A Cost Estimating Method for Agile Software Development

Shariq Aziz Butt[1](✉), Sanjay Misra[2], Gabriel Piñeres-Espitia[3], Paola Ariza-Colpas[3], and Mayank Mohan Sharma[4]

[1] The University of Lahore, Lahore, Pakistan
[2] Covenant University, Ota, Nigeria
sanjay.misra@covenantuniversity.edu.ng
[3] Universidad de la Costa, CUC, Barranquilla, Colombia
{gpineres1,pariza1}@cuc.edu.co
[4] Zillow Inc, San Francisco, USA
mayankmohans@zillowgroup.com

Abstract. In every software development project, the software effort estimating procedure is an important process in software engineering and always critical. The consistency of effort and timeline estimation, along with several factors, determines whether a project succeeds or fails. Both academics and professionals worked on the estimation approaches in software engineering. But, all these approaches have many problems that need to be addressed. One of the most difficult aspects of software engineering is estimating effort in agile development. This study aims to provide an effort estimation method for agile software development projects. Because in software engineering, the agile method is widely used for the development of software applications. The development and usage of the agile method are described in depth in this study. The framework is configured with empirical data gathered by projects from the software industry. The test findings reveal that the estimation method has great estimation accuracy in respect of mean magnitude of relative error (MMRE) and Prediction of Error PRED (n). The suggested approach achieves more accuracy for effort estimation as compare to others.

Keywords: Software effort estimate · Agile development · User stories · Metrics and measurement · Maintainability

1 Introduction

Since the 1940s, when the software era began, estimating software costs has been a critical but challenging task. Because the scale and significance of software applications have increased, so has the accuracy of software cost estimation. Software development professionals and academics are now working on developing techniques to measure development costs and schedules throughout the early 1950s. In the last 3 decades, software effort estimation techniques have been reported in the existing research. On the other hand, the domain of software effort estimation is still in its infancy [1, 2].

O. Gervasi et al. (Eds.): ICCSA 2021, LNCS 12955, pp. 231–245, 2021.
https://doi.org/10.1007/978-3-030-87007-2_17

Despite the introduction of various software cost estimation approaches that are successfully used in conventional software development, the complexity of modern software development techniques has resulted in a circumstance in which the usefulness of current effort prediction techniques tends to be limited [3]. One major challenge is agile software development. This approach is focused on a completely different definition of application development which can be measured using the FP analysis approach, nor can traditional effort estimation approaches that were designed primarily for sequential application development methods be used. As a modern software engineering approach, that is, agile development has received a lot of attention. It emphasizes good developer communication, fast product delivery, and change on request as the core components of agile development. For industrial applications, agile development practices are becoming more prevalent. The usage of effort estimation techniques in these types of projects is a challenging but essential process. Traditional estimation techniques necessarily require well-defined specifications. Agile methodologies do not support this activity. Changing demands, on the other hand, is viewed as a significant problem. Both of these factors are rendering estimation difficult in agile development. This paper provides an overall view of the current estimation methods and explains how to estimate agile development projects in depth [4, 5]. We are also proposing an estimation technique in this paper and used it for a case study. We also use a case study in this study and explain the outcomes in detail.

The paper contains 5 sections. Next, Sects. 2 and 3 summarize the works on cost estimation techniques and agile methodologies—Sects. 4, 5, and 6 present the proposed work, experimentations and conclusion, and future work.

2 Cost Estimation Techniques

To measure project costs, cost estimation methods, also known as model-dependent estimation methods that integrate statistics from previous projects with mathematical equations. As an input, these methods include the size of the system. COCOMO, SLIM, RCA PRICE-S, EER-SEM, and ESTIMACS are the most popular model-based techniques. Regression-based models, learning-oriented models, expert-based approaches, and composite Bayesian methods are some of the current effort estimation techniques. The regression methodology is used in the majority of software estimation models. Regression models are typically built using previously collected data from completed projects and the development of regression equations that describe the relationships between various variables. The mathematical method is based on the new project scope. To make predictions, this model is assessed using regression data. In such methods, the effort for efficient software development is merely a dependent variable for regression equations involving some expected variables such as size, effort adaptation factor, scale factor, and so on [6]. In certain cases, therefore, regression models require the fulfillment of specific requirements. Boehm and Sullivan (1999) addressed these conditions, which are focused on their experience with regression-based methods. These standard conditions contain the availability of a considerable dataset, with the absence of missing data objects, the absence of outliers, and the absence of correlation between the predictor factors. Ordinary least-squares regression (OLS), classification and regression trees (CART), step-wise examination of variance for inconsistent data sets (stepwise ANOVA), configuration of CART with OLS regression and comparison, multiple linear regression, and stepwise regression are all examples of regression models [7].

