USER INVOLVEMENT IN TRANSITION FOR AGILE METHOD UNDER TRADITIONAL WATERFALL MODEL USING A CASE STUDY BASED ON SOFTWARE PROJECT ACTIVITIES

Nalinee Sophatsathit

Computer Science Program, Faculty of Science and Technology, Suan Sunandha Rajabhat University, Bangkok, Thailand
nalinee.so@ssru.ac.th

Abstract

This study examines user involvement in a software project using agile method that is performed under traditional waterfall model. The focus is placed on practicality of user transitional activities from the agile method operating under the umbrella of waterfall model. We explore the level of transitional detailed activity using three experiments, namely, no detail waterfall, coarse-grained waterfall-agile, and fine-grained waterfall-agile. A preliminary study is conducted with the help of local professional developers to set up a process guideline for the project performance measurement. Three measurable activities are recommended, namely, change request, rework, and time buffering. COSMIC Function Point metrics are used to measure the above activities based on user's requests. The results show that user involvement pays off upon user acceptance test since less reworks are required, saving the effort and time buffering. This practicality will contribute to actual production of software project management.

***Keywords*: Agile method; waterfall model; COSMIC Function Point; user acceptance test.**

1. Introduction

Traditional software development process models such as waterfall suffer from heavy load at the beginning and the end of the development life cycle (LC). This is because developers must establish correct and complete software requirements specification (SRS) before project delivery and get user acceptance of the project at the end. In many cases, the effort of rework during user acceptance test (UAT) is considerable or the project is rejected. This is because from the setup of SRS until UAT, the span can be time-consuming. Things might change during this time span, e.g., technology and user change requests that render some specifications of the stated SRS obsolete or reengineering of the workflow. The advent of agile development method has been exercised by many small projects to fill this time-span gap and provides several contributing factors such as easy to adapt practice, build-a-little-test-a-little framework, and user involvement. The questions arise as to how far we should incorporate agile method into the project if we do not want to exercise the practice in its entirety, for example, during coding and testing since users may not have the skills to participate and contribute. Moreover, there is a tradeoff whether it is worth the effort to accommodate or refuse user's change requests due to time constraints, budget, or technical reasons. The determining criteria are essential considerations for every project manager.

Despite myriad studies of requirements change and traceability, process adjustment, and so on that render the study of user's change request and relating activities mundane, many agile practices are still not adopted by local software developers. One must realize that company culture, user knowledge, and disciplinary development practices are difficult to change since these changes are disruptive, costly, and hard to get accustomed. Criticisms about this study in conducting an old and un-innovative topic were carefully considered. Thus, we set out to focus on practical problems based on recommendations from local professional developers, namely, change request, rework, and time buffering, that will establish some practical guidelines to assist local developers. Such problems may be 'obvious and well-known' to some people, but are not obvious and well-known to handle in their work.

This paper looks into the process of keeping SRS up-to-date with change requests as project progresses. By waterfall model, the change would not be recognized until UAT stage which is very difficult to rectify and accommodate. This shortfall does not affect when using the agile method. But what must the projects that follow some course of action in between the two approaches do? This transition gap will be investigated in two ways.

The first way introduces a gradual transition from waterfall to agile by injecting agile activities into waterfall process so as not to create any disruptive changes during the development process. The second way employs agile activities from the beginning to the end of the project. A preliminary survey is conducted with the help of some local professional developers to obtain working guidelines to setup an experiment so as to validate the proposed method. Contributions of this study are two folds: (1) project activities that furnish important statistical inferences of user involvement, and (2) level of activity measurement breakdowns (no detail, coarse-grained, fine-grained) that the project manager should heed.

Organization of this paper is as follows. Section 2 describes a few related works to this study. Section 3 states the proposed models, namely, waterfall, coarse-grained and fine-grained waterfall-agile. Section 4 explains the experiments, starting from adopting developers' experience to set up project scenarios for user's change requests. The projects are then carried out using the three experiments. Section 5 discusses performance statistics being analyzed to reflect if the desired outcomes are attained. Some final thoughts and future work are given in Section 6.

2. Related work

Several literatures were reviewed to sort out the advantages and disadvantages of agile development. The benefits so obtained were adapted in the proposed framework. Abrahamson *et al.* [Abrahamson (2004)] adapted the agile method to mobile development applications called Mobile-D that combined Extreme Programming practice and scalability of Crystal methodologies. The experiences gained were fruitful such as efficient information sharing, increased progress visibility, early identification and solving of technical problems, and high process-practice coherence. Lee *et al.* [Lee (2010)] studied field data on software development agility from business and technology environments using qualitative and quantitative analyses to exemplify their increasing importance in dealing with changing requirements. This inevitably leads to frequent development interruptions. Wiesche [Wiesche (2021)] explored the inherent interruption of agile process and identified three categories of interruptions. At any rate, the team could develop means to prevent members from frequent interruptions that would disrupt the continuity of work, improved information retrieval, and reduced team dependencies. We will incorporate this guidance, along with those of Buglione *et al.* [Buglione (2007)] into the proposed approach.

