Received 20 June 2022, accepted 2 August 2022, date of publication 5 August 2022, date of current version 11 August 2022.
Digital Object Identifier 10.1109/ACCESS.2022.3196923

RESEARCH ARTICLE

Estimating Efforts for Various Activities in Agile Software Development: An Empirical Study

LAN CAO
Department of Information Technology and Decision Sciences, Old Dominion University, Norfolk, VA 23529, USA
e-mail: lcao@odu.edu

ABSTRACT Effort estimation is an important practice in agile software development. The agile community believes that developers' estimates get more accurate over time due to the cumulative effect of learning from short and frequent feedback. However, there is no empirical evidence of an improvement in estimation accuracy over time, nor have prior studies examined effort estimation in different development activities, which are associated with substantial costs. This study fills the knowledge gap in the field of software estimation in agile software development by investigating estimations across time and different development activities based on data collected from a large agile project. This study investigated effort estimation in various development activities, including feature development, bug fixing, and refactoring in agile software development. The results indicate that estimation of agile development does not improve over time, as claimed in the literature. Our data also indicate that no difference exists in the magnitude of estimation errors between feature tasks and bug-fixing/refactoring tasks, while bug-fixing and refactoring tasks are overestimated more frequently than feature tasks. This study also contributes to our knowledge about overestimation and underestimation patterns in agile software development.

INDEX TERMS Agile software effort estimation, bug-fixing effort estimation, refactoring effort estimation.

I. INTRODUCTION

The estimation of development effort is a critical activity in software development [1], [2]. Effort estimates guide the planning, budgeting, resource allocation, and monitoring of software development processes [3]. Inaccurate effort estimation may lead to various development problems and even cause project failure [4], [5]. Underestimating development efforts leads to schedule pressure [6] and project overruns [4], while overestimating leads to wasted resources [7].

Effort estimation in agile software development has received significant research attention since 2014 [8]–[11]. This is an important practice in agile planning, enabling the software development team to approximate task sizes and allocate workloads for an iteration [12]. Estimations are used to calculate an iteration's team productivity, helping the team create a reliable iteration plan to deliver the product increment.

The associate editor coordinating the review of this manuscript and approving it for publication was Laxmisha Rai.

While some studies focus on improving estimation-learning processes through formal models, such as machine learning [2], [3], [15], expert-based methods are the dominant approach used in agile software development [9]. The agile community believes that the agile development process facilitates developers' learning from short and frequent feedback loops [16]. Developers continuously adjust their estimation based on feedback from previous iterations [17]. As a result, the estimates become more accurate over time due to the cumulative effect of learning [10], [32], [33]. However, despite the vast number of studies on agile software effort estimation, there is a lack of empirical evidence on the improvement of estimation accuracy over time in agile development. Studies on effort estimation of software development, on the other hand, report that experts' previous experience in estimation does not lead to better judgment on similar tasks [4], [18], [35], [36]. As most of these studies are based on lab experiments with a small number of estimation tasks, there is a call for research to collect more data [19] to investigate estimation-learning processes. Examining whether learning happens over a longer period of time with a large amount of data from a real agile project would provide a nuanced understanding of the estimation process.

Bug fixing and refactoring are associated with substantial costs [6], [38] and estimating the efforts of these activities is critical for agile software development. While much emphasis has been placed on estimating general development efforts [8], estimating efforts in bug fixing and refactoring remain underexamined [20], [21]. Extant studies on bug fixing have focused on the development of prediction models that support a cost-benefit analysis of bug fixes. For example, one study [21] analyzed factors that affect the effort needed to fix a bug and the ability to predict the fix implementation effort using data from a NASA mission. To the best of our knowledge, no empirical studies have been conducted to investigate bug fixes or the estimation accuracy of refactoring tasks. One reason might be the difficulty in obtaining data on such activities. Even though most software development projects use some sort of bug-tracking system, the effort-estimation data is usually not well maintained. In many cases, information about bug fixes is hidden in the version control system that records changes made to the source code. The system does not identify whether a change involves bug fixing, refactoring, or implementing a new feature(s).

These gaps in effort estimation in agile software development motivated this study, which aims to answer the following research questions: 1) *Do effort estimations in agile software development become more accurate over time?* 2) *Are bug-fixing and refactoring effort estimations less accurate than feature development effort estimations?*

To answer these questions, we conducted a field study using data collected from a large agile software development project. The remainder of this paper is organized as follows: Section 2 describes the previous research on agile software effort estimation and the measures used in this study; Section 3 presents the hypotheses; Section 4 describes the research design; and the results are presented in Section 5. Section 6 provides a discussion of the findings, and Sections 7 and 8 discuss the study's limitations and conclusions, respectively.

II. RELATED WORKS

A. EFFORT ESTIMATION IN AGILE SOFTWARE DEVELOPMENT

Software effort estimation methods can be categorized as model-based, which are derived from prediction models; expert-based, which rely on human judgment; and hybrid methods that combine model-based and expert-based methods [22]–[25]. In agile software development, expert-based estimation techniques such as planning poker, expert judgment, and story points are the major strategies used [9], [26], while new techniques based on the intensive use of data and algorithms, such as machine learning, have been proposed [27].

Early research on effort estimation has focused on predicting the effort required to complete a software project that follows a traditional plan-driven development approach [2]. Traditional software development usually makes estimates only at the beginning of a project based on the features identified in the requirement specifications [28]. However, agile software development does not have an upfront requirement analysis phase, with features emerging and evolving throughout the entire development process. Software is developed through repeated small cycles, allowing for adaptation to changing requirements at any point during development.

A distinct characteristic of agile development is its multiple levels of planning and short feedback loop [12]. Agile planning generally includes *release planning* and *iteration planning*. Release planning includes identifying features that will be developed for a release and estimating the effort needed to develop features. Iteration planning defines a tactical development plan for an iteration. Selected features for the iteration are divided into stories, each of which is discussed, covering relative difficulties, size, complexities, uncertainties, technical challenges, and acceptance criteria [7]. Effort estimation at the iteration level involves estimating the effort needed to complete a backlog item, such as a user story. The iteration plans contain other types of items, such as bug fixes, and these are sometimes estimated as well. Estimates are adjusted continuously based on feedback throughout the development process [12].

Stories are typically subdivided into smaller units, termed tasks, during the iteration planning process. Developers then discuss approximately how long a task will take and its dependencies on other tasks or stories. Most prior studies on estimation in agile software development have been conducted using estimation data at the story level [9]. While some lab experiments have investigated estimation accuracy at the task level [9], to the best of our knowledge, no study has examined effort estimation at the task level using real project data.

