way to routinely update ranks each time a requirement is protected or excluded from the listing [10].Requirements evolution is needed to do this situation. So, the current techniques of prioritization can't update or reflect rank statues when a requirement is presented or deleted from the list. Consequently, it does not support iterative updates. A proper and trusty prioritization technique can be through rank updates. Such a limitation appears to surpass most of the current method [13].

5. Feedback consideration: Feedback from the previous iteration is very important to avoid any error or issue in process or technique used in prioritization. So, we need to take into consideration this feedback to improve the process or procedure.

6. Lack of fully implemented requirements prioritization systems: Out of this research it was apparent that most current prioritization techniques haven't been applied in reality as most of those prioritizations are complicated and they need huge time to be done. Prioritized requirements need to be implemented as it will enhance and support requirements prioritization on commercial enterprise and business point [14,15]. Before the efficient work of those algorithms, there should be exact capturing requirements clearly because the output of prioritization processes rely on the input while the target is to visualize software releases. Complete implementation will cause having a steady improvement of the software merchandise that are related with prioritized requirements.

3. The proposed framework

There are many existing techniques for requirement prioritization, but most of these techniques are designed to solve a specific issue. In the literature and practical requirement prioritization, we don't have any technique resolve all issues which are presented previously at once and don't have any integration between two or more techniques to solve these issues. So, we don't need to innovate new techniques or algorithms to resolve these issues. To address these issues, we need to organize and integrate between existing techniques into a new model or framework for strengthening weaknesses for each other to maximize benefit and achieve the ease of use.
This research proposed a generic framework to involve the requirement prioritization technique into complete model of prioritization and make it able to practice, easy to use and handle previous issues lack scalability, dependency requirements, continuous requirements change, feedback handling and lack of fully implemented requirements prioritization systems.
This framework has four stages: (see Figure 1).

Figure 1. The main four stages

Stage one is identification, which will identify stakeholder, Epic, Feature, and user story. Stage two is Verification, which will verify user story. Stage three is the estimation, which will size, effort and value of the user story. Stage four is prioritization, which will prioritize user stories and create product backlog item and update. Also, in this stage will create sprint backlog item and update it.

3.1. Identification

The first stage is identification. (see Figure 2).

