IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received January 20, 2022, accepted February 2, 2022, date of publication February 10, 2022, date of current version February 23, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3150878

Modeling Iteration's Perspectives in Software Engineering

MAMOONA MUMTAZ[1], NAVEED AHMAD[2], M. USMAN ASHRAF[3], AHMED ALSHAFLUT[4], ABDULLAH ALOURANI[5], (Member, IEEE), AND HAFIZ JUNAID ANJUM[6]

[1]Department of Software Engineering, University of Management and Technology, Sialkot 51310, Pakistan
[2]Faculty of Computing, National University of Computer and Emerging Sciences (FAST), Islamabad 44000, Pakistan
[3]Department of Computer Science, Government College Women University, Sialkot 51040, Pakistan
[4]Faculty of Computing and IT, Albaha University, Al Bahah 61321, Saudi Arabia
[5]Department of Computer Science, Majmaah University, Al Majma'ah 11952, Saudi Arabia
[6]Department of Mathematics, COMSATS University Islamabad, Islamabad 45550, Pakistan

Corresponding author: Mamoona Mumtaz (mamona.mumtaz@skt.umt.edu.pk)

This work was supported by the Deanship of Scientific Research (DSR), Majmaah University, Saudi Arabia.

ABSTRACT Iteration is ubiquitous during software development and particularly notable in complex system development. It has both positive and negative effects; the positives of iteration include improving quality and understandability, reducing complexity and maintenance, leading to innovation, and being cost-effective in the long run; Negatives of iteration include; time, cost, and effort overrun. Its management is a challenging task and becomes more complex due to the non-uniformity of the terminology used at various places. Although Software Development Life Cycles (SDLC) are highly iterative, not much work related to them has been reported in the literature. Insights into iteration are explained in this paper by defining different perspectives (Exploration, Refinement, Rework, and Negotiation) on iteration through literature review, modeling each perspective, and simulating the effect of each iterative perspective on project completion time. An attempt has been made to create awareness about efficient use of iteration during software development by informing which perspective of iteration has what kind of impact on project completion time to avoid delays.

INDEX TERMS Iteration, software engineering, software development lifecycle, modeling.

I. INTRODUCTION

Software development processes do not move in a straightforward, linear fashion. Deviation from linear movement is widespread and is depicted as iteration [1]. Iteration is performing a task again once it has been done. It is considered unavoidable and particularly notable in the life cycle of complex systems. Iteration can be categorized as planned and unplanned iteration. Planned iterations occur in iterative software development, particularly agile software development [2]. In comparison to planned iteration, unplanned iteration is costly and may affect a project's outcome. There are numerous causes of unplanned iteration in software development processes, including volatile nature of software scope [1], [3]; inconsistency revealed at some later stage [1]; changing business rules, client criticism on prototypes [2], [4]; mistakes uncovered by testing [5], [6]; complexity, ambiguity or unclear requirements [7].

Iteration is well studied in product development [8]–[11], construction [12], design [13]–[19], and engineering disciplines [13], [16], [19], [20], whereas very few authors discuss them from software development viewpoint. A study gathered and summarised insights into iteration from the design and development discipline. Additionally authors create the taxonomy of iteration that clarifies differences between different perspectives on iteration. They have selected a few articles from software engineering literature as well [17].

Iteration is hard to oversee and control. In the software industry, it is usually connected with time, cost, and effort overwhelms [4], [5], [21]. Early iteration on the investigation of thought lessens the recurrence of requirement changes later in the development and reduces completion time because there will be less modification on later phases of development cycle [22]. It expands the amount and quality of the end product and, in addition, diminishes the data sources required

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

to create it [23]. Iteration leads towards innovation [24]. It increases the development process's complexity and moves from linear towards dynamic. It also restricts the multifaceted nature and positively affects the understandability and conceptual clarity [25]. One can use it as a tool to refine unclear goals into clear objectives [7]. Iteration makes the design adaptable, i.e. it boosts the perseverance of the design in future force changes as far as time and cost as well [26]. Empirical studies highlight that it also reduces the complexity and size of the code. For instance, iteration reduces code complexity, makes reverse engineering simple, and enhances the software design [27]. It has beneficial outcomes, including investigating ideas, finding and remedying blemishes, removing inconsistencies, and permitting development under unpredictability and change.