The relationship among team members constituted healthy team chemistry. The results were on-time and on-budget completing software functionality, extensiveness, and efficiency of team response. Drury-Grogan [Drury-Grogan (2021)] examined cognitive artifact use during agile development process to manage the project. As project progressed, team cognition and cognitive artifact use changed. Understanding this team process behavior such as communication is essential. When team size grows and becomes larger, the lesser is the interactions among team members. Begel *et al.* [Begel (2007)] took a survey on agile software development (ASD) methodologies in industrial context and found Scrum to be the most popular ASD method. In the context of large scale ASD, team collocation, attitude, and morale are among important factors that were conducive toward the benefits, namely, improved communication and coordination (121 respondents), quick releases (101), flexibility of design—quicker response to changes (86), and more reasonable process (65), etc. This further emphasized how agile was adaptable in larger scale. Sophatsathit [Sophatsathit (2020)] proposed a manual activity planning technique, such as the symbolic flow map that fit and improved the manual workflow during user involvement. Anwer *et al.* [Anwer (2017)] reviewed several popular agile methods such as test-driven development (TDD), feature-driven development (FDD), dynamic system development model (DSDM), and Crystal methods that furnished their applicability for different project types, team sizes, and development environments. Glaiel *et al.* [Glaiel (2013)] examined the agile methods in comparison with the classical waterfall by creating the agile project dynamics (APD) model to study and experiment with its extensible design. Their insight contributed to agile management. Matharu *et al.* [Matharu (2015)] conducted an empirical study of agile methods in response to software change, yet still delivered high quality software products. They found the same popularity of Scrum method over Extreme Programming and Kanban methods. Kim *et al.* [Kim (2013)] looked into mobile application design that focused on components. This architectural view of system posed a challenge to the notion of flexible design and implementation that could very well fit with agile development method.

Głodziński [Głodziński (2019)] proposed a framework of project performance measurement and a set of measures that could be applied in project-based organizations. Buglione *et al.* [Buglione (2007)] pointed out issues related to estimation improvements in agile project based on several agile methodologies, e.g., time, velocity, and load factor. Such precautionary suggestions could benefit the measurement scheme to be developed in our study.

This research sets out to investigate various project activities pertinent to user involvement. In so doing, the transition from traditional waterfall model can be less disruptive to agile approach. Gemino *et al.* [Gemino (2021)] examined agile, hybrid, and traditional project management approaches to unveil that the first two approaches significantly increased stakeholder success over traditional one at the same budget, time, scope, and

quality. Due to the lack of industrial scale development setting, experienced developers are invited to establish the guidelines for agile method to be exercised. Three experiments are set up and described in the next section.

3. Proposed Method

Traditional measurement methods such as Lines-of-code (LOC) [Boehm (2000)], Function Point [Albrecht (1983)], and object point (OP) [Banker (1992)] have their pros and cons that, from time to time, lead to controversy. We decided to adopt COSMIC Function Point (CFP) [COSMIC (2017)]. The basic data movements used are entry, exit, read, and write which are shown in Figure 1. Their simplicity gradually becomes popular in business application software.

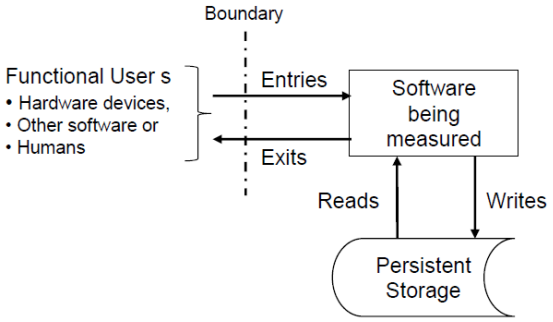

Fig. 1. The four types of data movements in CFP.

It is a well-known practice that traditional waterfall (WF) model is inflexible to fulfill frequent user requirements change during the development process. The problems are not only too late and difficult to go back revising the requirements as we progress on the development stages, but also are too expensive to repeat the processes already done to ensure that the changes are taken care of correctly.

Figure 2 shows a simplified WF flow of activities. The preliminaries involve a feasibility study conducted prior to the project commencement, producing the request for proposal (RFP) document for use in the project. Users are involved in the initial interviews by the developers to extract as much requirements as possible. When the requirements are settled, a contract is signed to mark the official commencement of the project. System activity and software architecture are established in accordance with the SRS document, followed by design, implementation, testing, documentation for the final deliverables, and sign-off. There is virtually no activity to accommodate requirement change during the development LC.

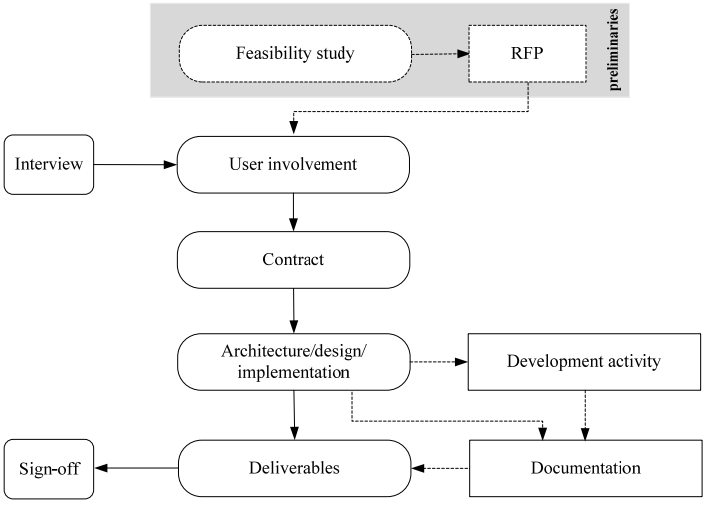

Fig. 2. Waterfall Development Practice.