Prior studies on effort estimation in agile software development have been summarized in several systematic review papers [9], [26], [29], [30]. These reviews investigated techniques used for effort or size estimation in agile software development, effort predictors used, characteristics of the data set used, and estimation accuracy. According to the reviews, these studies pursued very similar research questions, while the knowledge of effort estimation in agile software development remains dispersed [9]. The estimation accuracies in these studies are inconsistent and poorly reported, with many studies reporting inadequate accuracy results, making comparisons unreliable. These reviews point out the lack of empirical knowledge on effort estimation in agile software development [29], [30]. The data used in the empirical studies are limited for several projects, as well as estimation levels. Usman *et al.* [26] reports that most of the techniques have not attained acceptable prediction accuracy values (e.g., MRE $< 25\%$) regarding how close the estimated values are to actual values. The authors conclude that practitioners would find little guidance from the current literature on effort estimation in agile software development.

Moreover, the development activities involved are usually not made explicit in prior studies [9]. No comparisons between different development activities (e.g., feature development, refactoring, or bug fixing) regarding estimation accuracy have been conducted. This study aims to address these gaps by investigating effort estimation for feature development, bug fixing, and refactoring using extensive task-level data collected from a large agile project. Table 1 summarizes the differences between prior studies and this study.

TABLE 1. comparison of prior studies and this study.

	Prior studies	This study
Focus	Estimation accuracy [9, 26, 29, 30] Cost drivers [29]	Improvement of accuracy overtime; patterns of underestimation vs overestimation
Estimation Techniques	Planning poker, story points, expert judgment [9, 26] machine learning [27]	Expert judgment
Estimation Levels	Release/iteration [9]	Task level and aggregated story/ iteration level
Datasets	Lab experiments, Industry data (most studies are limited) [9]	Real project, large amount of data
Development Activities	Not explicit, general development [9, 26, 29, 30]	Feature development, bug fixing, refactoring

B. MEASURES

Myriad estimation accuracy measures have been used in the software engineering literature, none of which has been endorsed across the whole community. To reduce concerns over measurement bias, this study uses multiple measures, including Magnitude of Relative Error (MRE), Relative Error (RE), Balanced Relative Error (BRE), and Balanced Relative Error Bias (BREbias). MRE and BRE are used to measure estimation accuracy, and RE and BREbias are used to measure estimation error toward optimism (underestimation) or pessimism (overestimation). A positive RE/BREbias indicates a tendency toward underestimation, a negative RE indicates a tendency toward overestimation, and a zero RE means unbiased estimates.

Magnitude of Relative Error (MRE) = |actual effort - estimated effort|/actual effort

Relative Error (RE) = (actual effort - estimated effort)/actual effort

Balanced Relative Error (BRE) = |actual effort - estimated effort|/min (actual effort, estimated effort)

BREbias = (actual effort - estimated effort)/min (actual effort, estimated effort)

Prior research has pointed out the limitations of MRE (and RE) [31]. One concern is that MRE rewards underestimation [32], [33], with much less weight placed on underestimation than on overestimation [34]. The insufficient weight on underestimation can be illustrated by an example that entails two stories. Stories 1 and 2 have the same estimated effort: to be 10 ideal hours. Story 1 has been overestimated, with the actual effort at five hours, while Story 2 has been underestimated, with the actual effort at 20 hours. The MRE overestimation measure of Story 1 is $|5 - 10|/5 = 1.0$, while the MRE underestimation of Story 2 is $|20 - 10|/20 = 0.5$. This uneven weighting may be misleading because with practice estimation, performance is often rated relative to the estimated effort in software development [34]. To address this issue, this study also used more balanced accuracy measures, as prior studies have suggested [34]: BRE and BREbias. BRE evenly balances overestimation and underestimation. For the previous example, the BREs for both Stories 1 and 2 are 1.0.

To examine whether the estimation accuracy is acceptable, this study reports prediction-level PRED (25), which measures the percentage of MRE values within 25 percent of the total number of tasks. PRED (25) $\geq$ 75% is generally viewed as acceptable [9].

A concern exists that MRE and RE outliers may impact mean MRE and mean RE strongly [33]. Following a prior study's [4] suggestions, we also report more outlier-robust MRE and RE medians.

III. RESEARCH HYPOTHESES

Agile planning uses the analogy of "yesterday's weather" (i.e., today's weather can be predicted to be the same as yesterday's) to estimate productivity [17], [35]. Data from the preceding iterations are used to predict how much work can be finished in the coming iteration with adjustments. Even though this concept concerns the estimation of team capacity, it implies that developers learn from previous experiences and improve their efforts at estimation as a project progresses. Agile software development provides a learning environment in which developers frequently adjust their actions[16] and continuously self-correct [17]. This continuous adjustment quickly improves the estimates [12]. It is believed that uncertainty decreases significantly as developers obtain more knowledge about the project. As a result, the estimates become very accurate after a few iterations [12], [17], [35].

While the agile development literature has contended that estimation accuracy improves over time, some researchers have put forth different opinions. Jørgensen and Gruschke [4] argue that humans' capabilities for improving their own expert judgment are poor. One study found that experts' ability to learn from prior estimation experience is "disappointingly low" in both software maintenance and development [36]. Jørgensen and Sjøberg [37] conducted a study of 54 software professionals' work on extending and maintaining large administrative applications, finding that the amount of application-specific experience from a previous estimation task does not lead to more accurate assessments of a similar task. Feedback from previous tasks is difficult to apply properly for estimation purposes. Similarly, a study of six software project leaders' estimates over a period of three years found no significant improvement [18]. A few empirical studies have tested the learning effect with different results [7], [38]. To sum up, learning estimation skills from feedback is a difficult task, and empirical studies suggest that, generally, estimation does not improve over time. This inconsistency motivated us to test the following hypothesis:

Hypothesis 1. Estimation accuracy improves over time in agile software development.

Bug fixing is an important activity in agile software development [6], but prior studies on agile software development estimation have not examined effort estimation accuracy in bug fixing [9], [26]. It is believed that estimating the effort for bug fixing is more challenging than estimating the effort for feature development because estimating what is needed to fix a bug involves investigating the issue, looking into the source code, and isolating the problem [20]. Some developers believe that estimating bugs is a waste of team effort, because it may take longer than just fixing the bug. A common practice is to prioritize bugs and assign them to the iteration backlog, while each iteration is allocated a certain amount of time for bug fixing.

Similarly, prior research on refactoring reveals that software developers associate refactoring with substantial costs [39], and they usually cannot estimate the effort required for refactoring [40]. Refactoring effort is difficult to estimate because estimating the effort needed to understand the original code and the effort needed to test and remove bugs introduced by refactoring is difficult. As a result, estimations of bug fixes and refactoring are less accurate than estimations of features. Thus, we propose the following:

Hypothesis 2. Estimation accuracy for bug fixing is lower than for feature development.

Hypothesis 3. Estimation accuracy for refactoring is lower than for feature development.

Prior studies have suggested that cost and schedule overruns are common risks in software projects [2]. For example, Molokken and Jørgensen [41] found that most (60–80%) software projects encounter effort overruns. Another study [5] found that one out of every six projects incurs a cost overrun of 200% and a schedule overrun of almost 70%. Another study reported that, on average, large software projects run 66 percent over budget and 33 percent over time [42]. These findings suggest that software development efforts tend to be underestimated more frequently than overestimated [43], [44].