Other kinds of models, known as Learning-oriented methods, are focused on prior estimation knowledge and learning. These approaches aim to automate the estimating process through training themselves to create automated models based on existing experiences. These methods are able to incremental learning and refinement as new data is presented with time. Artificial intelligence methods, artificial neural networks, particular scenario analysis, machine learning techniques, decision-tree learning, fuzzy logic approaches, content knowledge, and linear regression are all examples of learning-oriented modeling techniques. COCOMO, SLIM, RCA PRICE-S, SEER-SEM, and ESTIMACS are the most popular model-based approaches. Depending on criteria like the size and required functionality of the application, these calculation approaches provide an estimated cost, effort, or time of a project. "Comparative analysis with previous related projects depending on individual memory" was discovered to be an effective expertise-based method. Where neither quantified quantitative evidence is available, expertise-based methods are beneficial [8].

They offer a simple, low-cost, and extremely productive method. Analogy-based estimation is another estimation method of estimating software effort. The method looks at previous projects and is using the details obtained as a rough approximation for the current project. The Checkpoint approach is a representation of a software estimation technique dependent on analogy. Heuristics are extracted from real project data or a formalization of expert analysis in this method. Some type of project data and details is being used to extracting these heuristics. The performance of these heuristics can then be used to estimating efficiency, quality, and scale. Expert opinion estimation is another common estimation method within software effort estimate and it is focused on the collective expertise of teams of expertise to generate project estimates. Where the estimation method is largely focused on "non-explicit, non-recoverable reasoning mechanisms," such as observation and experience, this method has been used.

Experts have chastised expert judgment methods regarding their dependency on human memory and the absence of replicability of these memory-based methods. Still, studies reveal that it is the most popular method for application development and estimation. Expert decision methods include the Delphi method and job breakdown structure (WBS), top-down and bottom-up estimation, and thumb rule [9].

The advantages of expertise-dependent estimation approaches are merged to implement a new semiformal estimating approach known as the Bayesian approach. The bayesian analysis considers that almost all estimation methods require datathat is either of limited quality or incomplete. This approach incorporates expert judgment to manage missing data and enables a better rigorous estimation method. The COCOMO II model was developed using Bayesian analysis, which is used in multiple scientific fields. A hybrid estimation method is Cost Estimation, Benchmarking, and Risk Analysis (COBRA) [10].

3 Agile Software Development

Agile software development is a set of iterative and gradual application development approaches in which specifications and ideas emerge from collaboration within self-organizing, cross-functional teams. Evolutionary planning, evolutionary growth and execution, a time-boxed iterative strategy, and fast and scalable responsiveness to change are encouraged. It has a theoretical model that encourages foreseen experiences during the creation process. In 2001, the Agile Manifesto invented the term. Scrum, Crystal Clear, Extreme Programming, Adaptive Software Development, Features Oriented Development, and Dynamic Systems Development Process are examples of earlier lightweight approaches. Following the publication of the Agile Manifesto in 2001, these are now commonly pointed to as agile methodologies. Methods vary from adaptive to predictive. In this continuum, agile strategies are on the adaptive side. Adaptive approaches concentrate on rapidly adjusting to changing circumstances. When a project's requirements change, an agile team must respond as well. Predictive strategies, on the other hand, concentrate on meticulously planning the future. For the duration of the development phase, a predictive team may disclose precisely what functionality and activities are expected. Predictive teams have a hard time modifying functionality [10].