Based on Figure 2, we envision some process improvement areas that could increase not only efficiency, but also quality of deliverables using different development practices. As mobile applications add frequent requirements change to working software, the development process must be adjusted accordingly. This requires systematic agility of operating procedures to handle the unceasing change requests.

To introduce user's change requests during project development, we propose two different user involvement schemes, i.e., coarse-grained (P1) and fine-grained (P2) involvements to make amend the above shortcoming. This is shown in Figure 3. The top Model T represents traditional WF development model, where the development process box encompasses typical WF development phases, namely, requirements, specification,

architectural and detailed designs, coding, testing, and ends with UAT, yielding the U_result. The middle Model P1 represents a coarse-grained which is a combination of WF and agile method. User involvement takes part in establishing SRS, architectural design, and resumes the phases of Model T until reaching testing where user involvement comes into play, followed by UAT yielding the C_result. The bottom Model P2 represents a fine-grained level of user involvement, where the user is a member of the team and stays involved for the entire development process, yielding the F_result. Note that the common activities during the development process, i.e., architectural and detailed designs, coding, and testing, are labeled as the development life cycle.

Model P1 is typically adopted by small firms as they realize the importance of agile flexibility which is suitable to their development setting. However, they cannot afford P2 possibly due to limitations of resources, in particular, time constraint that must deliver the output quickly within the budget. Model P2, on the other hand, can be fully exercised by larger firms since they can methodically go through the agile process to ensure high user satisfaction. At any rate, the two models employ common development life cycle to uniformly measure development activities in the same stages which allow comparable metrics for project effort, performance statistics, and other project outputs.

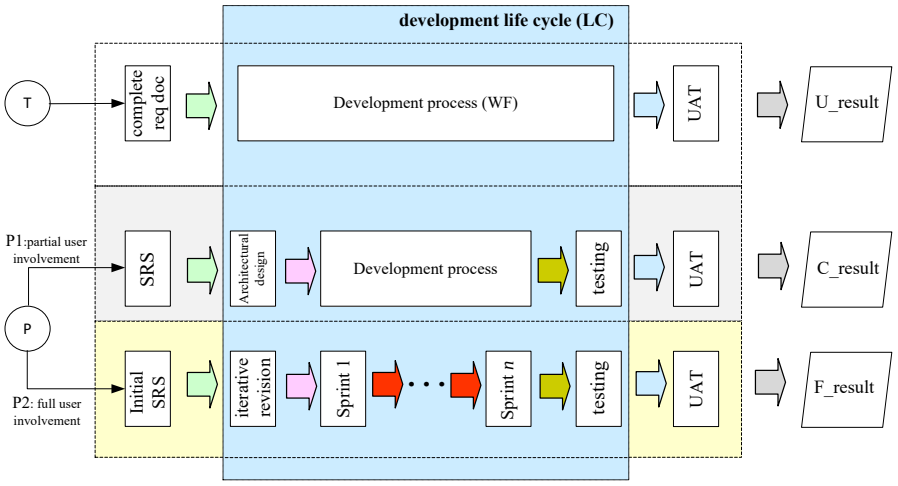

Fig. 3. Framework of the Traditional and Proposed approaches.

The essential project activity measurements to be considered in this study are as follows:

1. Change request: This is to measure the effort expended by user involvement in the development LC to complete change requests.
2. Rate of rework: This is to handle change requests and gauge the frequency of rework to support those changes based on team agility.
3. Time buffering: This is to measure the time it takes for additional allowances to accommodate the task performed to fulfill change request for the development life cycle.

These input data will be further deployed in the proposed framework to gauge the output performance measurements of U_result, C_result, and F_result from comparable aspects, namely, (1) effort of U_result, C_result, and F_result, and (2) comparative statistics of the above project activities resulting from (1).

4. Experiments

The following sections describe the experiment processes of building a mobile app for registration front-end in the order they are carried out. Three experiments are set up relatively similar in order to preserve the internal validity such as project requirements of the mobile app, design method (OOD), coding language, test cases, expected outcome, team size and ability. Therefore, the level of user detailed activity could be set up properly for a given project, thereby the results would yield comparable statistical inferences to satisfy the research objectives.

4.1. *Data collection*

Owing to the novelty of this research domain, no company would participate in the experiment since they could not afford the risk of project failure to their software deliverables. We could only conduct interviews from a few local developers to obtain the above project considerations. The interview focused on their experiences and the results gathered from agile method. Table 1 summarizes the feedbacks gathered from 4 local developers working for different firms to be used in our class projects. The statistics represent average values obtained from all four developers. Details on statistical outcome are further explained below.