One reason for the less frequent overestimation might be Parkinson's law, which says that the development team expands work to fill the time available for its completion [45]. Thus, even when a task is estimated to take longer than necessary, the developer may work in a way so that the actual time expended is close to the estimated time and does not report overestimation. Another possible reason for the difference between the frequencies of overestimation and underestimation is that they impact project management differently [46] and may be evaluated differently. For example, underestimations are more "expensive" than overestimations if the team must compensate the customer for potential delayed delivery of the product.

While prior studies suggest that software development efforts tend to be underestimated [43], an interesting review found that estimation is affected by several factors. For example, underestimation is reported more frequently than overestimation in studies from the engineering and management literature, but not in psychology literature [47]. The review also reports the correlation between the magnitude of tasks and frequency of under or overestimation. Larger tasks (e.g., projects in person-months) elicit underestimation, while smaller tasks (e.g., lab studies with task sizes in hours) elicit overestimation or unbiased estimates [47]. Similarly, another study [18] finds that small software development tasks are more likely to be overestimated. Considering that iteration planning of agile software development involves estimating small pieces of tasks [12], we hypothesize that the frequency of overestimation is higher. Thus, we propose:

Hypothesis 4. The frequency of overestimation is higher than the frequency of underestimation in agile iteration planning.

While prior studies have revealed task size, complexity, and difficulty could impact estimation [47], the difference between the magnitude of errors for underestimation and overestimation has not been reported. Thus, we propose the following:

Hypothesis 5. No difference exists between the magnitude of errors for underestimation and overestimation in agile iteration planning.

IV. RESEARCH METHOD

A. CASE DESCRIPTION

To test the hypotheses, we selected a case based on the following criteria: 1) the project uses the agile software development approach, 2) both estimated and actual data are recorded for multiple iterations over a period of time, and 3) estimation data for bug fixing and refactoring are available. A software development company involved in developing a project management system for managing agile software development processes was chosen based on those criteria. The project provides a rare opportunity for this study.

First, the product itself is an agile software project management tool that aims to integrate business and technical teams in a collaborative environment to define, plan, and test software releases. The product includes capabilities for release and iteration planning, scheduling and tracking, project status and team velocity tracking, requirement management, test management, and bugs and issue management. Second, the team uses its own product to manage its development process, with a goal of testing the product's functionalities. With this goal, the team is committed to recording detailed data on the process, including estimated and actual efforts of feature tasks, bug fixes, and refactoring tasks, as well as task assignments. In most organizations, recording these data is viewed as a burden, so it is rarely done. Finally, this project has taken an agile approach, following combined Extreme Programming (XP) and Scrum practices such as daily stand-up meeting, co-located team, on-site customer, pair programming, review meetings, and retrospective meetings. The development process is iterative, and each iteration takes two weeks. This project planned its product releases in two-month cycles, but formally announced new releases only four times a year. The project was on Release Five at the time of data collection. The releases were sequential—a new release was started only when the previous release was finished. New stories were implemented with each release, and a different team handled the maintenance of previous releases. Altogether, 15 developers had been working on the project during its first 34 iterations. For Iterations 35 to 46, some developers were removed from the project, with only five developers remaining (including a new developer who joined the project from Iteration 35).

B. DATA DESCRIPTION

The project data were captured by the team from iterations corresponding to the five releases. As the data from the first release were not complete, data from Releases 2, 3, 3.5, and 4 were used to test the hypotheses. The four releases included data from 46 two-week iterations (more than 21 months). The data included the estimated and actual efforts for feature, bug-fixing, and refactoring tasks; task assignments; task and story priorities; and status (incomplete, viewed as complete by developers/customers).

In addition to analyzing the recorded data, we conducted four interviews about estimation practices and challenges with the product owner, two developers, and the president of the company. The four informants were highly experienced in agile software development. We also participated in one iteration planning meeting to observe the estimation process. The meeting lasted about two hours, and the discussions among team members (product owner, scrum master, and other members) were recorded. The data collected during this meeting were compared with information collected from the interviews for triangulation.

Due to the product's unique nature (agile project management tool), the team was diligent in tracking its processes and recording all the estimated and actual effort data from all activities. During the iteration planning meetings, each story planned for the next iteration was subdivided into tasks and estimated at the task level by the developers. Developers signed up to tasks based on those estimations. At the end of an iteration, the actual effort of each task and whether the task was complete were recorded by the developers.

Estimation for this project was conducted only at the task level. Story-level and iteration-level estimations and actual effort data were aggregated from task-level data. The team did not conduct independent estimations for stories and iterations. The aggregation of data significantly impacts the results from data analysis. Depending on the aggregation level and approach, we can elicit very different results on estimation accuracy for exactly the same data set.

At the story level, when examining aggregations over several tasks, the estimation bias measures (RE and BREbias) do not always reflect estimation quality. When aggregating data from the lower level, the effect of larger volumes dampens random variation's impact. For example, a story contains four tasks with estimated/actual efforts of 1/1, 6/5, 3/2 and 4/7. Even though the MREs for three tasks are quite large (0.2, 0.5, and 0.43), the story-level MRE calculated by comparing actual aggregated story-level effort (14) with story-level estimation (15) is only 0.067. Thus, estimations using aggregated data have lower MREs and REs than the estimations using the disaggregated data.

This study used both disaggregated and aggregated data in the analysis. For example, the story-level MRE was calculated as:

Story-level MRE = |actual story effort – estimated story effort|/actual story effort

actual story efforts $=\sum_{i=1}^{i=n}$ (actualtaskeffort)

estimated story efforts $=\sum_{i=1}^{i=n}$ (estimatedtaskeffort)

with n as the number of tasks in the story.

Similarly, iteration-level MRE was calculated using the aggregation of estimated story efforts and actual story efforts.

At the task level, about 7,000 estimates and actual effort data were collected on features, bug fixes, and refactoring tasks. These data were aggregated into data at the story level, eliciting more than 1,000 units of story-level data, which were examined and cleaned up carefully. First, some data contained missing information. To deal with the missing-data issue, we removed tasks with missing data on either task estimation or actual time. Second, we removed data from tasks marked incomplete. If a task was marked incomplete, the actual effort recorded was the actual effort spent on the task but does not reflect the actual real effort needed to finish the task. Altogether, 1,972 tasks remained after the cleanup, among which 1,282 were feature tasks, 377 were bug fixes, and 313 were refactoring tasks. These data were used for the analysis. Table 2 provides the descriptive statistics of tasks (actual sizes).

Among the 46 iterations, two were dedicated to bug fixes (one was missing all estimation data), 12 were dedicated to refactoring (also termed ''hardening'' iterations), and one contained only actual effort data and lacked any estimation data. As a result, 31 iterations contained both feature and bug-fixing data, including 304 stories that contained 1,282 feature tasks. The story-level and iteration-level data were aggregated from the task-level data, which were also used for the analysis. However, considering that we removed a large portion of incomplete data, the sizes of stories and iterations were smaller than the original stories and iterations. The descriptive statistics on the stories and iterations were not included in Table 2 to avoid misunderstandings.