Agile Software Processes Features
Modularity: it is a significant aspect of the agile software development process since it enables a process to be split down into modules called activities.

Iterative: Agile software systems are iterative, based on short intervals. A specific collection of tasks is performed during each loop.

Time-Bound: Each iteration and scheduling has a time limit. Sprint is the name given to this period.

Parsimony: is a key component of agile software development. That is, they only need a limited number of activities to reduce risks and accomplish their objectives.

Adaptive: The agile process adapts to new threats that are discovered through any iteration. Likewise, agile processes can handle any new operation or changes to existing ones [11].

Incremental: Agile processes are incremental, rather than trying to construct the whole system at once. Instead, it divides the complex structure into increments that can be worked on in sequence, at various times, and at different speeds.

Convergent: The central tenet of agile development is to get the structure closer to reality. This aim is attained by using all available strategies to achieve performance as quickly as possible [12].

People-Oriented: Agile methods emphasize people over processes and technologies. They change organically as a result of adaptation. Motivated developers increase their efficiency and performance. After all, they are the most skilled people in the industry to create these improvements.

Collaborative: Agile methods enable team members to interact with one another. Any software development project needs effective communication. Collaboration is needed to quickly integrate a large project, although increments are now being created in parallel.

3.1 Effort Estimation in Agile Modeling

In waterfall, the manager calculates a team member's productivity potential by calculating how long those activities may take and then assigning work depending on the team member's maximum time available. In regard to evaluating a team member's ability, agile methodology provides a specific method. To begin with, it assigned work to an overall team rather than a single person. This stresses the importance of group effort from a theoretical perspective. Second, it declines to calculate work in terms of effort due to doing so would adversely affect the self-organization necessary for the methodology's effectiveness. This is a substantial departure from the waterfall: rather than a boss, calculating time on behalf of others and setting priorities based on conjecture, Scrum team members determine their own tasks based on commitment and degree of complexity [13].

There is no only method for teams to quantify their work in Agile Methodology. It recommends that teams cannot calculate effort in terms of time but rather use a much more obfuscated measure. Numeric scaling, t-shirt size, the Fibonacci series are all common estimation approaches. The most significant thing is that the team has shared knowledge of the scale it employs. Therefore each team member is familiar with the scale's values.

The team gathers for the Sprint Planning Session to quantify their commitment to the user stories in the backlog. The product client requires these forecasts in order for him or her to efficiently priority products in the backlog and, as a response, predict delivery depending on the team's velocity. This necessitates an accurate evaluation of the work's complexity by the Product Owner. To avoid forcing a team to decrease its commitment evaluation and bring on new work, it is suggested that the Product Owner does not follow the estimations. And when a team determines itself, measures must be taken to decrease the cost of influencing those results. As a result, it is suggested that all team members reveal their predictions at the same time. This method parallels a game of poker in that everyone "shows their hands" at the same time [14].

And when teams have a common interpretation of their size, they cannot help but make various estimates. It also takes several rounds of calculation to conclude in a single effort estimate which represents the entire team's perception of a story's complexity. Experienced teams, on the other hand, should be able to reach an agreement after merely a few sessions of planning poker. Usually, effort calculation occurs throughout Release Preparation at the start of the new iteration. The figure depicts a part of the XP-Project [15].

Research Problem

The majority of current effort estimation methods were designed to help conventional sequential application development methodologies, while Agile Software Development is an iterative process. If conventional methods are used to estimate the commitment of agile software projects, the findings would almost certainly be unreliable. The existing practice of effort estimation for agile development projects, on the other hand, is focused on a complete iteration. As a result, an effort estimation method is required to estimate the development effort for agile software projects depending on Agile Software Development features [13].

4 Proposed Model

The majority of Software Effort Estimating Models estimate a project's budget, length, and resources. In the case of Agile Development, however, this will not be the case. There are some main distinctions within the agile and conventional approaches to team organization.

4.1 Agile Teams Are Whole Teams

The whole team is an Extreme Programming (XP) activity that suggests getting enough expertise inside the team to complete the mission. The idea is that the software team possesses the necessary testing, databases, user interface, and other expertise and does not depend on external experts or specialists' teams.