Question	Change	Rework	Time
1. Frequency of requirement change request	42		
2. Ratio of change request being honored	**84**		
3. Amount of effort (man-hour) expended on each change request	37		
4. Success rate of rework (%)		82	
5. Number of members on the team	5		
6. Allowances (% effort) for performing assigned task	15		65
7. Amount of time the team spent on fulfilling a request			1360

Table 1. Change request and development statistics.

The first question reveals that on the average 42 change requests are issued for each project. However, not all change requests are honored since some are trivial, some are too tall an order to handle, others are expensive, unrealistic with state-of-the-practice technologies, etc. The number of outstanding requests becomes 42 * 0.84 = 35 requests. Hence, the effort required to carry out these outstanding change requests are equal to 35 * 37 = 1,295 man-hours. Yet the success rate is only 82%. This means that only 35 * 0.82 = 29 change requests out of 42 are fulfilled, and the effort needed to complete all change requests are 29 * 37 = 1,073 man-hours or 215 man-hours per member, based on 5-member team size. Consequently, the failed effort amounts to 1,295 – 1,073 = 222 man-hours. The team is operating at 1,073/1,295 = 83% efficiency, which turned out to be close to the ratio of change requests being honored.

Taking a 5% allowance for human factor consideration [Sophatsathit (2020)], which is 65 man-hours based on the overall effort expended on change requests of 1,295 man-hours, the total time spent on the change request is approximately 1,360 man-hours. Note that the shaded areas denote inapplicable values to the questions.

4.2. *Subjects*

Our subjects were senior in Computer Science working on their software projects. They were educated academically on various computer science subjects and practically trained with many local companies during junior summer internship. Consequently, they could well serve as qualified apprentice for this research [Salman (2015)].

4.3. *Experimental setup*

The students formed three small teams to develop a small scale mobile app, one to represent each model. App scale was not the prime issue since this study focused on performance analyses with and without user involvement that could affect the outcome of agile development process.

Model T was simple and straightforward to establish since they only followed the traditional waterfall model. Thus, change requests were taken care of in two stages, i.e., requirement change at ‘complete req doc’ and output change at UAT. P1 and P2, on the other hand, were somewhat setup differently. The students were encouraged to rotate task roles among themselves according to agile method. For example, they switched role daily from programming to testing, reading code, designing user interface, writing project report, and playing user. At each predetermined breakpoint, they gathered to hold a scrum meeting for status update on requirement change outcome. In so doing, it maintained development agility of team activity. The user was the key participant who made a request for requirement change at the scrum meeting. The request in turn was assessed to determine its extent (such as conflicting with existing requirement set, degree of complexity, effort and time needed to execute) whether the request should be upheld. If that was the case, a rework was assigned to the members of the team who were in charge of that requirement implementation, testing, and requirement validation. The final issue was the time it took to carry out such a request. For time performance analysis, we adopted the estimation units from Buglione [Buglione (2007)], namely, ideal time, velocity, and load factor. They are defined as follows:

- Ideal time:
  The time without interruption where you can concentrate on your work and feel fully productive
- Velocity:
  Measuring the project velocity, you simply count up how many user stories or how many programming tasks were finished during the iteration. Total up the estimates that these stories or tasks received
- Load factor:
  The load factor equals actual calendar days to complete a task divided by the developer's estimated "ideal" days to do it

4.4. *Procedures*

Three teams of students were set up to carry out T, P1, and P2 development experiments. Table 2 shows the team formation consisting of 7 members per team arranged in 6+1 manner. One member is designated to play the user’s role for the duration of three weeks, except for the Model T team as there was no role rotation. This set up adapted the proposed project team coordination by Brede Moe *et al.* [Brede Moe (2018)] that should

continuously change coordination mechanisms over time as the project was progressing, as well as exercised the notion of Kanban Coach suggested by Shamshurin *et al.* [Shamshurin (2019)] that would significantly improve team performance.

p#	activity	duration (wk)	iteration(wk)	team size
0	team formation/planning/breakdown	3	3	7
1	form setup	3	2	6+1
2	registration	2	1	6+1
3	data entry	2	2	6+1
4	data processing	4	3	6+1
5	summary	1	1	6+1
	total	15	12	

Table 2. Mobile apps profile with user involvement.

The iteration of this stage encompasses planning and work breakdown by all seven members of the team. This is denoted in p#0 entry. Entry p#1 denotes the form setup stage that lasts for three weeks, but iterates over role rotation among the 6+1 members for two weeks. The remaining one week was set aside for documentation duty. Entry p#2, …, p#5, denoting form setup, registration, data entry, data processing, and summary, respectively, were arranged in the same manner. However, the shaded areas did not apply to Model T. Note that project duration spanned 15 weeks over the entire semester, while the iteration process only cycled only 12 weeks. The three non-iteration weeks were allocated for documentation work, allowances, and general discussions.

Figure 4 depicts the flow of planned manual activities using symbolic flow map. This helped the team visually plan their manual activities in an orderly sequence. Any zigzag flows, as shown in Figure 4(a), would be rearranged to reduce disruptions of activity sequence, thereby smoothing and increasing the efficiency of manual activities. The rearrangement is carried out by swapping step 2 and 3 in Figure 4(a) to get step 1 and 3 done continuously. In the meantime, the meeting and inspection could be conducted in one sitting as shown in Figure 4(b). Thus, the symbolic flow map visually helps smooth out the flow of manual activity for the team.