Although, iteration is unavoidable during software development [1], [6], [28]–[32], detailed analysis of unplanned iteration does not exist. Despite the iterative nature of SDLC, little attention has been paid to it. Most of the authors have used varied terms to refer to iteration at different places. Management of iteration is a challenging task, and it becomes more complex due to the use of different terms to define iteration at different places. To resolve issues related to iteration, there is a need for uniform terminologies. In software engineering literature, no such study exists that considers the issues surrounding unplanned iteration in software development processes. The authors of the present article gather insights into iteration to clarify the differences between different perspectives on iteration and find the impact of each perspective on project completion time. This article contributes in two ways; first, by defining different perspectives on iteration —based on source and stage of the SDLC in which it occurs— to clarify the distinctive viewpoints of iteration that consistently exist in the software engineering discipline; second, by modeling the impact of each perspective on project completion time.

A comprehensive analysis of different viewpoints of iteration in software engineering processes has been created through *literature review*. From an in-depth analysis of all relevant publications, it has been found that iteration positively influences quality, understandability, productivity, and conceptual clarity. The software, which is developed iteratively, provide flexibility and reduce maintenance. It reduces the complexity and code size, removes inconsistencies, fixes defects, and leads towards innovation. Iteration increases time, effort, and cost for a short time, but it is cost-effective in the long run. Overall, non-functional requirements get easily satisfied by iteration. This article defines different perspectives on iteration which exist in the software engineering discipline. Next, the authors have modeled the impact of each iterative situation on project completion time by using and enhancing [33]. A project manager should know of all these to manage a project in a better way.

The rest of this paper is organized into four sections. Section II introduces different perspectives on iteration to distinguish between diverse iterative circumstances in software development processes and demonstrates that previously no such distinction existed. Section III models each iterative perspective. Section IV verifies iterative perspectives and models by comparing simulation runs with deterministic solutions. Section V summarizes key points and conclusion.

II. PERSPECTIVES ON ITERATION

Iteration has different perspectives because different iterative circumstances may have diverse sources and impacts. An iteration might be seen from a different point of view contingent on the concern.

A. EXPLORATION

Dynamics of exploration involve an iterative process of seeing different alternatives, assessing those solutions, and selecting the optimal one. It incorporates the investigation of new thoughts to tackle an emerging issue and iterative convergence to a solution. Every progression of iterative procedure comprises of either a straightforward, surely known expansion, design, or alteration in implementation inspired by a better understanding of an issue acquired through the process [34]. Exploration alludes to an iterative process that concocts straightforward solutions, so there is less to change and rolling out those improvements is less demanding and enhances quality [7]. Exploration refines the problem statement and creates an in-depth analysis of the problem [35]. Authors portray iteration as exploring diverse design options, predicting each discretionary design's quality and selecting the one that best fits into a particular context [36]. It is central in critical thinking and problem-solving processes. Exploration usually includes the iteration of requirement building and high-level design, however not of low-level configuration, execution and testing [37].

Literature highlights that exploring different design options in the early stage of design are beneficial (see Table 1). Longitudinal action research found that investigating different design choices is the synchronous analysis of the issue and its solutions. Particularly for software development, performing this investigation during early or architectural design is advantageous [31]. Exploring the design choices is essential during software development, and just a couple of strategies exist to help in performing this investigation systematically. In User-Centred Software Development (UCSD), iterative prototyping for the exploration of the different design choices is one of the significant activities [40].

In requirement gathering, explore the requirements to get a higher-level understanding. In design, exploration is for inventing the innovative and straightforward solution [7]. Many writers talk about exploration in early design phases as [36] code iteration as exploring diverse design options, predicting each discretionary design's quality and selecting the one that best fits into a particular context. Exploration usually comes in earlier phases of SDLC. It usually includes an iteration of requirement building and high-level design [37]. System risks, e.g. risks identified with deficient, conflicting

TABLE 1. Summary: benefits of exploring different options in the early stage of software development process.