V. RESULTS

A. HYPOTHESIS 1

To test Hypothesis 1, we examined whether estimation accuracy improved over the 21 months from Iteration 1 to

TABLE 2. **Descriptive statics of feature, bug fixes and refactoring tasks.**

	Feature	Bug	Refactoring
Number of tasks	1282	377	313
Average size (hours)	4.5	4.3	3.2
SD	5.5	5.5	3.7
Median size (hours)	3	2	2

Iteration 46. To compare estimation accuracy across iterations, task estimates were aggregated into iteration-level estimates. The iteration-level MREs over 31 iterations (iterations 14, 21, and 22 do not include complete data or feature data) are provided in Figure 1a.

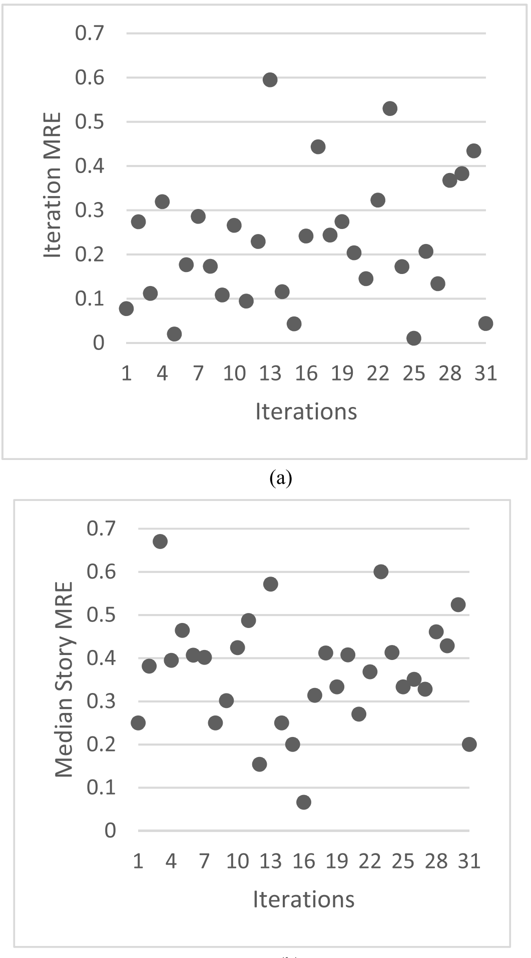
Considering that the aggregation may result in the loss of significant information, we also examined the median MRE and BRE of each iteration's stories. Medians were used to reduce the impact of "outliers." Figures 1b and 1c show the medians of story MRE and BRE for each iteration. These figures indicate that even though iteration-level estimation is more accurate than story-level estimation because of aggregation, the effort estimations on both levels do not improve over time.

To further check whether estimation accuracy improves over time, we grouped the story-level estimation data into three phases: early phase (Iterations 1–9; 106 stories); middle phase (Iterations 10–26; 99 stories); and late phase (Iterations 27–34; 99 stories). A Kruskal-Wallis test on MRE found no significant differences among the three phases. (The p-value is .0599, and the H statistic is 5.63 [2, N = 304].) Further comparisons of BRE, RE, and BREbias between the three groups (early/middle, middle/late, and early/late) show no significant differences either (Table 3). Thus, estimation accuracy does not improve over time (i.e., H1 is not supported).

TABLE 3. **Estimation accuracy on early, middle, and late phases of development.**

	P	H (2, 304)
MRE	0.1058	4.49
BRE	0.0625	5.50
RE	0.8086	0.42
BREbias	0.8375	0.35

While the data show that the estimation accuracy does not improve overtime, our interviews with the developers elicit surprising findings on their overconfidence and the gap between reality and developers' perceptions about learning. Overconfidence in the accuracy of effort estimates' typically means that the developers have made unrealistic assessments of the estimations' underlying uncertainties [4]. Many developers believe that they have learned along the way and that the experiences they have gained improve their estimates. One developer emphasized the role of experience: *"Experience is the biggest factor.... As we had been working on this project for a while, and we had more agile experience and knew the code base very well, we got very accurate [on estimation]."*



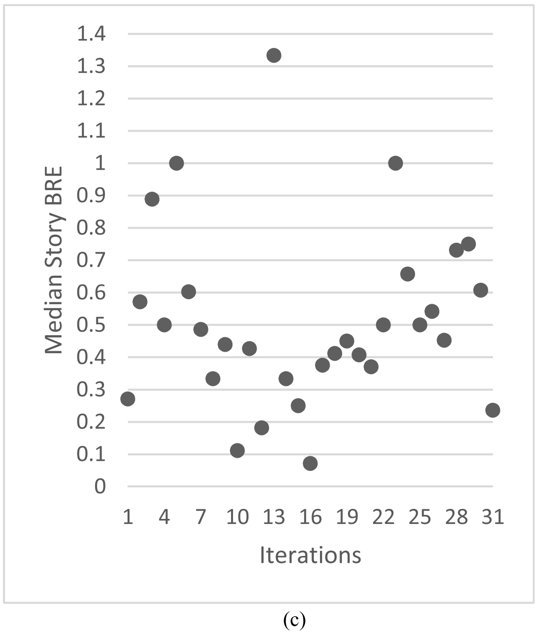


FIGURE 1. **a. Iteration feature MRE overtime. b. Median MRE of feature story overtime. c. Median BRE of feature story overtime.**

Another developer expressed a similar view: *"After several iterations into the project, there was good understanding of what could be accomplished. Experience with Scrum and XP improved accuracy as time went by.... It took several iterations to get good accuracy."*

When asked about estimation challenges, developers pointed out that the task's unknowns make prior experience irrelevant. As the developer explains: *"How long will it take to implement the features? Sometimes I don't want to answer this question because it is extremely difficult to estimate how long it will take to implement a software feature. The problem is that there are far too many unknowns and [I] have no way to extrapolate from previous experience. [For]) some features, I can provide a number [of hours for implementing a feature], and [for others], I have no idea."*

B. HYPOTHESIS 2

Next, we compared bug fixing and feature development's estimation accuracies. The comparison was conducted at the task level because most of the bug data were recorded only at the task level. The data on bug fixes contained more missing estimations (many were not estimated, and only the actual effort data were available) and are scattered across iterations. Altogether, 377 bug-fixing tasks were used for the analysis.

As the data are not normally distributed, we used the Mann-Whitney U test (Wilcoxon signed-rank test), which is more robust than the t-test because it compares the sums of ranks. Surprisingly, the results indicated no significant difference between estimation accuracy in the bug fixes and feature tasks. There were 377 bug-fixing tasks MRE (mean = 0.70, median = 0.45, SD = 1.41) compared with 1,282 feature tasks MRE (mean = 0.63, median = 0.40, SD = 1.46), indicating no significant difference. The z-score was −1.21, and the p-value was 0.23. Likewise, further comparisons of BRE, RE and BREbias between feature tasks and bug-fixing tasks found no significant difference (Table 4). Thus, the estimation accuracy for bug fixing is not lower than feature development (i.e., H2 is not supported).