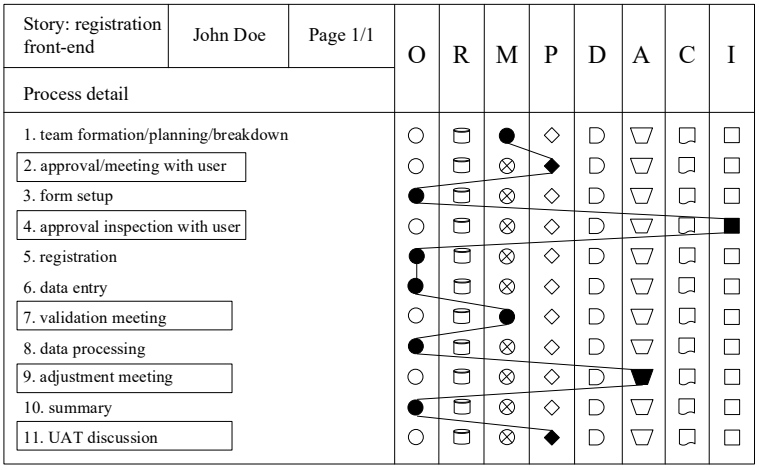

Fig. 4(a). Symbolic flow map of manual activity.

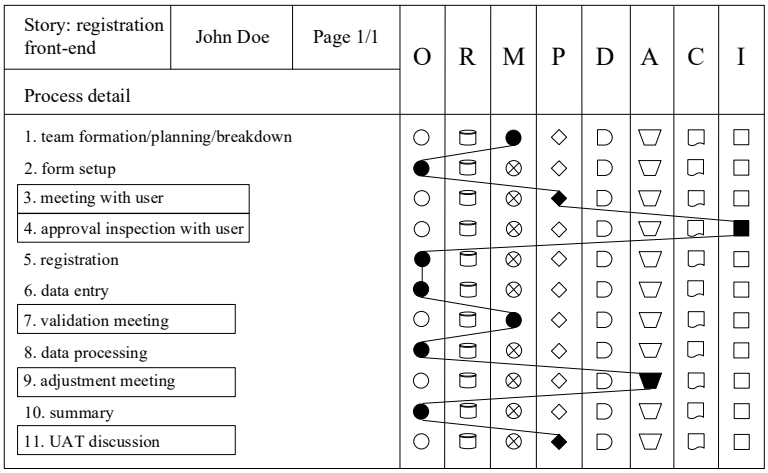

Fig. 4(b). Rearranged symbolic flow map of manual activity.

(legend: O-operation, R-read, M-meeting, P-planning/decision, D-delay, A-adjustment, C-code, I-inspection/review)

Table 3 shows an adaptation variation of a Kanban card for the above activities by the team. The local developers initially served as the Kanban coach for the teams to arrive at the analysis of user story below. The sample form setup stories are all feasible to work out. Their consistencies, however, are not all supporting one another. Neither does the completeness. Case in point, box type and fill-box tag are consistent since the tag should describe what the fill-box does. Nevertheless, the expressiveness of fill-box might not be fully reflected by the box type or placement position in that the complete requirements, in written language, cannot be narrated solely by description (textbox) or other selections. Combinations of them are often deployed to help make the requirement complete and proper placement makes it readable. For example, a radio button or checkbox may be used with additional textbox for users to add comments on that question.

As far as completeness of user story is concerned, stories 2-4 complete the design. The stories 1 and 5 concern file processing that do not contribute to the design of form. They merely support the operation of form implementation.

User story (form setup)	Feasibility	Consistency	Completeness
Create form file	√	1, 5	-
Create fill-box	√	2, 3	√
Create fill-box tag	√	2, 3	√
Select box type, placement position	√	3, 4	√
Modify, and save file	√	1, 5	-

Table 3. Analysis of user story.

Table 4 summarizes the frequency of rework by all models. Model T called for user involvement in p#0 and p#5 which was evident from UAT statistics. The initial requirements took 28 reworks and additional 16 reworks to get user acceptance. Activities p#1-4 in the development LC box (shaded in gray) were performed based on requirements document established in p#0 without user involvement. The final p#5 was done with 11 reworks and additional 25 reworks to get user acceptance.

The stories were slightly different for P1 during week 1-5 and 13-14, having higher reworks than week 6-12 because the latter went without user involvement. In P2, reworks were performed throughout the development LC and tapered down in user acceptance stage since all the changes were mostly taken care of in the preceding stages. The final three differences exhibit the load of rework incurred in all models. These will be further discussed along with the effort summary in Table 5.

Table 5 summarizes the effort measured by COSMIC function point. Model T shows high effort at p#0 owing to requirements establishment and increases at p#5 as user exercise their acceptance activities. Model P1 and P2, on the contrary, start slowly from p#0 and uniformly spread out the effort in the development LC. Thus, UAT in each stage takes less rework and effort to carry out as the user has primarily settled on the change requirements.