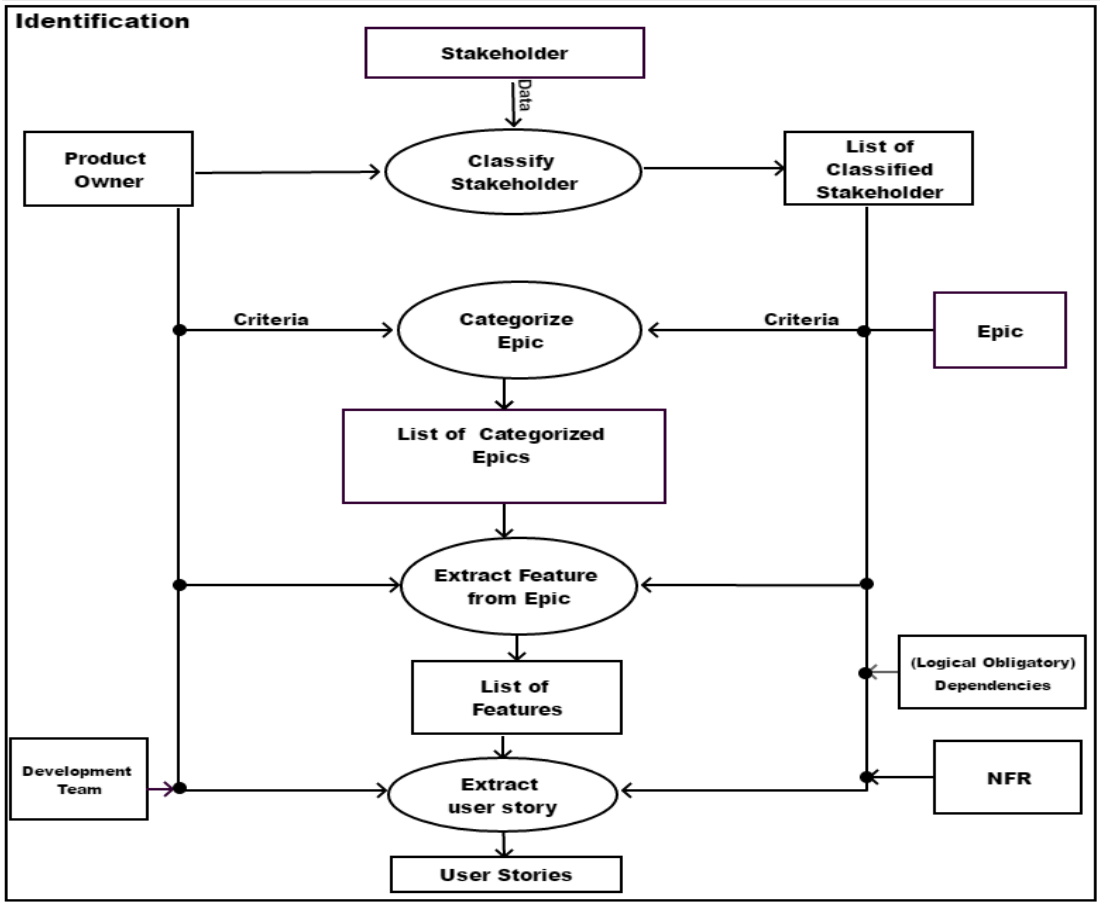

Figure 2. The Identification stage

This stage (Identification stage), which act as preparing stage before start prioritization process. This stage is an important stage Because it aims to a candidate and set priority for stakeholders to extract a list of prioritized stakeholders and place priority to EPICS to obtain a list of categorized EPICS and list of features, which will be used for extract user stories and use these user stories to next stage. All processes in this step will doing by product, development team and customer.

3.2. Verification

The second stage is verification. (see Figure 3).

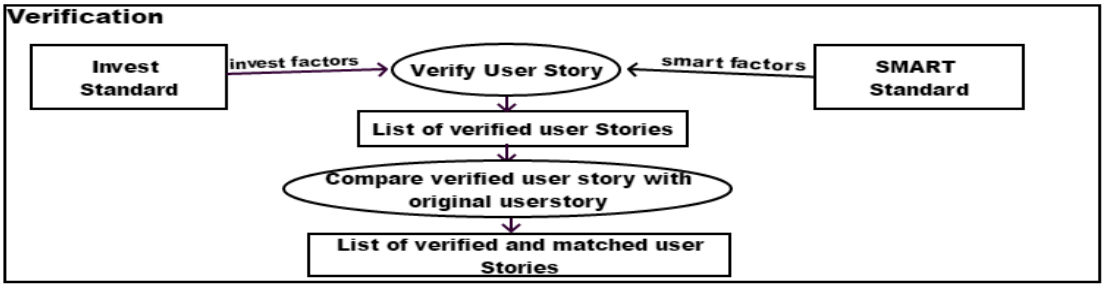

Figure 3. The verification stage

In this stage (verification stage) we will choose two sample techniques from existing techniques, which are INVEST & SMART to verify user stories which have been previously identified. Thus, it will have verified requirements. (Explained later).

3.3. Estimation

The third stage is Estimation. (see Figure 4).

Figure 4. The Estimation stage

In this stage (Estimation stage) we will choose a sample technique from existing techniques, which is Story point. to estimating the size, effort, and value of the requirements, which we verified through the previous stage. The output of this stage will be a list of verified and estimated user stories. (Explained later). As a feature work we can add extra technique for estimation as Work breakdown structure (WBS).

3.4. Prioritization

The fourth stage is creating PBI (product backlog item) and processes. (see Figure 5).