TABLE 4. **Feature and bug fix estimation accuracy (task level).**

	Mean		Median		p	Z score	Effect size
	Ft	Bt	Ft	Bt			
MRE	0.63	0.70	0.40	0.45	0.23	-1.21	0.03
BRE	0.93	1.00	0.50	0.50	0.40	-0.85	0.02
RE	-0.29	-0.38	0.00	0.00	0.21	1.24	0.03
BREbias	-0.01	-0.09	0	0	0.20	2.29	0.03

Ft = feature task (n = 1282), Bt = bug fix task (n = 377)

C. HYPOTHESIS 3

We then compared the estimation accuracies in refactoring and feature development. The continuous refactoring efforts were not recorded, and instead, the project recorded 84 refactoring stories (313 refactoring tasks) in 12 iterations dedicated to refactoring.

At the task level, the Mann-Whitney U test found no significant difference in MRE between the 313 refactoring tasks (mean = 0.43, median = 0.43, SD = 1.53) and the 1,282 feature tasks (mean = 0.63, median = 0.40, SD = 1.46). The z-score was -0.91, and the p-value was 0.45. Moreover, BREs were not significantly different, but RE and BREbias were significantly different between feature tasks and refactoring tasks (i.e., the directions of estimation bias were opposite, and refactoring was more overestimated). However, the effect sizes were very small. A summary of all measures' results is presented in Table 5.

TABLE 5. **Feature and refactor estimation accuracy (task level).**

	Mean		Median		p	Z score	Effect size
	Ft	Rt	Ft	Rt			
MRE	0.63	0.83	0.40	0.43	0.36	-0.91	0.023
BRE	0.93	1.02	0.50	0.50	0.92	-0.10	0.0025
RE	-0.29	-0.59	0	0	0.003	3.00	0.075
BREbias	0.01	-0.40	0	0	0.003	3.00	0.075

Ft = feature task (n = 1282), Rt = refactor task (n = 313)

Considering that refactoring tasks were grouped as stories, we also ran the Mann-Whitney test at the story level, and the results were the same as the task-level results. No significant difference in MRE (or BRE) was found between the 84 refactoring stories (mean = 0.71, median = 0.32, SD = 1.58) and the 304 feature stories (mean = 0.48, median = 0.34, SD = 0.63). The z-score was −0.81, and the p-value was 0.42. The RE and BREbias were significantly different, indicating that refactoring was more overestimated. The effect size was 0.16 (small). A summary of the results was presented in Table 6. Thus, the estimation accuracy for refactoring was not lower than for feature development (i.e., H3 is not supported).

While the data showed that estimation accuracies for bug fixings and refactoring were not lower than for feature development, our interviews found that developers often perceive estimations on feature tasks more accurately: "*We should spend more time on estimates of bugs. Our estimates of features are more accurate. The team in general had a pretty good feel for the estimate [of features], but not for the bugs, particularly bugs that are not able to be reproduced, or those that would require a lot of effort to trace."*

TABLE 6. **Feature and refactor estimation accuracy (story level).**

	Mean		Median		p	Z score	Effect size
	Fs	Rs	Fs	Rs			
MRE	0.48	0.71	0.34	0.32	0.42	-0.81	0.04
BRE	0.75	0.87	0.48	0.33	0.20	1.29	0.07
RE	-0.05	-0.47	0.11	-0.02	0.0012	3.24	0.16
BREbias	0.23	-0.31	0.13	-0.02	0.0012	3.24	0.16

Fs = feature story (n = 304), Rs = refactor story (n = 84)

D. HYPOTHESIS 4

We then compared the frequency of overestimates and underestimates in all development activities (feature, bug fixing, and refactoring).

1) FEATURE DEVELOPMENT

At the task level, the frequencies of underestimates and overestimates of feature tasks are almost the same. Among the

1,282 feature tasks, 456 were overestimated, while 453 were underestimated (373 had zero RE; i.e., no bias). Next, story-level data were compared. Considering that the story data were aggregated from task-level data, we expected the story-level to reveal no difference between overestimate and underestimate frequency. However, the story-level data indicated that underestimates were more frequent (171 stories) than overestimates (100 stories). Through a careful examination of the data, we found that one reason for more frequent underestimates at the story level was "scope creep," in which some tasks that are not initially identified emerge during the implementation of the story. These tasks are marked as "non-initial tasks" (i.e., they are not initially identified in the iteration planning). As a result, the story-level estimates did not include estimates of these tasks. The estimated and actual efforts of the newly emerged tasks were recorded after they were identified and implemented. As a result, a story's actual effort included both the initially identified tasks and newly emerged tasks, while a story's estimated effort did not include new tasks. This gap causes more underestimation at the story level; thus, underestimation was more frequent than overestimation at the story level for feature development.

2) BUG FIXING

The frequencies of underestimation and overestimation of bug fixes were examined only at the task level. Among the 377 bug-fixing tasks, overestimation (151) was more frequent than underestimation (125), with 101 bug-fixing tasks estimated accurately (RE = 0). Thus, overestimation was more frequent than underestimation at the task level for bug fixing.

3) REFACTORING

The frequencies of underestimation and overestimation of refactoring were compared at both the task and story levels. Among the 313 refactoring tasks, overestimation (129) was more frequent than underestimation (91), with 93 refactoring tasks estimated accurately (RE = 0). At the story level, among the 84 refactoring stories, overestimation (42) was more frequent than underestimation (31), with 11 stories estimated accurately. Thus, overestimation was more frequent than underestimation for refactoring.

To sum up, H4 was partially supported. Bug fixing and refactoring were overestimated more frequently. For feature tasks, the chances of underestimated or overestimated were the same at the task level. At the story-level, features were underestimated more frequently due to scope creep. This explains why large tasks were underestimated more than small tasks (Table 7).

E. HYPOTHESIS 5

Finally, we compared the magnitudes of error for overestimation and underestimation in all development activities (feature development, bug fixing, and refactoring). Considering that this comparison is sensitive to the balance between the weight on underestimation and overestimation, we used BRE because MRE puts insufficient weight on underestimation.

TABLE 7. Frequency of overestimation and underestimation.

	Number of underestimations	Number of overestimations
Feature tasks	453	456
Feature stories	171	100
Bug fix tasks	125	151
Refactoring tasks	91	129
Refactoring stories	31	42

1) FEATURE DEVELOPMENT

At the task level, the Mann-Whitney U test between the BRE of the 456 overestimates (mean = − 1.29, median = −1.00, SD = 1.42) and 453 underestimates (mean = 1.32, median = 1.00, SD = 1.64) found no significant difference (p = 0.44, z-score = 0.77). The BRE at the story level was similar to that of the task level, with the 100 overestimate stories (mean = -0.81, median = − 0.50, SD = 1.00) and 171 underestimate stories (mean = 0.87, median = 0.50, SD = 0.99) indicating no significant difference (p = 0.50, z = 0.68). Thus, no difference existed between the magnitude of errors for underestimation and overestimation for feature development.