However, the differences of effort expended between T, P1, and P2 are worth mentioning. Both P1 and P2 expended less effort than that of T since user's requirements had not yet settled at p#0 as they evolved while the development went on. P1 struggled to get over the development LC performing higher reworks and effort than T. On the other hand, P2 spent more reworks (due to user involvement throughout the development LC) but less effort than T. The most noticeable difference was the effort expended by P2 which turned out to be considerably less than P1, despite there were relatively even number of reworks to perform. Incidentally, verifying the differences of rework and effort among the three models turns out to be consistent, that is, frequency difference

of rework between P2-P1 = 6 = (P2-T) – (P1-T) = (91) – (85) = 6; likewise, effort difference becomes P2-P1 = (-9) – (82) = -91.

p#	T		T-no_user involvement	P1				P1-partial_user involvement	P2	UAT	total P2	P1-T	P2-T	P2-P1
	wk 1-15	UAT		wk 1-5	wk 6-12	wk 13-14	UAT							
0	28	16	44	15	7	0	9	31	33	7	40	-13	-4	9
1	7	0	7	11	5	12	5	33	29	3	32	26	25	-1
2	4	0	4	9	4	8	4	25	24	2	26	21	22	1
3	10	0	10	12	6	7	7	32	37	3	40	22	30	8
4	12	0	12	14	5	9	8	36	40	3	43	24	31	7
5	11	25	36	17	4	14	6	41	21	2	23	5	-13	-18
												85	91	6

Table 4. Frequency of rework results with and without user involvement.

p#	T		T-no_user involvement	P1				P1-partial_user involvement	P2	UAT	total P2	P1-T	P2-T	P2-P1
	wk 1-15	UAT		wk 1-5	wk 6-12	wk 13-14	UAT							
0	51	84	135	30	0	0	35	65	58	16	74	-70	-61	9
1	18	0	18	18	16	19	18	71	42	11	53	53	35	-18
2	11	0	11	15	22	11	12	60	30	7	37	49	26	-23
3	17	0	17	14	19	10	21	64	39	10	49	47	32	-15
4	22	0	22	19	10	14	24	67	40	8	48	45	26	-19
5	14	92	106	11	12	24	17	64	32	7	39	-42	-67	-25
												82	-9	-91

Table 5. COSMIC function point measure of effort.

Model	Ideal time	Velocity	Load factor
T	3	0/-	5/3
P1	2	46	3/2
P2	1	83	2/1

Table 6. Time analysis.

Table 6 shows time analysis of all models based on the definitions given earlier. Model T exhibits the highest ideal time since the developers could work continuously without any interruption from the user, except meeting on work-in-progress update. However, the functional and non-functional requirements were inherently different from user stories, thereby the velocity was inapplicable. Developer of Model P1 only interacted with user at the beginning and final stages kept the velocity somewhat less than that of P2, where user injected change requests regularly. As a consequence, the velocity was relatively high. Finally, Model P2 showed the highest load factor as the developers had to take care of change requests frequently.

5. Discussions

The results reveal some beneficial aspects that can be drawn below.

5.1. *Practitioner's benefits*

The amount of rework is quite heavy at the beginning and the end of the project under T. This is predictable since the emphasis must be placed on SRS to get a correct project outcome. A stringent UAT must also be exercised at the end to reach the stated outcome in SRS, causing high rework effort inevitably since there is no user involvement in between. Nonetheless, the overall of rework frequency and effort for the project are moderate as shown in box-plot 'A' of Figure 5 and 6.

P1 is in no different situation. The coarse-grained user involvement in architectural design (wk 1-5) and testing (wk 13-14) stages cause slight increase in rework frequencies, while the middle development stage (wk 6-12) remains low. The overall project numbers are relatively even with T. This is shown in box-plot 'B' of Figure 5 and 6 which enclose all the above three stages.

Model P2 exhibits considerable lower rework than those of P1 and the T models as shown in box-plot 'C' of Figure 5. This is the direct effect of user involvement. It is apparent from Figure 6 that these numbers are also proportional to the project effort. The higher the effort, the higher the development cost.

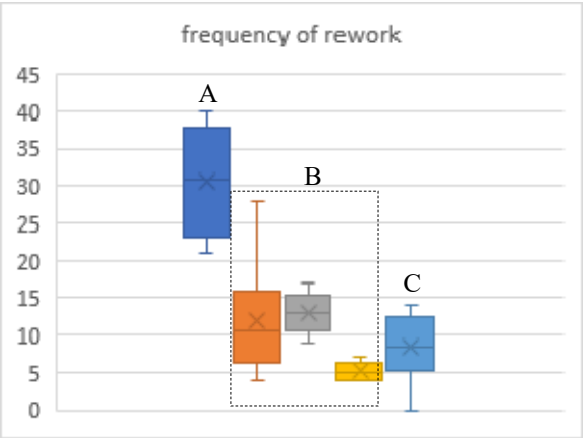

Fig. 5. Frequency of rework.

