2020

A Supporting Tool for Requirements Prioritization Process in Agile Software Development

ramdan mowad

Faculty of Computers and Information Technology, Future University in Egypt, Cairo, Egypt,
ramdan.mowad@fue.edu.eg

Khaled AbdElazim Muhammad

College of Computing and Information Technology, Arab Academy for Science, Technology and Maritime Transport University (AASTMT), Cairo, Egypt, resalasoft@gmail.com

Essam Elfakharany

College of Computing and Information Technology, Arab Academy for Science, Technology and Maritime Transport University (AASTMT), essam.fakharany@gmail.com

Follow this and additional works at: <https://digitalcommons.aaru.edu.jo/fcij>

Part of the [Computer and Systems Architecture Commons](#), [Data Storage Systems Commons](#), and the [Digital Communications and Networking Commons](#)

Recommended Citation

mowad, ramdan; Muhammad, Khaled AbdElazim; and Elfakharany, Essam (2020) "A Supporting Tool for Requirements Prioritization Process in Agile Software Development," *Future Computing and Informatics Journal*: Vol. 5: Iss. 1, Article 2.

DOI: <http://doi.org/10.54623/fue.fcij.5.1.2>

Available at: <https://digitalcommons.aaru.edu.jo/fcij/vol5/iss1/2>

This Article is brought to you for free and open access by Arab Journals Platform. It has been accepted for inclusion in Future Computing and Informatics Journal by an authorized editor. The journal is hosted on [Digital Commons](#), an Elsevier platform. For more information, please contact rakan@aarj.edu.jo, marah@aarj.edu.jo, u.murad@aarj.edu.jo.

Future Computing and Informatics Journal

Volume 5
Issue 1 (2020) *Volume 5 Issue 1*

Article 2

2020

A Supporting Tool for Requirements Prioritization Process in Agile Software Development

ramdan mowad

Faculty of Computers and Information Technology, Future University in Egypt, Cairo, Egypt,
ramdan.mowad@fue.edu.eg

Khaled AbdElazim Muhammad

College of Computing and Information Technology, Arab Academy for Science, Technology and Maritime Transport University (AASTMT), Cairo, Egypt, resalasoft@gmail.com

Essam Elfakharany

College of Computing and Information Technology, Arab Academy for Science, Technology and Maritime Transport University (AASTMT), essam.fakharany@gmail.com

Follow this and additional works at: <https://digitalcommons.aaru.edu.jo/fcij>

Part of the [Computer and Systems Architecture Commons](#), [Data Storage Systems Commons](#), and the [Digital Communications and Networking Commons](#)

Recommended Citation

mowad, ramdan; Muhammad, Khaled AbdElazim; and Elfakharany, Essam (2020) "A Supporting Tool for Requirements Prioritization Process in Agile Software Development," *Future Computing and Informatics Journal*: Vol. 5 : Iss. 1 , Article 2.

DOI: <http://doi.org/10.54623/fue.fcij.5.1.2>

Available at: <https://digitalcommons.aaru.edu.jo/fcij/vol5/iss1/2>

This Article is brought to you for free and open access by Arab Journals Platform. It has been accepted for inclusion in Future Computing and Informatics Journal by an authorized editor. The journal is hosted on [Digital Commons](#), an Elsevier platform. For more information, please contact rakan@aarj.edu.jo, marah@aarj.edu.jo, u.murad@aarj.edu.jo.

A Supporting Tool for Requirements Prioritization Process in Agile Software Development

Khaled Abdelazim Muhammad^{1a}, Ramadan Moawad^{2b} Essam Elfakharany^{3c}

¹College of Computing and Information Technology, Arab Academy for Science, Technology and Maritime Transport University (AASTMT), Cairo, Egypt

²Faculty of Computers and Information Technology, Future University in Egypt, Cairo, Egypt

³College of Computing and Information Technology, Arab Academy for Science, Technology and Maritime Transport University (AASTMT)

^a Khaled @ resalasoft.com, ^b ramdan.mowad@fue.edu.eg , ^c essam.elfakharany@aast.edu

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, change of requirements, and requirements dependencies. The prioritization techniques need to be more continuous, scalable, implemented easily and integrated with software development life cycle. This paper introduces a supporting tool for a proposed 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. The proposed framework is validated in a real case study using its supporting tool, and the results are promising.

Keywords: Requirements engineering, requirements prioritization, continuous requirements prioritization, Scalable requirements prioritization, requirements prioritization framework, supporting tool.