2) BUG FIXING

The Mann-Whitney U test found no significant difference (p = 0.68, z-score = 0.42) between the BRE of overestimated bug-fixing tasks (mean = −1.36, median = −1.00, SD = 1.46) and underestimated bug-fixing tasks (mean = 1.37, median = 1.00, SD = 1.55). Furthermore, no difference was found between the magnitude of errors for underestimation and overestimation for bug fixing.

3) REFACTORING

At the task level, the Mann-Whitney U test found a significant difference (p = 0.001, z-score = 3.25, effect size = 0.22) between the BRE of overestimated refactoring tasks (mean = -1,72, median = −1.00, SD = 2.06) and underestimated refactoring tasks (mean = 1.07, median = 0.75, SD = 1.14). However, at the story level, the Mann-Whitney U test found no significant difference between the BRE of overestimated refactoring stories (mean = −1.19, median = −0.45, SD = 1.99) and underestimated refactoring stories (mean= 0.76, median = 0.5, SD = 1.02). The p value was 0.73, and the z-score was 0.35. Thus, the magnitude of error for underestimation and overestimation for refactoring was significantly different at the task level, but not at the story level.

To sum up, for feature development and bug fixing, the magnitude of estimation error for the overestimated and underestimated tasks/stories was not significantly different. Refactoring indicated different patterns: At the task level, the overestimated refactoring tasks had a higher degree of estimation error, while no difference existed at the story level (Table 8). Thus, H5 was partially supported. Table 9 provides a summary of the results of all hypotheses.

TABLE 8. **Bre of overestimation and underestimation.**

	Mean		Median		p	Z score
	underest	overest	underest	overest		
Ft	1.32	-1.29	1.00	-1.00	0.44	0.77
Fs	0.87	-0.81	0.50	-1.00	0.50	-0.68
Bt	1.37	-1.36	1.00	-1.00	0.68	0.42
Rt	1.07	-1.72	0.75	-1.00	0.001	3.25
Rs	0.76	-1.19	0.5	-0.45	0.73	0.35

Ft = feature task, Fs = feature story, Bt = bug fix task, Rt = refactoring task, Rs = refactoring story

TABLE 9. **Summary of results.**

H1	Not supported Estimation accuracy does not improve overtime
H2	Not supported No difference exists between estimation accuracy for bug-fixing and feature development.
H3	Not supported No difference exists between estimation accuracy for refactoring and feature development
H4 .	Partially supported The frequency of overestimation is higher than the frequency of underestimation for bug-fixing, refactoring, and story-level feature tasks (No difference for task-level feature task)
H5	Partially supported No difference exists between the magnitudes of errors for underestimation and overestimation for feature development, bug-fixing, and story-level refactoring (higher magnitude of errors for the task-level refactoring).

VI. DISCUSSION

This study contributes to the research on effort estimation in agile software development in multiple ways. First, it examines the long-held assumption that agile software development facilitates developers' learning from short and frequent feedback loops [16]. However, this study's results challenge this assumption. Our data indicate that the estimation accuracy in agile software development does not improve over time, even though timely feedback is provided for each estimation. This finding corresponds with a previous study's results on non-agile projects [48], which found that the uncertainty range was nearly identical throughout the project, contradicting the conventional view of estimation getting better as a project progresses.

This study supports the findings of a study on the lack of learning from lessons-learned sessions in software development [4]. In both software development and other contexts, prior experience is unrelated to expert judgments' [49] accuracy due to the lack of proper thinking and learning models. One reason for learning failure in software development is the difficulty of transferring experience from one context to another [50]. In situations with high uncertainty and unstable task relations, feedback is not sufficient for learning [51]. Some researchers [49] have suggested that outcome feedback may even be detrimental to performance. While estimation accuracy in agile development is not improved through learning from experience, this implies that other factors' impact on the estimation process is significant. Thus, future research is needed to examine other factors, such as learning structures and formal training in estimations.

While this study's goal does not include comparing estimation accuracy with that of other studies, our data reveal some insights into effort estimation accuracy at the task level, including that accuracy levels differ from developers' perceptions (e.g., many developers believe that their estimates are accurate with less than 5% error) [44]. A recent review [9] reports that the MREs of studies using expert-based estimation methods are 12% to 33%, but both the MMREs and PRED (25) of feature tasks, bug fixing, and refactors in this study are much higher (Table 10).

TABLE 10. **Accuracy of effort estimation (task level).**

	Feature Task	Bug Task	Refactoring Task
N	1282	377	313
MMRE	0.63	0.70	0.83
PRED (25)	0.37	0.35	0.41

Prior research has suggested reasons for inaccurate estimation, including project management-related issues, team-related issues, and task-related issues [44]. However, many of these issues are not present in the focal project. For example, during the 21-month development period, the team was quite stable, with a low turnover rate. While some members were more experienced than others, all members were competent and helped each other. This project's uniqueness comes from the fact that the product was an agile tracking tool, so the teams were committed to following agile practices. They also diligently tracked their estimation and actual efforts, while on most other agile teams, extensive tracking might be viewed as work overload. The agile community believes that estimation is easier for small tasks, but our data indicate that estimates for very small tasks are not in an acceptable accuracy range (e.g., with PRED [25] $> 75\%$).

This aligns with the recent discussion on the value of estimation in agile software development. A #NoEstimate movement has been evolving since 2012, with proponents arguing that estimates of backlog items do not directly add value to the development process and merely put more pressure on teams. This movement calls for developers to reduce the estimation process or even eliminate it if possible [52]. One alternative would be to use the average size of user stories from historical data for planning.

The second contribution from this study concerns the insights revealed on the estimation of various development activities. The study, to the best of our knowledge, is the first to examine various development activities' estimation accuracy in agile software development with empirical data—specifically feature development, bug fixing, and refactoring. Our analysis indicates that no difference exists in estimation errors between feature tasks and bug/refactoring tasks at the iteration planning level. This is quite surprising, because people usually think that bug fixes and refactoring tasks involve more uncertainties, as they require both software comprehension and modification [53].

This finding may be attributed to the early identification and prompt fixing of bugs in agile software development. Bugs detected during a sprint are usually analyzed immediately, and a course of action is determined. One course of action is to try to fix the bug within the current sprint; otherwise, a work item related to the affected user stories is created and placed in the product backlog, where it is scheduled to be completed during the next sprint or any other future sprint [6].

In agile software development, refactoring is an important activity that is performed continuously to improve software design and quality. Our focal project did does not record the continuous refactoring effort. The data from 12 centralized refactoring iterations indicate that, compared with feature development, refactoring is overestimated more frequently. One possible reason is technical debt, which is a metaphor used to discuss the consequences of sub-optimal design decisions taken when short-term goals are prioritized [54]. These refactoring tasks are from dedicated iterations (e.g., before a release) that have already had the technical debt when performed. Realizing the technical debt, developers may add extra buffers for the debt interests when they estimate the needed effort.