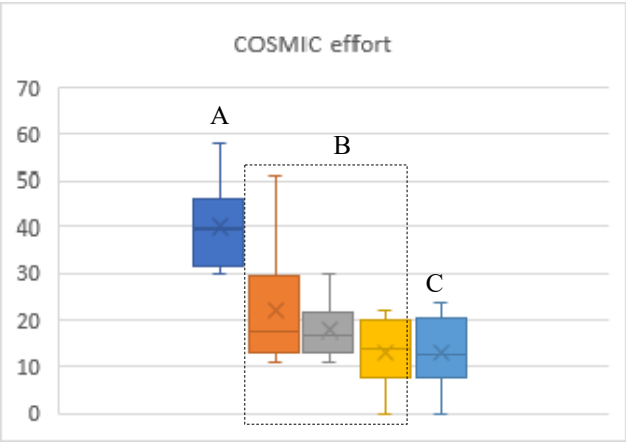

Fig. 6. COSMIC effort.

If we focus closer to the end of project, we can see that the UAT plots turn the table around. Frequency of rework and effort of model T exploded as demonstrated in box-plot 'A' of Figure 7 and 8, respectively. More user involvement under P1 than T reduces UAT rework and effort considerably as shown in box-plot 'B' in the figures. The best outcome turns out as predicted—under P2 with full user involvement. From users and software product point of view, these numbers mean higher user satisfaction.

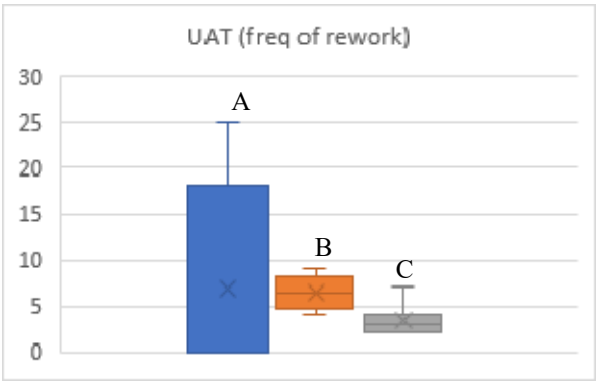

Fig. 7. UAT frequency of rework.

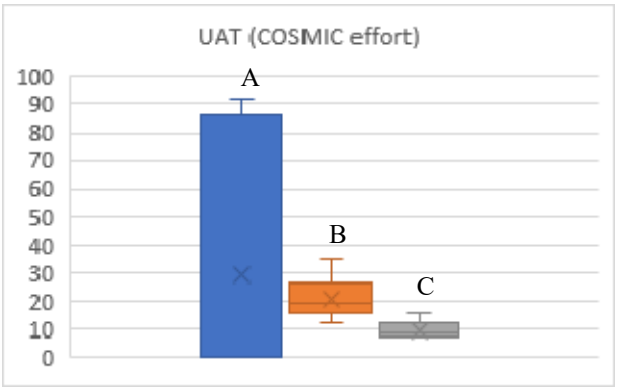

Fig. 8. UAT COSMIC effort.

5.2. *Small production*

We requested preliminary the local developers to exercise P1 since they employed similar coarse-grained waterfall-agile method. Table 7 shows the effort expended by a 3-person team working on a small sub-system of a project. The project manager served as the user of the team.

p#	P1				total
	wk1-5	wk6-12	wk13-14	UAT	
0	118	65	0	75	258
1	86	48	89	49	272
2	72	37	71	36	216
3	91	58	64	63	276
4	110	47	76	72	305
5	124	39	110	57	330

Table 7. COSMIC measure of effort.

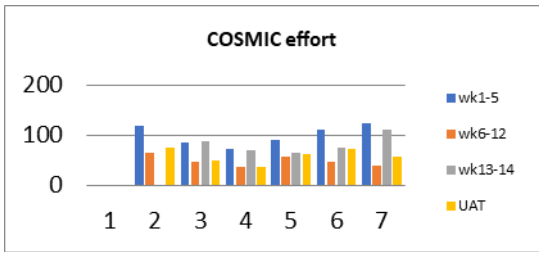

Fig. 9. COSMIC effort.

Figure 9 shows the plot of output COSMIC effort which reflects unsurprising results similar to those of the experiments. The initial requirements setup (wk1-5) took the highest effort expended, followed by corrective effort during wk13-14 in preparation for UAT step. As a consequence, the practicality of user involvement in proper level of activity breakdown helped reduce the effort as proposed by this study.

6. Conclusion

This study has reaffirmed the benefits from two aspects of the proposed method, i.e., user involvement and level of activity breakdowns in software development process model. We did not target for any innovative method since we focused on solving the problems for software developers to obtain some practical ways of working that constituted the bread-and-butter of software project. The results were satisfactory in both qualified computer science seniors and a small professional software development team that contributed to slightly higher user acceptance.

As far as time analysis was concerned, the three factors ideal, time, velocity, and load factor were not quite consistently contributed to any models. No single model out-performed others for adopters to decide what to follow.

One important result obtained from this study is domains of applicability. Care must be taken in adapting the results to the appropriate team size, project scale, and development model since agile may not fit some development/application settings. At any rate, future work should focus on the quality of software deliverables that can be directly or indirectly affected by the degree of user involvement.