Effects	Selected benefits with references
Innovation	Exploration of the different design choices invents simple solution, leads towards innovation by feature differentiation and competitiveness [7, 24]
Rework	Early exploration reduces the occurrence of later changes, also makes changes less demanding, thus it reduces both the rework and the effort required for it [7, 22]
Understandability	Initial specification get clarified after exploration i.e., it gives better understanding of the problem [22, 34, 36]
Satisfaction	When ever there is an exploration in the early phases, the end product meets user needs and the user gets satisfied [22, 36]
Quality	Exploring diverse design options improves the quality of design and end products [7, 23]
Complexity	Exploration invents simple solutions those are less complex [36]
Risk Analysis	Risks related to incomplete, inconsistent and imprecise requirements get managed [38, 39]
Time	if there is exploration then project completes timely because later stages development time decreases [37]

and vague requirements, are all explored and managed through the requirements elicitation stage [38]. Exploration alludes to the procedure to make and/or check the initial system specification [41]. Open innovation should be embraced as a complementary approach to ease internal innovation. There are two types of innovation those typically software development firms adopt, i.e., exploration and exploitation to stand in a promptly changing technological environment [42].

Early exploration on the investigation of thought lessen the recurrence of requirement changes later in the development and reduce completion time because lesser modification will be required on later phases of the development cycle [22]. It invents simple arrangements which are less intricate [34]. At the point when there is an investigation of multiple ideas at early stages, then development proceeds with continuous improvement [43]. Exploration more often includes an iteration of requirements and high-level design, however not of later stages, so software development life cycle grows timely because later stages development time decreases [37] and customer satisfaction increases [36]. It refers to an iterative process that concocts simple solutions, and then changes become less complex [7].

B. REFINEMENT

Refinement enhances initial specification and has subtypes in terms of its impacts. One of those is refactoring that has minimal impact. This type of refinement is done when sufficient time is available or where products have aesthetic appeal or assessment criteria is subjective. It portrays a situation where essential requirements have been satisfied and experience further iterations to upgrade optional qualities, e.g., enhance the style or diminish cost. In general, refinement is the process of removing impurities and improving something by making small changes, e.g., refactoring. Different viewpoints about the after-effects of refinement from the literature are summarized in Table 2 and discussed below.

A few researchers characterize refinement as beginning with the basic introductory implementation of a small part of the issue and iteratively upgrading existing version, e.g., after reviewing the prototype with users, developers refine and extend it, this process continues through several iterations [34]. In the same way, [37] additionally depict refinement in the software development lifecycle as the first iteration should produce a miniature version of the system, and each iteration then enhances that version. *Extreme programming* is an agile technique that stresses the continuous refactoring of codebase [44]. While developing embedded systems, we refine the basic description into another representation that mirrors the choices we have made within exploration [36]. During software development, unclear arrangement of objectives is progressively refined into requirements [45]. It is also implausible that an architectural design process does not involve cycles to upgrade the design [46]. Traditional process management drives the differences out of processes by nonstop estimation, identifying errors, and process refinements [7]. In the software development process, refinement also exists in removing code clones. They depict that numerous practices can be utilized to eliminate the clones from code [47]. Larger organizations have different practices than agile, e.g., change control board in larger organizations and refactoring and continuous improvement in agile [32].

In SDLC, the prototype should be designed first and then refined in the next iterations [48]. User interface design and agile concern iteration, change, and refinement [2]. Numerous authors discuss refinement in the analysis, design, and implementation stages. As before, requirement analysis requirements are accumulated with little detail, and afterwards, those are detailed up within iterations (refinement) in the analysis [49]. In design, refinement is designing with a straightforward introductory outline, incorporating it persistently, and refactoring the outline [25], [50], [51]. In implementation, refinement is refactoring of code and database [25], [50]–[52]. According to [47], refinement (iteration) occur during implementation and maintenance stage.

Refactoring has both positive and negative aftereffects in terms of quality improvement. While coding, refactoring improves practicality, upgrades execution, diminishes code size, removes duplicate code, improves testability, improves extensibility and require less work to incorporate

TABLE 2. Summary: after-effects of refinement.