This study also contributes to our knowledge about overestimation and underestimation patterns in agile software development. Empirical studies on software development frequently report effort and schedule overruns [5], [41]. A survey of agile effort estimation studies [43], [44] reported that the tendency to underestimate effort is greater than the tendency to overestimate. However, our data indicate that estimation tendencies are different for different levels of data aggregation and for different development activities (feature development, bug fixing, and refactoring). For example, the story-level estimates of feature development have more underestimates than overestimates due to scope creep, but at the task level, no difference exists between underestimation and overestimation frequency. This explains why large tasks are underestimated more often than small tasks. Another interesting finding is that bug fixes (task level) and refactoring (both task and story levels) are overestimated more frequently. A potential explanation is that developers add more buffers for software comprehension [53] before making changes to the source code.

VII. LIMITATIONS AND FUTURE RESEARCH

We identified three limitations in this study. First, our data were collected from a single project with 15 developers, which may limit our results' generalizability. Second. the number of variables included in this study was limited. Data on factors that may influence estimation changes) were not available. Third, our data were recorded only at the task level.

Future research could investigate other agile software development projects to verify the findings. Effort estimation using various techniques, such as planning poker games and story points, should be studied. Furthermore, future research is needed to examine the impact of other factors on learning, such as learning structures and formal training in estimations. Effort estimates made at the story and release planning levels should be examined.

VIII. CONCLUSION

This study investigates effort estimation in various development activities, including feature development, bug fixing, and refactoring in agile software development based on data collected from a large agile project. The results indicate that estimation of agile development does not improve over time, as claimed in the literature. Our data also indicate that no difference exists in the magnitude of estimation errors between feature tasks and bug/refactoring tasks, while bug fixes and refactoring tasks are overestimated more frequently than feature tasks. This study builds on research concerning software estimation in agile software development by investigating estimations across time and different development activities.

REFERENCES

[1] T. DeMarco, *Controlling Software Projects: Management, Measurement and Estimation*. New York, NY, USA: Yourdon Press, 1982.

[2] M. Choetkiertikul, H. K. Dam, T. Tran, T. Pham, A. Ghose, and T. Menzies, "A deep learning model for estimating story points," *IEEE Trans. Softw. Eng.*, vol. 45, no. 7, pp. 637–656, Jul. 2019.

[3] M. Jorgensen, "What we do and don't know about software development effort estimation," *IEEE Softw.*, vol. 31, no. 2, pp. 37–40, Mar. 2014.

[4] M. Jørgensen and T. M. Gruschke, "The impact of lessons-learned sessions on effort estimation and uncertainty assessments," *IEEE Trans. Softw. Eng.*, vol. 35, no. 3, pp. 368–383, Jan. 2009.

[5] B. Flyvbjerg and A. Budzier, "Why your IT project may be riskier than you think," *Harv. Bus. Rev.*, vol. 89, pp. 23–25, Sep. 2011.

[6] L. Cao, B. Ramesh, and T. Abdel-Hamid, "Modeling dynamics in agile software development," *ACM Trans. Manage. Inf. Syst.*, vol. 1, no. 1, pp. 1–26, 2010.

[7] L. Cao, "Estimating agile software project effort: An empirical study," in *Proc. Americas' Conf. Inf. Syst.*, Toronto, ON, Canada, 2008, pp. 1–11.

[8] E. Dantas, M. Perkusich, E. Dilorenzo, D. F. S. Santos, H. Almeida, and A. Perkusich, "Effort estimation in agile software development: An updated review," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 28, no. 11n12, pp. 1811–1831, Nov. 2018.

[9] M. Fernandez-Diego, E. R. Mendez, F. Gonzalez-Ladron-De-Guevara, S. Abrahao, and E. Insfran, "An update on effort estimation in agile software development: A systematic literature review," *IEEE Access*, vol. 8, pp. 166768–166800, 2020.

[10] M. Adnan and M. Afzal, "Ontology based multiagent effort estimation system for scrum agile method," *IEEE Access*, vol. 5, pp. 25993–26005, 2017.

[11] M. Hamid, F. Zeshan, A. Ahmad, F. Ahmad, M. A. Hamza, Z. A. Khan, S. Munawar, and H. Aljuaid, "An intelligent recommender and decision support system (IRDSS) for effective management of software projects," *IEEE Access*, vol. 8, pp. 140752–140766, 2020.

[12] M. Cohn, *Agile Estimating and Planning*. London, U.K.: Pearson, 2005.

[13] H. D. P. De Carvalho, R. Fagundes, and W. Santos, "Extreme learning machine applied to software development effort estimation," *IEEE Access*, vol. 9, pp. 92676–92687, 2021.

[14] M. S. Khan, F. Jabeen, S. Ghouzali, Z. Rehman, S. Naz, and W. Abdul, "Metaheuristic algorithms in optimizing deep neural network model for software effort estimation," *IEEE Access*, vol. 9, pp. 60309–60327, 2021.

[15] N. Rankovic, D. Rankovic, M. Ivanovic, and L. Lazic, "A new approach to software effort estimation using different artificial neural network architectures and Taguchi orthogonal arrays," *IEEE Access*, vol. 9, pp. 26926–26936, 2021.

[16] L. Cao and B. Ramesh, "Agile software development: Ad hoc practices or sound principles?" *IT Prof.*, vol. 9, no. 2, pp. 41–47, Mar. 2007.