4.2 Agile Teams Are Formed (Mostly) of Generalized Specialists

A generalizing professional, also known as a craftsperson, is someone who has one or even more technological skill sets to which they can directly contribute to the team has had at least a basic understanding of the software development and the application domain wherein they operate, and most significantly consistently tries to learn new expertise in both their current professions and other fields. Evidently, new IT specialists and experienced IT professionals will have to work for this objective because they have often experienced just one field. The sweet spot among the two extremes of expertise is generalizing with experts.

4.3 Agile Teams Are Stable

Agilests realize that shifting team composition can be counterproductive to a project's progress.

A performance we work hard to maintain our team as stable as possible, which is much simpler to do when people are generalizing experts.

Since there is no requirement to measure the project's manpower needs, the suggested approach is designed to measure the Agile Software 'project's delivery time and expense.

Agile professionals, particularly Scrum professionals, have suggested a range of scales for measuring project expected effort, such as:

- On a level of one to three, we rate the effort, with one being the minimum and three being the maximum.
- The Fibonacci series [1, 2, 3, 5, 8.... ... so on] is used. So, for example, a Story rated as eight is one that is too big to measure accurately and must be graded as an epic and broken down into shorter stories.

Other approaches exist, but these are two widely prominent. In both cases, the calculations are not formulated in terms of time modules; instead, they are simply measures of Relative Effort, that is, a reasonable yardstick for comparison. Both of these approaches are successful and commonly used, but they ignore the fundamental factors that influence effort and uncertainty. As a result, we created a new model which we think is very successful. This method is also consistent with how to build Plot, defect, and risk rankings [16].

4.4 Determining the Effort

A number of factors influence our potential to calculate effort precisely. To produce reliable and efficient estimates, precise prediction requires a multivariate model. The problem is deciding which dimensions to calculate. Suppose we use a SWOT analysis to category the scenarios based on internal vs. external factors. In that case, we can exclude several possibilities by concentrating our focus on the factors we can control and paying little consideration to the factors we cannot. We restrict the vectors to two to make the method as easy as practicable so that we use it and do not try to avoid it because 'it's too complicated. The use of two vectors is also consistent with the rest of the technique.

4.5 Story Size

The project size is a rough approximation of the work's relative size in terms of real development time. Table1 displays 5 elements that have been allocated to various types of user stories based on their scale. The team has the potential to adjust the wording of the Guideline summary and redefine the requirements.

Table 1. Scaling of stories**.**

Value	Rules
5	• An incredibly long story • Moreover broad to estimation correctly • Almost definitely could be divided into a series of minor stories • Could be separated into a new project
4	• A huge size of story • Needs a developer's concentrated effort over a prolonged period of time (think and over a week of task) • Try splitting it up into a series of shorter stories
3	• Abstemiously huge stories • Consider working for two to five days
2	• Plan on putting in a day or two of effort
1	• A very short tale with a very low effort amount • Imagine working for just a few hours

4.6 Complexity

This is the level of difficulty in one or both the Story's specifications and its technical complexity. The estimate's uncertainty causes concern; the higher the difficulty, the greater the uncertainty. Table 2 displays the five meanings that have been allocated to user stories based on their existence. These rules, such as the Story Length table, are not set in stone. It can be tweaked through the team, but we have classified them to account for all aspects of agile software development [17].

Table 2. Scaling of user stories as per difficulty.

Value	Guidelines
5	• Exceptionally difficult • There are several needs on other stories, processes, and subcomponents • Signifies a valuable set of skills or expertise that's also missing from the team • It's complicated to define how difficult it is to say a story • Many unknowns • Important refactoring is required • Substantial research is needed • It necessitates making tough decisions • Relative to the story, the story's consequences have a huge influence
4	• Huge Difficult • There are several dependency on other stories, processes, or subcomponents • Signifies a valuable set of skills or expertise that's not well-represented on the team • The product owner may find it challenging to accurately explain the story • A considerable amount of refactoring is needed • Completion necessitates specialized programming skills • It necessitates some tough decisions
3	• Interconnections on other stories, structures, or subsystems are medium • Signifies a relatively good skill set or expertise within the team • The owner finds it challenging to correctly interpret the tale • Completion necessitates intermediate technical skills • The story's implications have a minor influence outside of the story
2	• Technical and business specifications that are easy to comprehend • To finish, you'll need basic to moderate tech skills • The Story's impact are mostly entirely contained inside the Story
1	• There are very few, if any, unknown factors in this situation • There are no ambiguities in the technological and business specifications • The Story's impact are entirely contained inside the Story