1.Introduction

From the literature, there are many existing techniques for requirement prioritization (RP), but most existing techniques of RP lack scalability, dependability, continuous prioritization, rank updating, implementation tool, ease of use and full implementation for methods or algorithms in the agile software development environment [1]. So, most of these prioritization techniques need to be more continuous, scalable, implemented simply and integrated with the software development life cycle and do not work separately [2]. Our motivation for proposing a fully integrated framework and supporting tool arises from the limitations of current methods as mentioned above.

This research introduces a supporting tool for the proposed framework in an agile software development environment to involve RP into a complete model of prioritization and make prioritization process easy to use and try to handle previous issues as rank updating, computational complexity, implementation tool, Implementation and testing in Agile environment and scalability [3]. The proposed RP framework use MoSCoW as one of the common techniques in RP[4]. 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 [7]. The framework is tested in a case study performed in the business environment on real Enterprise Resource Planning (ERP) system. This case study is used for demonstration only, but the framework can be applied in any real software system This framework has four stages [10]: (see Figure 1).

Figure 1. The main four stages for the proposed framework

Table 1 :shows the input, output and suggested steps for prioritizing requirements.

Input	list of stakeholders, EPICS and NFR.
Output	Recommended Prioritized list of requirements(PBI).
Process	Begin Stage 1: Identification Step 1: Classify stakeholders Step 2: Sort EPICs Step 3: Extract features from EPICs Step 4: Extract user story Stage 2: Verification Step 5: Verify user story Step 6: Compare verified user story with original user story Stage 3: Estimation Step 7: Estimation Stage 4: Prioritization Step 8: Prioritize user stories and create or update product backlog item (PBI) Step 9: Determine sprint backlog item (SBI) Step 10: Feedback handle End

2. Supporting Tool

The tool was implemented in Odoo version 12.0 CE, which is one of the common open-source ERP, using python 3 and PostgreSQL 10 database. As seen in (Figure 2: Use Case for Support Tool) the tool has product owner, classified stakeholders and development team as actors, and it has ten activities. It takes EPICs, stakeholders, and NFR as inputs and process them to generate, after ten processes, the Recommended Prioritized list of requirements (PBI).

Figure 2: Use Case for Support Tool.

The requirements automatically appear against their names to stakeholders to set its rank. The relative weights are then processed or computed to display the final ranks of requirements. The tool aids the re-weighting and re-computation of ranks whenever requirements evolve. We can include or exclude stakeholders and requirements if needed. However, deleting or adding a stakeholder or requirement, the tool will recalculate the rank automatically. The tool automatically re-compute the new ranks of each requirement based on the new weight of stakeholders.

Figure 3 shows a support tool main screen that was implemented in the Odoo community. The tool takes relative weights as input and processes stakeholder criteria to produce a list of priorities. It provides an easy way to switch from the web pages to various interface menus. The tool also provides a way for stakeholders to be included or excluded. The main three steps of the tool are: 1- the administrator will add new stakeholders. As many stakeholders as a software project needs should be handled by the suggested method. 2- The requirements for consensus tend to trigger the scoring process automatically against their titles. 3- The relative weights are then processed or calculated to indicate the final range of requirements.

Figure 3: Tool Main Screen

3. Case study

The proposed framework was implemented on real Enterprise Resource Planning (ERP) system. The Human Resource Management System (HRMS) was selected as its critical system that manages employee data such as a part of ERP system.

Each module performs a separate function within the HRMS that helps with information gathering or tracking. HRMS modules can assist with:

- 1- Managing payroll
- 2- Gathering, storing, and accessing employee information
- 3- Keeping attendance records and tracking absenteeism
- 4- Manage time schemas
- 5- Fingerprint integration
- 6- Performance evaluation

7- Recruitment

8- Learning management

9- Employee self-service

10- Employee scheduling

The proposed framework is evaluated using a really small and medium-sized enterprises (SMEs) software project. The proposed framework is compared to the existing technique used in the requirement prioritization (MoSCoW). We implemented our case study using the

MoSCoW technique and the proposed Framework. Our case study contains two Epics, five stakeholders, five development team (working 8 hours per day and 5 days per week). Sprint time is two weeks (from 30 to 35 story points). Then we will use the man hour (MH) for calculating the sprint cost. As a condition each of MoSCoW and the proposed framework must deliver the right system with required features and functions.

3.1 Application of the proposed framework on the real case study using the supporting tool.

This section provides parameters, indicators and algorithmic steps of the computational process of prioritizing software requirements with the proposed framework. The proposed Framework employs Multi-Criteria Decision Analysis (MCDA) to calculate ranks of each requirement.