References

[1] Abrahamsson, P.; Hanhineva, A.; Hulkko, H.; Ihme, T.; Jäälinoja, J.; Korkala, M.; Koskela, J.; Kyllönen, P.; Salo, O. (2004): Mobile-D: An Agile Approach for Mobile Application Development, OOPSLA'04, Vancouver, British Columbia, Canada.

[2] Albrecht, A.; Gaffney, J **.J.** (1983): Software function, source lines of code, and development effort prediction **:**A software science validation, IEEE Transactions on Software Engineering,9(6), pp **.**639–648**.**

[3] Anwer, F.; Aftab, S.; Waheed, U.; Muhammad, S. S. (2017): Agile Software Development Models TDD, FDD, DSDM, and Crystal Methods: A Survey, International Journal of Multidisciplinary Sciences and Engineering, 8(2), pp. 1-9.

[4] Banker, R**.**; Kauffman, R**.**; Kumar, R**.** (1992): An empirical test of object-based output measurement metrics in a computer aided software engineering (case) environmen, Journal of Management Information Systems, 8(3), pp **.**127–150**.**

[5] Begel, A.; Nagappan, N. (2007): Usage and Perceptions of Agile Software Development in an Industrial Context: An Exploratory Study, First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007).

[6] Boehm, B.; Abts, C.; Brown, A.; Chulani, S.; Clark, B.; Horowitz, E.; Madachy, R.; Reifer, D.; Steece, B. (2000). *Software Cost Estimation with COCOMO II*, Upper Saddle River, NJ :Prentice Hall PTR.
[7] Brede Moe, N.; Rolland, K.; Dingsøyr, T. (2018): To schedule or not to schedule? An investigation of meetings as an inter-team coordination mechanism in large-scale agile software development, International Journal of Information Systems and Project Management, 6(3), pp. 45-59.
[8] Buglione, L.; Abran, A. (2007): Improving Estimations in Agile Projects: Issues and Avenues, Proceedings Software Measurement European Forum (SMEF), pp. 265-274.
[9] Drury-Grogan, M. L. (2021): The Changes in Team Cognition and Cognitive Artifact Use During Agile Software Development Project Management. Project Management Journal, 52(2), pp. 127–145.
[10] Gemino, A.; Reich, B. H.; Serrador, P. M. (2021): Agile, Traditional, and Hybrid Approaches to Project Success: Is Hybrid a Poor Second Choice? Project Management Journal, 52(2), pp. 161–175.
[11] Glaiel, F.; Moulton, A.; Madnick, S. (2013): Agile Project Dynamics: A System Dynamics Investigation of Agile Software Development Methods, Working Paper CISL# 2013-05, Composite Information Systems Laboratory (CISL), Sloan School of Management, MIT, Cambridge, MA 02142, pp. 1-29.
[12] Głodziński, E. (2019): Performance measurement of complex project: framework and means supporting management of project-based organizations, International Journal of Information Systems and Project Management, 7(2), pp. 21-34.
[13] Guideline for Sizing Business Application Software—The COSMIC Functional Size Measurement Method, versions 4.0.1/4.0.2 (2017): https://cosmic-sizing.org/wp-content/uploads/2017/05/ COSMIC-Method-v4.0.1-Bus-App-Guideline-v1.3-1.pdf.
[14] Kim, H. K. (2013): Architecture for Adaptive Mobile Applications, International Journal of Bio-Science and Bio-Technology, 5(5), pp. 197-210.
[15] Lee, G.; Xia, W. (2010): Toward Agile: An Integrated Analysis of Quantitative and Qualitative Field Data on Software Development Agility, MIS Quarterly, 34(1), pp. 87-114.
[16] Matharu, G. S.; Singh, H.; Mishra, A.; Upadhyay, P. (2015): Empirical Study of Agile Software Development Methodologies: A Comparative Analysis, ACM SIGSOFT Software Engineering Notes, 40(1), pp. 1-6.
[17] Salman, I.; Misirli, A.; Juristo, N. (2015): Are students representatives of professionals in software engineering experiments?, 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Florence, Italy, 2015, pp. 666-676.
[18] Shamshurin, I.; Saltz, J. S. (2019): Using a coach to improve team performance when the team uses a Kanban process methodology, International Journal of Information Systems and Project Management, 7(2), pp. 61-77.
[19] Sophatsathit, P. (2020): Software Analytics for Manual Activities using Developer Work Elements, Journal of Information Processing, Information Processing Society of Japan, 28, pp. 279–291.
[20] Wiesche, M. (2021): Interruptions in Agile Software Development Teams, Project Management Journal, 52(2), pp. 210–222.

Authors Profile

Nalinee Sophatsathit, receives her bachelor degrees in Communication Arts from Sukhothai Thammathirat University and Computer Science from Phetchaburi Teacher's College, Masters in Computer Science from College of Engineering, Chulalongkorn University, and Ph.D. in Innovation Management from Innovation College, Suan Sunandha Rajabhat University. She is a faculty member in Computer Science Program, Faculty of Science and Technology, Suan Sunandha Rajabhat University. Her research interests are innovation in IT and software usability. She can be reached at nalinee.so@ssru.ac.th.