The effort of a specific User Story is estimated with these two parameters by the following basic equation as shown in Eq. 1:

$$Effort\ of\ Suer\ Story = ES = Complexity \times Size \tag{1}$$

The overall project effort will become the amount of all actual customer story efforts as per Eq. 2.

$$E_f = \sum_{x=1}^{n} (ES)_x \tag{2}$$

The cumulative project commitment would be equal to the number of all client story efforts.

4.7 Defining Agile Velocity:

The length is measured in the unit of effort, and the time (numerator) is the duration of the sprint. As a result, the velocity is measured as follows in Eq. 3:

$$V = \frac{User\ Stories\ Completed}{Sprint\ Time} \tag{3}$$

In a standard sprint, the Measured Velocity is literally the number of user stories of effort your team achieves. Velocity is classified as the amount of product backlog effort the team may manage in one module of time in Agile.

4.8 Velocity Configuration

Before you start calibration, you must finish the optimization method. Therefore, in our calibration, there are focused on two aspects.

i. The Friction or Persistent Factors that minimize Project Velocity also are a persistent drag on performance.
ii. The Variable or Dynamic Factors which trigger the Project Velocity to be abnormal by decelerating the project or members of the team.

Prior to calibration, optimize both of these variables to increase the accuracy of the velocity measurement. It is necessary to get a reliable Velocity because it's the base for several Agile and Scrum parameters.

Numerous powers can impact the team's velocity in software development. It is your responsibility as a team leader, project manager, or manager to reduce external factors that negatively affect the team's speed.

- Staff placement is the team made up of the right required skills?
- Improvements to the procedures, such as agile techniques, build, deliver, testing, and so on.
- Interruptions, noise, insufficient ventilation, low lighting, awkward seating and chairs, insufficient hardware or software, and so on are all examples of environmental causes.
- Team dynamics certain team members could not get along with others.

The environment influences majority of friction forces. Their consequences are long-lasting. They are often usually the easiest to deal with. Friction forces are typically weak forces on their own. They can have a substantial effect when added together. They must be removed or minimized to achieve maximum Team Velocity. The sum of all four fraction factors is used to measure friction, as presented in Eq. 4.

$$FR = \sum_{x=1}^{4} (FF)_x \tag{4}$$

4.9 Variable or Dynamic Forces:

Forces that are complex or dynamic are often volatile and unpredictable. They slow down the project and will its velocity. Their effects may be dramatic at times; however, they are typically temporary. For our objectives, we are looking at the cost (in terms of productivity) to members of the team and the team as a whole. If you cannot completely eradicate a force that decreases velocity, render it as stable and accurate as possible (minimal and infrequent deceleration). Your velocity would be more consistent and reliable if the force is stable and predictive [18].

- Team changes: new members, changes, and shifts in duties and responsibilities.
- New software: Discovering new programming tools, database systems, languages, and so on necessitates a reduction in velocity until they are mastered.
- Outside-of-project tasks: Team members carry on additional duties outside of the project. Switching projects can have a considerable impact on efficiency.
- Stakeholders: Stakeholders can be slow to respond to requests for information from developers or testers, causing delays. They may well have unrealistic expectations from the teams as well.
- Changing project requirements: New project requirements can necessitate expertise that the team lacks or is lacking. Obtaining the expertise, whether by introducing new team members or acquiring the skill set of a current team member, would affect productivity.

Table 3 shows the force factors that are variable or adaptive. Again, the same comparison is used to assign a value as it can be for size.