3.1.1 Stage 1: Identification

Classify Stakeholders related to Weight Scale (WS). This step is done by the product owner (PO) based on stakeholder's classification matrix (see figure 4). The stakeholder's weight will be used in requirement rank calculation (see table 2).

Figure 4: A stakeholder's matrix

Table 2: Stakeholders Weight Scale (WS)

Terms	Numeric rating
Extremely high (EH)	5
Very high (VH)	4
High (H)	3
Fair (F)	2
Low (L)	1

Step 1: Classify stakeholders

Explanatory example has five stakeholders and product owner set their weights as follows (see Table 3).

Table 3: Preference weights of Stakeholders

Stakeholders	Weight
Stakeholder 1	5 (EH)
Stakeholder 2	2 (F)
Stakeholder 3	4 (VH)
Stakeholder 4	1 (L)
Stakeholder 5	3 (H)

Step 2: Sort EPIC

Our explanatory example has two EPICs:

- 1-Time management (attendance schemas)
- 2-Payroll accounting

The Product owner and the classified stakeholders will manage the non-sorted EPICs to formulate the list of sorted EPICs regarding their criteria.

Step 3: Extract features from EPICs

Product owner and the classified stakeholders will extract feature from epics. After extraction process we have five features:

- 1- Salary rules
- 2- Leaves management
- 3- Finger print attendance management
- 4- Salary structure
- 5- Employee pay slip

Step 4: Extract user stories from Features

Classified stakeholders and development team will breakdown the features into user stories keeping in mind the NFRs and logical obligatory dependencies (LOD). The output for this process is sixteen user stories plus one NFR plus two LOD. So finally, we have nineteen user stories.

LOD and NFR are:

LOD

1- Get attendance record from finger print depends

on →Integrate with finger print machine

2- Print employee pay slip any time depends on

→Salary calculate batch

NFR

when employee login to make leave request must hide record for other employees (security).

3.1.2 Stage 2: Verification

The development team and classified stakeholders will verify user stories related to INVEST concept, which mean the good user story could be Independent, Negotiable, Estimable, Small, Testable. This stage designed in the proposed framework as a dynamic stage. It can be configured as stakeholders need. For example, we can neglect user story if it does not match INVEST full or partial matching or identify it again [5].

Step 5: Verify user story

After verification stage we have one user story does not conform the Valuable as INVEST concept, so it will be ignored. And another two user stories don't conform the Small as INVEST concept, so they will be broken down to four user stories instead of two. At the end of this process we have twenty user stories.

Step 6: Compare verified user stories with original user stories

The development team and classified stakeholders compare the verified user stories with the base user stories to validate the requirement to check if it is still met and there is no deviation occurred in the previous steps based on the expertise opinion. All user stories

pass this step, so we have to continue with the same twenty user stories.

3.1.3 Stage 3: Estimation Step 7: Estimation

The Development team start to estimate the effort for each user story by using story point estimation technique [6]. According to most scale sizing using in agile development Fibonacci sequence: 1,2,3,5,8,13,21. This stage design in the proposed framework is a dynamic stage. It can be configured as stakeholders need. For example, one can neglect user story if it has estimation effort more than any percentage or identify it again. After estimation stage we have twenty user stories estimated (see Table 4).

Table 4: Requirement estimation

User Stories	Story Point
US 1	5
US 2	3
US 3	2
US 4	8
US 5	3
US 6	1
US 7	2
US 8	3
US 9	5
US 10	21
US 11	13
US 12	1
US 13	5
US 14	13
US 15	8
US 16	8
US 17	13
US 18	21
US 19	1
US 20	21

3.1.1 Stage 4: Prioritization

Set rank for requirements by PO and stakeholders based on the adaptation of MoSCoW method as a Weight Scale (see Table5).

Table :5

Terms	Numeric rating
Must (M)	4
Should (S)	3
Could (C)	2
Won't (W)	1

Set rank for Requirements R1, R2, R3, ..., Rn (i.e. n – Requirements) by PO and Stakeholders S1, S2, S3, ..., Su (i.e. u – number of Stakeholders). The weights in Table 6 is an example for one stakeholder across 4 requirements.