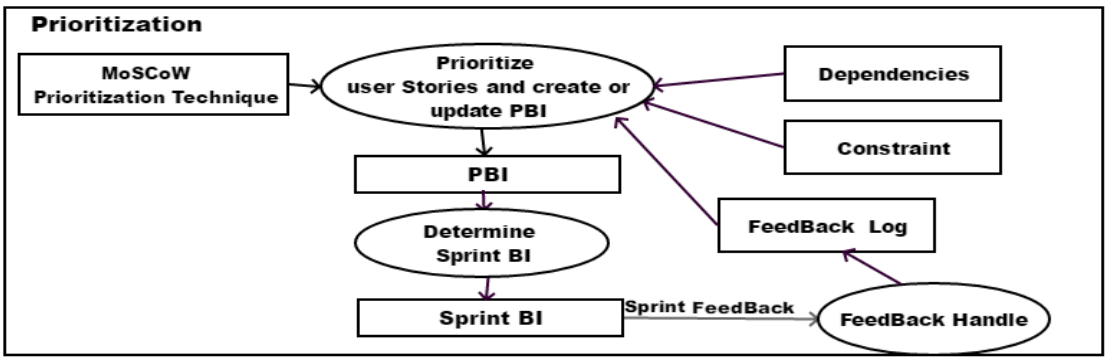

Figure 5. The Prioritization stage

This stage (Prioritization stage) will start with prioritizing requirement. we will choose a sample technique from existing techniques, which is MoSCoW. As a feature work we can add extra technique for prioritization as Kano (customer satisfaction). Now we will use MoSCoW to extract PBI. Thus, retrieve sprint or iteration backlog item from BPI. After that, take feedback from sprint development, which determines whether to record requirements as implemented or reprioritize requirements or update PBI. This is done with putting into consideration the constraints like project constraints, environmental constraints, and other ones. There are many constraints will affect in prioritization, for example: -

Cost: In software development, the main cost is the number of hours is spent in all development cycle. So, this cost could be prioritized to can reduced as can.

Time: Time aspect related to cost. However, time is affected by many other factors including the level of parallelism in development, education desires, need to increase aid infrastructure, full production standards, and many other.

Risk: Every project has some risks. These risks categorized into two groups (internal and external risk). So, this risk could be managed to can calculate risk level by estimate risk impact for the requirement.

Volatility: volatility is a normal characteristic of the requirement, especially in agile development.

There are many reasons for requirement Volatility, for example, business changes, market changes, stakeholders' changes, not an explicit requirement. Volatile requirement affects the cost and planning of a project. So, volatility could be managed to reduce their effect on the project.

Consequently, these four stages lead to ten steps: (see Figure 6). The first stage which is Identification contain four steps as follows: -

3.4.1. Classify stakeholders.

The backbone in agile software is humans or people which work in the project (stakeholders). So, must identify them and there is knowledge need to be shared between them and make all the stakeholders cooperate due to their work commitments and priorities [16]. The product owner will do this classification of stakeholders according to criteria related to business. We need to create backlog stakeholders as product backlog because stakeholders prioritization happens continuously. In the end, the output of this step is a list of classified stakeholders.

3.4.2. Categorize EPIC.

An epic captures a big image of work. It is a big user story that can be divided into the number of smaller stories. This epic needs several sprints to complete. In this step, the list of stakeholders working with the product owner to set their EPICS according to their criteria. The output of this step is a list of categorized EPICS.

3.4.3. Extract features from EPIC.

a feature is the primary unit of software releases. Extracting feature from EPIC will be done on this step by the product owner and stakeholders. The output of this step is a list of features.

3.4.4. Extract user story.

The last step in the identification stage is extract user story from the feature which will be done by the development team and must be customer involved taking into consideration requirement dependencies (logical or hidden dependencies) and NFR (non-functional requirement). The user story is a description consisting of one or extra sentences within the industry language of the end user or a system user that show what a desire to do as a part of his or her job characteristic. Extract user story from feature will happen continuously. Because any change in EPIC will affect in the feature, which will be used for extract user story. Now we can start the next step in the framework using these user's stories.

3.4.5. Verify user story.

This step is in the verification stage aim to review and verify user story to be sure that it is compatible with: - INVEST. it helps to bear in mind the features of an excellent quality user story. which will be used in a Scrum or Kanban backlog and extreme programming. All user story should conform to INVEST concept as follow: - Independent: user story needs to be able to work around, taking into consideration their relative priority without much effort. This is one of the properties of agile methodology such as Kanban, Scrum, and XP. So, in case addressed dependent between two or more user stories, we have to mix them right into a single user story. Negotiable: the user story ought to be self-contained, in a manner that there's no inherent dependency on any other user story. Valuable: end user or customer must get value from user story. Estimable: A size of user story must be able to estimate. Small: User stories should be short to become possible to plan, task and prioritize with a specific level of certainty. Testable: to make test development possible the user story must provide the necessary information.