Table 3. Friction factors

Variable factor	Moderate	High	Very high
Changes in the Team to Be Required	1	0.87	0.86
New Tools are Introduced	1	0.88	0.74
Activities of members of the team beyond of the project	1	0.86	0.98
Stakeholder responses is supposed to take longer than expected	1	0.98	0.77
Inconsistency in the details is to be anticipated	1	0.85	0.87
Changes in the environment that are anticipated	1	0.88	0.84

The product among all six predictor factors is used to measure Dynamic Force (DF), as mentioned in the Eq. 5.

$$DF = \sum_{x=1}^{9} (VF)_x \tag{5}$$

The frequency of negative velocity eventually became known as deceleration. Deceleration is the result of friction and Dynamic Forces acting on the velocity in our scenario. It's estimated as follows Eq. 6:

$$D = FR \times DF \tag{6}$$

We quantify Final Velocity in addition to adapting velocity towards a more predictable range in Eq. 7:

$$V = (Vx)^D \tag{7}$$

4.10 Completion Time

The time required to complete the project time of completion as per Eq. 8:

$$\begin{aligned} T &= \frac{E}{V}\, days \\ &= \frac{\sum_{x=1}^{n} (ES)_x}{(Vx)^D} days \end{aligned} \tag{8}$$

In this computation, as mentioned in Eq. 9, the unit of T is days that may be transformed to months by dividing by the number of working days per month. Here the *WD* is describing the working days.

$$T = \frac{\sum_{x=1}^{n} (ES)_x}{(Vx)^D} \times \frac{1}{WD}\, days \tag{9}$$

4.11 Cost of Development

While there is no specific attribute in the method to estimate cost, some of the Agile Software Development team is also present. We performed a survey of 14 software industries at CMMI level 3 to determine monthly spending per project using Project Team Salary as the component. Because some industries have multiple teams working on multiple projects at the same time, all costs have been estimated for one project per month. Table 4, the average monthly costs, as well as their ratios to Team Salary, are discussed.

The Development Cost are computed using the Net Ratio of the Table 5 is as follows:

$$Cost = 2.933 \times TS \times T \tag{10}$$

Here T is the computed time in months, and TS is the monthly Team Salary mentioned in Eq. 10.

Table 4. monthly average costs, their ratios to Team Salary.

Costs	Amount	Ratio as per 'Team's Salary
Team salary	457897	0.787
Non IT members salary	147895	0.558
Tools	25436	0.444
Softwares	7845	0.478
Rent	12456	0.245
Traveling	32456	0.024
Repair and maintenance	7854	0.255
Changes accommodation	2547	0.142

4.12 Uncertainty of Calculation

The estimated time is only likely predicted when you are not assured with estimates. For example, when you are 100 percent assured in your estimate, the estimated time would also be the highest possible time; however, if you are not assured with your estimation method, the estimated time will only be unlikely predicted. Based on the level of assurance, the times vary in this situation. This variation is known as the span of uncertainty. Optimism Point is the lower limit of this range, and Pessimistic Point is the upper bound. We present a new variable with Confidence Level (CL) that will be used to estimate optimistic and Pessimistic Time in calculating the esteem impact on time as stated in Eq. 11.

$$Time_{prob} = T$$

$$Time_{prob} = \frac{1-(100-CL)}{100} \times T \tag{11}$$

Summary of the Model

- Number of User Stories
- Work Days per Month (WD)
- Monthly Staff Pay
- Number of Days inside a Sprint (Sprint Time)
- Units of Effort Done by the Team in a Sprint
- Estimator's Estimation Confidence (CL).
- Metric for Story Size (Table 1)
- Metric for Story Complexity (Table 2)
- Metric for Friction Factor (Table 3)
- Metric for Variable Factors (Table 4)

5 Evaluation

Completion time (T) is estimated as:

$$T = \frac{\sum_{x=1}^{n}(ES)_x}{(Vx)^D} \times \frac{1}{WD}\,days$$

5.1 Case Study

The total number of user stories is 54.

Sprint Size = 14 Days Team Velocity = 52.

No of Working days per Month = 20.

Monthly Team Salary = 450000.

85 percent assurance level in estimates.