Table 6: Sample Preference weights of requirements

Requirements	MoSCoW	MoSCoW weight Scale
R1	S	3
R2	C	2
R3	M	4
R4	W	1

As shown in (Table 7) The proposed framework will calculate a decision matrix for each stakeholder of all the requirements by applying next equation:

$$rank_j = rank_{j.si}$$

where n no of requirements and u No of Stakeholder

and by using the next loop:

While ($i \leq u$) do

While ($j \leq n$) do

Table 7: Rank requirements Based on MoSCoW

Stories	Stakeholders					Rank Sum
	S1	S2	S3	S4	S5	
US 1	4	3	3	4	3	17
US 2	3	2	3	4	3	15
US 3	4	4	2	3	2	15
US 4	1	2	4	2	3	12
US 5	3	1	1	3	4	12
US 6	1	3	3	1	2	10
US 7	4	3	4	4	3	18
US 8	4	4	4	4	4	20
US 9	3	4	4	4	4	19
US 10	3	3	3	3	3	15
US 11	3	2	2	3	1	11
US 12	3	3	3	3	3	15
US 13	3	1	3	2	1	10
US 14	2	1	2	1	2	8
US 15	2	2	2	2	2	10
US 16	2	2	2	2	3	11
US 17	4	3	3	4	4	18
US 18	2	2	3	2	2	11
US 19	4	3	3	4	3	17
US 20	2	1	2	1	3	9

As shown in (Table 8) the tool calculates ranks of each requirement using Weighted Average Decision Matrix (WADM) across the project stakeholders related to stakeholder weight using next equation:

$$RankSum_j = \sum_{i=1}^u rank_{j.swi}$$

where Swi = stakeholder weight for each requirement and u is No of Stakeholder

Table 8: Rank requirements with proposed framework

User Stories	Stakeholders						Rank Sum
	weight	S1	S2	S3	S4	S5	
		5	2	4	1	3	
	Rank						
US 1	20	6	12	4	9		51
US 2	15	4	12	4	9		44
US 3	20	8	8	3	6		45
US 4	5	4	16	2	9		36
US 5	15	2	4	3	12		36
US 6	5	6	12	1	6		30
US 7	20	6	16	4	9		55
US 8	20	8	16	4	12		60
US 9	15	8	16	4	12		55
US 10	15	6	12	3	9		45
US 11	15	4	8	3	3		33
US 12	15	6	12	3	9		45
US 13	15	2	12	2	3		34
US 14	10	2	8	1	6		27
US 15	10	4	8	2	6		30
US 16	10	4	8	2	9		33
US 17	20	6	12	4	12		54
US 18	10	4	12	2	6		34
US 19	20	6	12	4	9		51
US 20	10	2	8	1	9		30

Step 8: Prioritize user stories and create or update product backlog item (PBI)

This obtains the performance scores for the requirements by summing the rank (R_j) of each requirement across the stakeholders and sorting result as a stack in PBI using next equation, (see Table 9).

$$P_i = \sum_{j=1}^n R_j$$

Table : 9

#	Prioritized Requirements (PBI)
1	US 8
2	US 7
3	US 9
4	US 17
5	US 1
6	US 19
7	US 3
8	US 10
9	US 12
10	US 2
11	US 4
12	US 5
13	US 13
14	US 18
15	US 11
16	US 16
17	US 6
18	US 15
19	US 20
20	US 14

Step 9: Determine sprint backlog item (SBI)

PO and classified stakeholders should pull the items from the PBI to fit inside the sprints as SBI taking into consideration spring capacity (see Table 10).

Table 10: sprint backlog item (SBI)

#	FW	SP Effort	Sprint	Sprint Cost
1	US 8	3	Sprint 1	31
2	US 7	2		
3	US 9	5		
4	US 17	13		
5	US 1	5		
6	US 19	1		
7	US 3	2		
8	US 10	21	Sprint 2	33
9	US 12	1		
10	US 2	3		
11	US 4	8		
12	US 5	3	Sprint 3	29
13	US 13	5		
14	US 18	21		
15	US 11	13	Sprint 4	30
16	US 16	8		
17	US 6	1		
18	US 15	8		
19	US 20	21	Sprint 5	34
20	US 14	13		

Step 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 of the next sprint. Now we store this feedback into feedback log to use it as future work for enhancing the proposed framework.

4. Results

We implemented the proposed requirement prioritization framework in a real business environment case study. And the same case study is implemented also using traditional agile by using MoSCoW method.

4.1 MoSCoW implementation result:

With MoSCoW we start sprints with 16 user stories but the final executed user stories are 20. This gap affects sprint productivity because we have non-productive cost 2 sprints (400 MH). This affect the time to market, increase cost and maybe pay the penalty to the client (see Table11, Figure 5 and Figure 6).

4.2 Framework implementation:

With the proposed framework we start sprints with 20 user stories. But how this happens: the classified stakeholders and development team extract 16 user stories from the features plus 1 NFR plus 1 logical obligatory dependency plus 1 US dependent on another one, so now we have 19 user stories. Then After verification stage we have 1 user story does not conform the Valuable as INVEST concept, so it will be ignored. And another 2 user stories don't conform the Small as INVEST concept, so they will be broken down to 4 user stories instead of two. So finally, we have 20 user stories. This work done on sprint Zero.