SMART. All requirements or user story should conform to the SMART concept [17] as follow: -

Specific – all requirements should be clear and accurate. Measurable – all requirements should be measurable so we know when we have reached them. Agreed – The technical team and the business agree on the project deliverables. Realistic – Projects must be possible based on current technologies,

current capabilities within the organization, etc. Time-bounded – Projects must have a definite start and stop date. At the end of this step, the output is a list of verified user stories.

3.4.6. Compare verified user story with original user story

If we don't validate user story to be sure that it is matching with original user story may be losing time to modify after the requirement has been started. So, we need to be sure the list of verified user stories is matched with the original user story. So, this comparison will be done by compare verified user stories with their parent feature through product owner and client as (expertise opinion).

3.4.7. Estimation.

This step is to estimate size, effort, and value of user stories, which will be done by the development team using a sample of existing techniques as Story point (SP) technique. A story point is a handy and green measurement technique for estimating the quantity of effort a group needs to develop a specific characteristic. This is used to size the effort required to implement a story. As a future work we can implement another technique as Work breakdown structure (WBS). Work breakdown structure is an incremental and hierarchical deliverable part of the project as a tree structure, which suggests a segmentation of effort required to carry out an objective. At the end of this step the output is a list of verified and estimated user stories.

3.4.8. Prioritize user stories and create or update product backlog item (PBI).:

The last stage in the framework is the prioritization stage. This is the critical stage, which will contain three steps. The first step is prioritizing user story and create product backlog item (PBI). In this step will use a sample of existing techniques. Which is MoSCoW. The MoSCoW method presents a framework for prioritizing for using in the management. MoSCoW is a reasonably simple way to sort user stories into priority order. It's a way to help teams quickly understand the customer view of what is essential for launch and what is not. MoSCoW stands for: M - MUST have this. S - SHOULD have this if possible. C-COULD have this if this does not affect other. W - WON'T have this time but would like in the future [18].

As a future work we can add extra technique as Kano. The Kano model improves appropriate prioritization and boosts consumer satisfaction and is a theory of product development. It is developed in the 1980s by Professor Noriaki Kano, and This model classifies customer preferences into five categories. Attractive (Excitement), One-Dimensional (Performance), Must- Be (basic), Indifferent, Reverse. At the end of this step, the output is product backlog item (PBI). A product backlog is a list prioritized of tasks that are obtained from the roadmap and its elements for assigned to the development team. The (PBI) arranged as a stack. So, the development team works firstly from the top of (PBI) which contain the most valuable items, so we know what to deliver first. There are many factor or constraint effect in requirement prioritization process. So, we need to identify these factors to help teamwork in requirement prioritization to determining which requirement will be implemented first [19]. These constraints are categorized into three groups. The first group is Project constraint as budget, time and resources. The second group is environment constraints as infrastructure, operating system, and user experience. The third group is an unexpected change in requirement or the project overall. And we must take into consideration requirement dependences, which will be affect also in prioritization process. The second step in the prioritization stage is Determine sprint backlog, which will pull some prioritized user story from (PBI) related to sprint time. The third step in the prioritization stage is Feedback handle. In this step, the framework takes into consideration the feedback from previous sprint or iteration to avoid any problem in next sprint.

3.4.9. Determine sprint backlog item (SBI).

This step is the second step in the prioritization stage. The product backlog item (PBI), which create in previous step act as a repository of all requirements are ready for developed. These requirements will have developed in continuous sprints. This step will determine and pull from (PBI) stack regarding

sprint capacity. At the end of this step, the output is the sprint backlog item (SBI). This step will be doing till finish all requirements at product backlog item (PBI).

3.4.10. Feedback handle.

This step is the third step in the prioritization stage. It aims to take feedback from the development team about the current Sprint to use this feedback for enhancement next sprint, and this feedback also aims to enhance and refinement product backlog item (PBI). So, this step is an analytic and auditing step, which will reflect on the prioritization stage at all. As a feature work we can use this historical data for enhancement prioritization process.