5.2 Outcomes

EFFORT = 320 SP.

INITIL VELOCITY = 4.7

Table 5. Development Cost are computed using the Net Ratio.

P.No	Effort	Vi	D	V	Size of sprint	Working days	Team's Salary	Act: Time	Est time	Real Cost	Calculated cost	Time MRE	Cost MRE
1	145	3.1	.576	2.6	10	22	240000	54	67	1300000	1134105.13	6.84	13.84
2	211	4.5	.612	3.4	10	21	250000	80	92	1500000	1571554.8	12.84	4.13
3	154	5	.787	2.2	10	22	240000	67	43	1100000	781058.42	6.23	1.88
4	212	3.4	.775	2.7	10	22	200000	75	78	2200000	3013676.1	1.25	5.54
5	135	3.8	.814	3.1	10	22	400000	29	38	640000	565172.21	8.257	8.73
6	248	5.2	.814	4.5	10	22	300000	72	84	3100000	278400.74	3.28	8.43
7	98	3.1	.848	2.3	10	22	160000	46	28	500000	451022.75	18.13	8.97
8	146	2.7	.724	4	10	22	270000	82	93	2700000	1523187.3	8.56	10.23
9	75	4.8	.857	2.3	10	22	290000	45	48	600000	416175.47	3.78	2.15
10	123	4.7	.847	2.1	10	22	150000	73	45	2100000	1356288.4	5.36	4.48
11	254	5.7	.747	4.0	10	22	340000	36	39	700000	678821.11	7.67	1.76
12	143	4.8	.754	3.8	10	22	220000	28	29	540000	486251.52	6.32	7.24
13	112	4.8	.864	2.7	10	22	210000	41	46	500000	547583.57	9.264	11.16
14	85	2.7	.882	1.5	10	22	100000	41	35	300000	383636.54	14.12	2.45
15	73	3.6	.854	1.8	10	22	100000	35	33	240000	321452.12	5.67	4.46
16	278	6	.921	3.7	10	22	230000	12	12	3000000	1782574.3	9.12	1.52
17	201	5	.751	2.7	10	22	280000	48	51	700000	881740.21	3.45	3.56
18	152	7	.651	2.6	10	22	240000	43	42	2000000	852765.53	3.73	4.70
19	124	3	.701	1.5	10	22	120000	71	65	1400000	1544111.18	4	2.42

(*continued*)

Table 5. (*continued*)

P.No	Effort	Vi	D	V	Size of sprint	Working days	Team's Salary	Act: Time	Est time	Real Cost	Calculated cost	Time MRE	Cost MRE
20	245	2.4	.607	3.8	10	22	320000	65	42	700000	763438.54	7.81	5.68

FRICTION FACTOR (FR) = 0.612413.
DYNAMIC FORCES = 0.87658.
DECELRATION = 0.531456.
VELOCITY = 2.4
TIME = 5.2 MONTHS.
COST = 5152552.18.
$Time_{Prob}$ = 5.1 MONTHS.
$Time_{Optim}$ = 5.4 MONTHS.
$Time_{Pessi}$ = 6.8 MONTHS.
$Cost_{Prob}$ = 5132782.18.
$Cost_{Optim}$ = 4628615.25.
$Cost_{Pessi}$ = 5674727.31.

6 Conclusion

A software effort estimation modeling for Agile Software projects is discussed in this work. The model's prediction is based on User Stories. The concept is designed to meet most of the features of agile methodology, particularly updated versions and iteration, with the aim to address the major issues faced by agilests. We have designed this method for the accurate estimation based on 'developer's expertise and experience of working and skills to predict the accurate estimation of the effort to done a user story. We have revealed that the estimation of the user stories are almost accurate as per the method suggested in this paper. Due to the biased nature of the developers and different levels of expertise, the estimation needs a significant method.