The actual number of sprints is 5 sprints which needed for finishing all user stories. It is the same number of sprints planned based on the proposed framework. So we don't have any extra effort and can deliver the right system with required features and function (see Table12).

The case study results show that the proposed framework has 100% productivity (depending on our case study settings) and no gap between planned sprints and actual sprints. The delivery time had been decreased by 40% which saves time, cost and effort. After the successful tool implementation of the framework, the issues like scalability, rank updates and computational complexity had been solved (see Table 13 and Figure 7).

Table 11: MoSCoW Final Result

Sprint	PBI	planned Sprint BI	added US	removed US	Cause	Actual sprint BI	remaining PBI	sprint cost	productiv e cost	Non- productiv e effort
1	16	4	0	0		4	12	200	200	0
2	12	4	1	0	LOD US.	3	10	200	125	75
3	10	4	2	1	! INVEST (not small)	3	8	200	160	40
4	8	4	0	1	! INVEST (! valuable)	3	4	200	110	90
5	4	4	2	1	! INVEST (not small)	3	2	200	165	35
6	2	4	1	0	Dependent US	3	0	200	130	70
7	0	0	1	0	NFR	1	0	200	110	90

Table 12: Proposed Framework Planned and Actual Sprint

Sprint	PBI	planned BI	sprint cost	added US	removed US	Actual sprint BI	remaining PBI
1	20	4	200	0	0	4	16
2	16	4	200	0	0	4	12
3	12	4	200	0	0	4	8
4	8	4	200	0	0	4	4
5	4	4	200	0	0	4	0

Figure 5. The final result of MoSCoW we have extra 2 sprints (400 MH)

Figure 6: Total MoSCoW Productivity

Table 13: MoSCoW vs Proposed Framework final result

		planned	Actual
MoSCoW	No of Sprints	4	7
	Sprint Cost (MH)	200	
	Total (MH)	800	1400
FW	No of Sprints	5	5
	Sprint Cost (MH)	200	
	Total (MH)	1000	1000

Figure 7: MoSCoW vs Proposed Framework

5. Conclusion and Future Work

This research suggests a system requirements prioritization framework and its supporting tool. Our goal is to create effective support aiding companies in making structured and objective decisions when prioritizing requirements and deliver the right system with required features and function. The mission of the proposed framework is to support companies in their objective prioritization work within system development. Right now, it only outlines the most important prioritization components. Hence, we suggest it be an initial version for both the industry and academia. We also propose to further evolve it.

The competitive advantage of the proposed framework over other methods used in agile development is that the proposed framework has three preparing stages identification, verification, and estimation before start prioritization. These stages aim to prepare requirements before start using any technique in the prioritization process. For that, the proposed framework supports the software companies to develop the right solutions with high quality keeping the time and cost constraint untouched. Moreover, the proposed framework is designed to be natively integrated with the most common existing techniques of verification, estimation, and prioritization, which increases the flexibility and customizability of the framework.

The case study results show that the proposed framework has 100% productivity (depending on our case study settings) and no gap between planned sprints and actual sprints. The delivery time had been decreased by 40% which save time, cost and effort. It also handles previously addressed issues as dependability, NFR and continuous prioritizing.

The developed tool was designed and implemented to cater to large requirements

and stakeholders. It is easy to use with a friendlier user interface, reduced computational complexities and has addressed rank reversals issues. After the successful tool implementation of the framework, the remaining issues like scalability, rank updates and computational complexity had been solved. The developed tool can classify ranked requirements in sequential order with a linked graph to visualize the prioritized results at a glance.

6. References

- [1] 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.
- [2] 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.
- [3] 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.
- [4] S. Hatton, "Early Prioritisation of Goals," *Adv. Concept. Model. – Found. Appl.*, pp. 235–244, 2007.
- [5] L. Buglione and A. Abran, *Improving the User Story Agile Technique Using the INVEST Criteria*. 2013.
- [6] M. Usman, E. Mendes, and J. Börstler, "Effort estimation in agile software development," *Proc. 19th Int. Conf. Eval. Assess. Softw. Eng. - EASE '15*, vol. 3, no. 7, pp. 1–10, 2015.
- [7] 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.
- [8] 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.

[9] 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.

[10] K. Abdelazim, R. Moawad and E. Elfakharany, "A Framework for Requirements Prioritization Process in Agile Software Development," *Journal of Physics: Conference Series*, pp. 1-11, 2020.