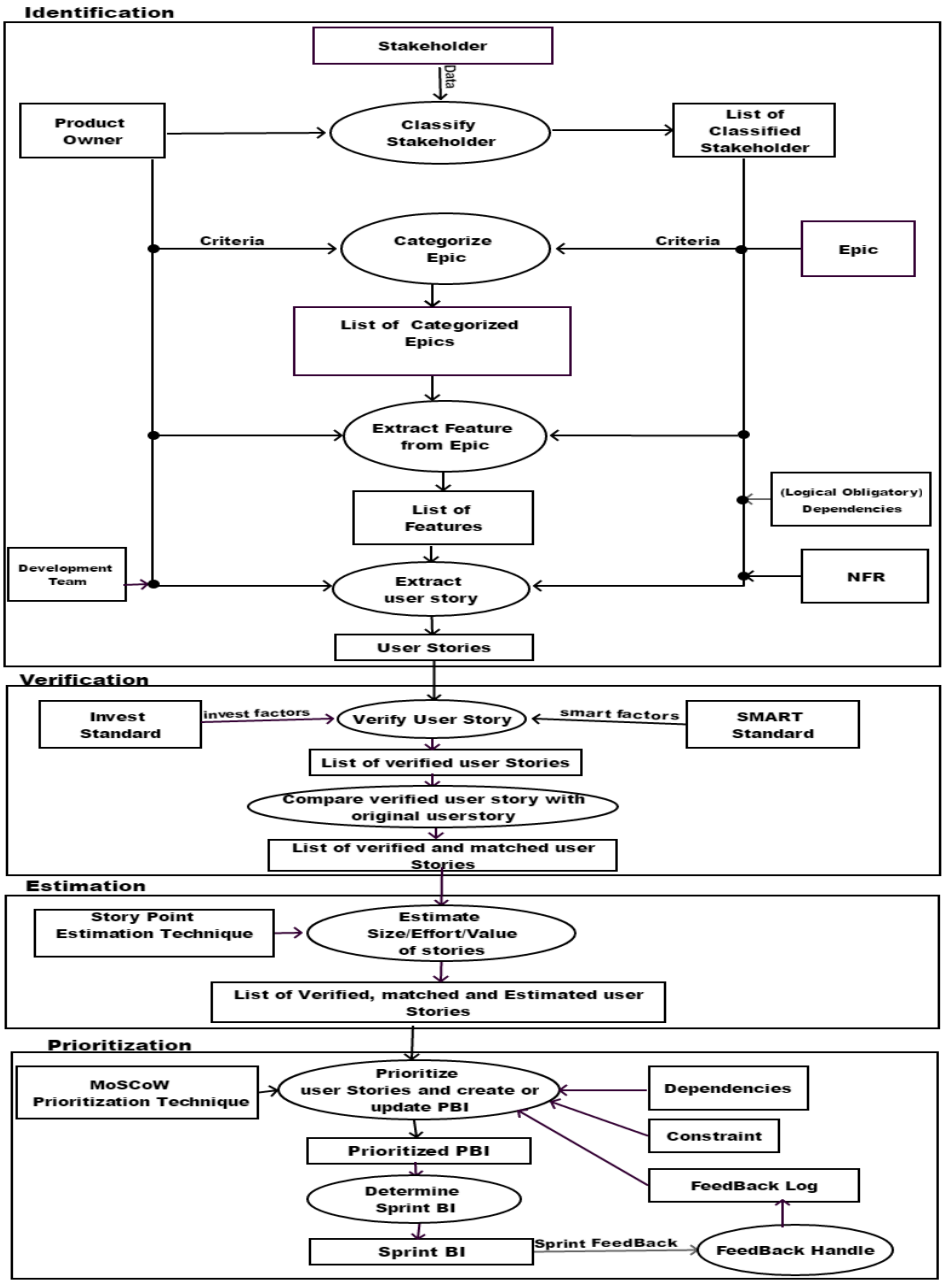

Figure 6. The Conceptual framework for continuous and scalability agile requirements prioritization process

4. Conclusion and future work

Most of the existing prioritization techniques need to be more continuous, scalable, simply implemented and integrated with software development life cycle and not work separately. We try to make a fully integrated framework to be ready for agile development processes which start early from epic and user story. Requirements almost have changed many times throw development cycle. This framework makes agile development easy to handle any changes in requirement at any stage of the development cycle. As a future work, we are developing a supporting tool, which implements the proposed framework and start using this tool for real open source ERP (Enterprise Resource Planning) application development. This tool must be easy to use and exploit its database for analysis projects behavior to improve and enhance this framework.