References

1. Popli, R., Chauhan, N.: Sprint-point based estimation in scrum In: Proceedings of IEEE Conference, GLA University, Mathura, 9–10 March 2013
2. Bhalereo, S., Ingle, M.: Incorporating vital factors in agile estimation through algorithmic methods Int. J. Comput. Sci. Appl. Technomath. Res. Foundat. **6**(1) 85–97 (2009)
3. Misra, S., Omorodion, F.M., Damasevicius, R.: Metrics for measuring progress and productivity in agile software development. In: Abraham, A., Sasaki, H., Rios, R., Gandhi, N., Singh, U., Ma, K. (eds.) IBICA 2020. AISC, vol. 1372, pp. 469–478. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-73603-3_44
4. Attarzadeh, I., Hock Ow, S.: Software development effort estimation based on a new fuzzy logic model. Int. J. Comput. Theory Eng. **1**, 1793–8201 (2009)
5. Butt, S.A., Misra, S., Anjum, M.W., Hassan, S.A.: Agile project development issues during COVID-19. In: International Conference on Lean and Agile Software Development, pp. 59–70. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-67084-9_4

6. Misra, S.: Pair programming: an empirical investigation in an agile software development environment. In: Przybyłek, A., Miler, J., Poth, A., Riel, A. (eds.) LASD 2021. LNBIP, vol. 408, pp. 195–199. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-67084-9_13
7. Abioye, T.E., Arogundade, O.T., Misra, S., Akinwale, A.T., Adeniran, O.J.: Toward ontology-based risk management framework for software projects: an empirical study. J. Softw. Evol. Process **32**(12), e2269 (2020)
8. Rimal, Y., Pandit, P., Gocchait, S., Butt, S.A., Obaid, A.J.: Hyperparameter determines the best learning curve on single, multi-layer and deep neural network of student grade prediction of Pokhara University Nepal. J. Phys. Conf. Ser. **1804**(1), 012054 (2021). IOP Publishing
9. Butt, S.A., Abbas, S.A., Ahsan, M.: Software development life cycle & software quality measuring types. Asian J. Math. Comput. Res **11**(2), 112–122 (2016)
10. Przybylek, A., Kowalski, W.: Utilizing online collaborative games to facilitate agile software development. In: 2018 Federated Conference on Computer Science and Information Systems (FedCSIS), pp. 811–815. IEEE, September 2018
11. Butt, S.A., Gochhait, S., Andleeb, S., Adeel, M.: Games features for health disciplines for patient learning as entertainment. In: Digital Entertainment, pp. 65–86. Palgrave Macmillan, Singapore (2021).
12. Przybyłek, A., Kotecka, D.: Making agile retrospectives more awesome. In: 2017 Federated Conference on Computer Science and Information Systems (FedCSIS), pp. 1211–1216. IEEE, September 2017
13. Behera, R.K., Jena, M., Rath, S.K., Misra, S.: Co-LSTM: Convolutional LSTM model for sentiment analysis in social big data. Inf. Process. Manage. **58**(1), 102435 (2021)
14. Kumari, A., Behera, R.K., Sahoo, K.S., Nayyar, A., Kumar Luhach, A., Prakash Sahoo, S.: Supervised link prediction using structured-based feature extraction in social network. Concurrency Comput. Pract. Exp. e5839 (2020)
15. Anusuya, V., Gomathi, V.: An efficient technique for disease prediction by using enhanced machine learning algorithms for categorical medical dataset. Inf. Technol. Control **50**(1), 102–122 (2021)
16. Behera, R.K., Shukla, S., Rath, S.K., Misra, S.: Software reliability assessment using machine learning technique. In: International Conference on Computational Science and Its Applications, pp. 403–411. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-95174-4_32
17. Arogundade, O.T., Atasie, C. Misra, S., Sakpere, A.B., Abayomi-Alli, O.O., Adesemowo K.A.: Improved predictive system for soil test fertility performance using fuzzy rule approach. In: Soft Computing and Its Engineering Applications: Second International Conference, IcSoftComp 2020, Changa, Anand, India, 11–12 December 2020, Proceedings, vol. 1374, p. 249. Springer, Cham (2021). https://doi.org/10.1007/978-981-16-0708-0_21
18. Butt, S.A.: Study of agile methodology with the cloud. Pacific Sci. Rev. B Human. Soc. Sci. **2**(1), 22–28 (2016)