[17] K. Beck, *Extreme Programming Explained: Embrace Change*. Boston, MA, USA: Addison-Wesley, 2000.
[18] J. Hill, L. C. Thomas, and D. E. Allen, "Experts' estimates of task durations in software development projects," *Int. J. Project Manage.*, vol. 18, no. 1, pp. 13–21, Feb. 2000.
[19] M. C. Ohlsson, C. Wohlin, and B. Regnell, "A project effort estimation study," *Inf. Softw. Technol.*, vol. 40, no. 14, pp. 831–839, Dec. 1998.
[20] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in *Proc. 4th Int. Workshop Mining Softw. Repositories*, Minneapolis, MN, USA, May 2007, p. 1.
[21] M. Hamill and K. Goseva-Popstojanova, "Analyzing and predicting effort associated with finding and fixing software faults," *Inf. Softw. Technol.*, vol. 87, pp. 1–18, Jul. 2017.
[22] M. Jorgensen and M. Shepperd, "A systematic review of software development cost estimation studies," *IEEE Trans. Softw. Eng.*, vol. 33, no. 1, pp. 33–53, Jan. 2007.
[23] M. Jørgensen, "A review of studies on expert estimation of software development effort," *J. Syst. Softw.*, vol. 70, nos. 1–2, pp. 37–60, 2004.
[24] T. Mukhopadhyay, S. S. Vicinanza, and M. J. Prietula, "Examining the feasibility of a case-based reasoning model for software effort estimation," *MIS Quart.*, vol. 16, no. 1, pp. 155–171, 1992.
[25] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," *IEEE Trans. Softw. Eng.*, vol. 23, no. 11, pp. 736–743, Nov. 1997.
[26] M. Usman, E. Mendes, F. Weidt, and R. Britto, "Effort estimation in agile software development: A systematic literature review," in *Proc. 10th Int. Conf. Predictive Models Softw. Eng.*, Turin, Italy, Sep. 2014, pp. 82–91.
[27] J. Wen, S. Li, Z. Lin, Y. Hu, and C. Huang, "Systematic literature review of machine learning based software development effort estimation models," *Inf. Softw. Technol.*, vol. 54, no. 1, pp. 41–59, 2012.
[28] B. Ramesh, L. Cao, and R. Baskerville, "Agile requirements engineering practices and challenges: An empirical study," *Inf. Syst. J.*, vol. 20, no. 5, pp. 449–480, Nov. 2007.
[29] S. Bilgaiyan, S. Sagnika, S. Mishra, and M. Das, "A systematic review on software cost estimation in agile software development," *J. Eng. Sci. Technol. Rev.*, vol. 10, no. 4, pp. 1–14, 2017.
[30] T. Schweighofer, A. Kline, L. Pavlic, and M. H. ko, "How is effort estimated in agile software development projects?" in *Proc. Workshop Softw. Quality Anal., Monit., Improvement, Appl.*, Budapest, Hungary, 2016, pp. 79–110.
[31] I. Myrtveit, E. Stensrud, and M. Shepperd, "Reliability and validity in comparative studies of software prediction models," *IEEE Trans. Softw. Eng.*, vol. 31, no. 5, pp. 380–391, May 2005.
[32] Y. Miyazaki, M. Terakado, K. Ozaki, and H. Nozaki, "Robust regression for developing software estimation models," *J. Syst. Softw.*, vol. 27, no. 1, pp. 3–16, Oct. 1994.
[33] M. Jørgensen, T. Halkjelsvik, and K. Liestøl, "When should we (not) use the mean magnitude of relative error (MMRE) as an error measure in software development effort estimation?" *Inf. Softw. Technol.*, vol. 143, Mar. 2022, Art. no. 106784.
[34] K. Molokken-Ostvold and M. Jorgensen, "A comparison of software project overruns–flexible versus sequential development models," *IEEE Trans. Softw. Eng.*, vol. 31, no. 9, pp. 754–766, Sep. 2005.
[35] K. Beck and M. Fowler, *Planning Extreme Programming*. New York, NY, USA: Addison-Wesley, 2001.
[36] M. Jørgensen, "A review of studies on expert estimation of software development effort," *J. Syst. Softw.*, vol. 70, pp. 37–60, Feb. 2002.
[37] M. Jørgensen and D. I. K. Sjøberg, "Impact of experience on maintenance skills," *J. Softw. Maintenance Evol., Res. Pract.*, vol. 14, no. 2, pp. 123–146, Mar. 2002.
[38] P. Abrahamsson and J. Koskela, "Extreme programming: A survey of empirical data from a controlled case study," in *Proc. Int. Symp. Empirical Softw. Eng.*, 2004, pp. 73–82.
[39] M. Kim, T. Zimmermann, and N. Nagappan, "An empirical study of RefactoringChallenges and benefits at Microsoft," *IEEE Trans. Softw. Eng.*, vol. 40, no. 7, pp. 633–649, Jul. 2014.
[40] M. F. Zibran and C. K. Roy, "Conflict-optimal scheduling of prioritised code clone refactoring," *IET Softw.*, vol. 7, no. 3, pp. 167–186, Jun. 2013.
[41] K. Molokken and M. Jørgensen, "A review of surveys on software effort estimation," in *Proc. Int. Symp. Empirical Softw. Eng.*, Rome, Italy, Sep. 2003, pp. 223–230.
[42] M. Bloch, S. Blumberg, and J. Laartz, "Delivering large-scale it projects on time, on budget, and on value," *McKinsey Quart.*, vol. 27, pp. 2–7, Jun. 2012.
[43] M. Jorgensen and K. Molokken-Ostvold, "Reasons for software effort estimation error: Impact of respondent role, information collection approach, and data analysis method," *IEEE Trans. Softw. Eng.*, vol. 30, no. 12, pp. 993–1007, Dec. 2004.
[44] M. Usman, "Improving expert estimation of software development effort in agile contexts," Ph.D. dissertation, Dept. Softw. Eng., Blekinge Inst. Technol., Karlskrona, Sweden, 2018.
[45] B. W. Boehm, *Software Engineering Economics*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1981.
[46] N. Mittas and L. Angelis, "Overestimation and underestimation of software cost models: Evaluation by visualization," in *Proc. 39th Euromicro Conf. Softw. Eng. Adv. Appl.*, Santander, Spain, Sep. 2013, pp. 317–324.
[47] T. Halkjelsvik and M. Jørgensen, "From origami to software development: A review of studies on judgment-based predictions of performance time," *Psychol. Bull.*, vol. 138, no. 2, pp. 238–271, 2012.
[48] T. Little, "Schedule estimation and uncertainty surrounding the cone of uncertainty," *IEEE Softw.*, vol. 23, no. 3, pp. 48–54, May 2006.
[49] K. R. Hammond, *Human Judgement and Social Policy: Irreducible Uncertainty, Inevitable Error, Unavoidable Injustice*. New York, NY, USA: Oxford Univ. Press, 1996.
[50] K. Lyytinen and D. Robey, "Learning failure in information systems development," *Inf. Syst. J.*, vol. 9, no. 2, pp. 85–101, Apr. 1999.
[51] N. Schmitt, B. W. Coyle, and L. King, "Feedback and task predictability as determinants of performance in multiple cue probability learning tasks," *Organizational Behav. Hum. Perform.*, vol. 16, no. 2, pp. 388–402, Aug. 1976.
[52] V. Duarte, "No estimates: How to measure project progress without estimating," Create Space Independent Publishing Platform, Scotts Valley, CA, USA, Tech. Rep., 2015.
[53] T. M. Shaft and I. Vessey, "The role of cognitive fit in the relationship between software comprehension and modification," *MIS Quart.*, vol. 30, pp. 29–55, Mar. 2006.
[54] A. Martini, J. Bosch, and M. Chaudron, "Investigating architectural technical debt accumulation and refactoring over time: A multiple-case study," *Inf. Softw. Technol.*, vol. 67, pp. 237–253, Nov. 2015.

LAN CAO received the Ph.D. degree in computer information systems from Georgia State University, in 2005. She is currently a Professor in information technologies and decision sciences with Old Dominion University, Norfolk, VA, USA. Her work appears in many journals, including *Information Systems Research*, *Journal of Management of Information Systems*, *Journal of Association of IS*, *European Journal of IS*, *Information Systems Journal*, *Decision Support Systems*, *ACM Transactions on MIS*, and *IEEE Software* among others. Her research interests include agile software development and software process simulation and modeling.