References

[1] A. Hudaib, R. Masadeh, M. H. Qasem, and A. Alzaqebah, "Requirements Prioritization Techniques Comparison," *Mod. Appl. Sci.*, vol. 12, no. 2, p. 62, 2018.
[2] C. Duan, P. Laurent, J. Cleland-Huang, and C. Kwiatkowski, "Towards automated requirements prioritization and triage," *Requir. Eng.*, vol. 14, no. 2, pp. 73–89, 2009.
[3] A. Kobayashi and M. Maekawa, "Need-based requirements change management." pp. 171–178, 2002.
[4] P. Tonella, A. Susi, and F. Palma, "Interactive requirements prioritization using a genetic algorithm," *Inf. Softw. Technol.*, vol. 55, no. 1, pp. 173–187, 2013.
[5] P. Achimugu, A. Selamat, R. Ibrahim, and M. N. R. Mahrin, "A systematic literature review of software requirements prioritization research," *Inf. Softw. Technol.*, vol. 56, no. 6, pp. 568–585, 2014.
[6] T. M. Fehlmann, "New lanchester theory for requirements prioritization," *2008 2nd Int. Work. Softw. Prod. Manag. ISWPM'08*, 2008.
[7] A. T. Azar, P. Achimugu, and A. Selamat, "Computational Intelligence Applications in Modeling and Control," vol. 575, 2015.
[8] J. Karlsson, C. Wohlin, and B. Regnell, "An evaluation of methods for prioritizing software requirements," *Inf. Softw. Technol.*, vol. 39, no. 14–15, pp. 939–947, 1998.
[9] M. Ramzan, M. I. Babar, M. Rarnzan, and S. A. K. Gha, "Challenges and future trends in software requirements prioritization Challenges and Future Trends in Software Requirements Prioritization," no. May 2014, 2014.
[10] A. Perini, A. Susi, and P. Avesani, "A machine learning approach to software requirements prioritization," *IEEE Trans. Softw. Eng.*, vol. 39, no. 4, pp. 445–461, 2013.
[11] D. Greer and G. Ruhe, "Software release planning: An evolutionary and iterative approach," *Inf. Softw. Technol.*, vol. 46, no. 4, pp. 243–253, 2004.
[12] F. Moisiadis, "Prioritising scenario evolution," pp. 85–94, 2002.
[13] A. Auburn, "Sample selection: an algorithm for requirements prioritization," *ACM*, 2008.
[14] M. Ramzan, M. A. Jaffar, M. A. Iqbal, S. Anwar, and A. A. Shahid, "Value based fuzzy requirement prioritization and its evaluation framework," *2009 4th Int. Conf. Innov. Comput. Inf. Control. ICICIC 2009*, pp. 1464–1468, 2009.
[15] S. Barney, A. Aurum, and C. Wohlin, "Quest for a silver bullet: Creating software product value through requirements selection," *Proc. - 32nd Euromicro Conf. Softw. Eng. Adv. Appl. SEAA*, pp. 274–281, 2006.
[16] Z. Mansor, R. Razali, J. Yahaya, S. Yahya, and N. H. Arshad, "Issues and challenges of cost management in agile software development projects," *Adv. Sci. Lett.*, vol. 22, no. 8, pp. 1981–1984, 2016.
[17] M. Mannion and B. Keepence, "SMART requirements," *ACM SIGSOFT Softw. Eng. Notes*, vol. 20, no. 2, pp. 42–47, 2004.
[18] Z. Racheva, M. Daneva, A. Herrmann, and R. J. Wieringa, "A conceptual model and process for client-driven agile requirements prioritization," in *2010 4th International Conference on*

Research Challenges in Information Science - Proceedings, RCIS 2010, 2010.
[19] R. H. Al-Ta'ani and R. Razali, "A Framework for Requirements Prioritisation Process in an Agile Software Development Environment: Empirical Study," *Int. J. Adv. Sci. Eng. Inf. Technol.*, vol. 6, no. 6, p. 846, 2016.