Effects	Selected consequences with references
Quality	Refinement improves software quality [25, 53] It improves the quality of legacy code by removing code smells [26, 54][55]
Flexibility	Refinement improves extensibility and it becomes easier to add new features both w.r.t time and cost[26, 56, 57]
Maintainability	Refinement has a positive effect on maintainability w.r.t both quality and cost [25, 27, 57, 58, 59, 60]
Productivity	Refinement improves productivity and helps developers to program faster [25, 56] Refinement decrease long term productivity [61]
Complexity	Refinement confines the complexity, improves software design, removes undesirable inter module dependencies, and makes reverse engineering easy [25, 26, 53, 56, 57, 62]
Testability	Refined code or document has less number of defects or bugs [56, 57]
Code Size	Refinement reduces code size by removing clones or duplicated code [27, 57]
Understandability	Refinement converts the vague goals into understandable requirements [25, 45, 53, 56] Refinement increases conceptual clarity and makes the code easier to understand [26, 63, 64] After refactoring it become hard for developers to understand code changes, they take time to readjust, and productivity less for short term [57, 65]
Modularity	Refinement improves the modularity and readability [57]
Innovation	Refinement leads towards exploration and optimizes architecture [46]
Cost Effective	Refinement is associated with cost minimization in long run [26]
Behavior	Occasionally the behavior of code changes after refining and adding new features [57]
Correctness	it is difficult to ensure program correctness after refactoring [57]
Integration	There is a difficulty of merging and integration after refactoring [66]

new components. It improves quality and reduces time to market [57]. Refinement helps work faster, either we measure direct or indirect, there is a positive influence of refactoring on software quality. Refactoring restricts the multifaceted nature, overall positively affects the understandability and conceptual clarity for application engineers and understudies [25], [64]. It enhances extensibility, simpler to include new elements, i.e. reduce its complexity, simplify reverse engineering, and enhance the software design. If there are clones in the system, they have to change everywhere, so maintenance effort increases. Because of the negative effect of clones, one can uproot code clones by dynamic refactoring (iterative refinement) [27].

However, some studies claim that all types of refactoring do not constantly improve quality; sometimes, refactoring degrades software quality as well [67]–[70]. Code refactoring could bring about an efficiency punishment in the short run if the coding style gets to be not the same as the style designers have become appended to [65]. It makes the configuration versatile, i.e. support the persistence of the outline in future power changes as far as time and cost [26]. Refinement leads towards the investigation of new ideas that reduce change costs by creating straightforward arrangements so that there is less to change, and rolling out those improvements is less complex [36]. The consequences of clones can be classified in both positive and negative ways. While seeing an optimistic viewpoint, clones diminish the development time by reusing code. There are no compelling reasons to compose the new code, so clones in the code enhance development efficiency [47].

C. REWORK

Rework is one of the iteration's perspectives that seem most regularly in literature. It is reattempting a work in the same manner as before due to changed information or suppositions. Rework requires the reiteration of an assignment since it has initially endeavoured with incorrect data and suppositions. An example of rework in software development is a change in requirements, or simply requirements misunderstood. Rework may be produced because a procedure is excessively unpredictable, so that it is impossible to recognize the most productive order of work execution beforehand. It may be because of issues that appeared during analysis or requirement changes. If the timing constraints require starting a project with incomplete information, it is impossible to eliminate rework because of changed input information later. Rework is adverse because of the increase in time and cost without any improved software performance and quality. After-effects of the rework gained from the literature are summarized in table 3.

Cycles often get to be compulsory when some irregularity is found. At the point when there is irregularity, then need to revise to eliminate the issues [1]. If the operational outline is not correct, it will be perceived at a prior stage, and iterations with requirements and design can be revamped [81]. Change can be due to many reasons, software testing sometimes detects errors in programming, while clients can change the requirement, and usability testing can bring about change even without mistakes or changes in requirements [2]. Requirements instability alludes to evolution or fluctuations in requirements throughout the SDLC, and it causes rework [3]. Many cycles in the refinement procedure are because of prerequisites just misjudged and/or misconstrued [82]. The additional limit for change makes projects vulnerable to prerequisites change, or requirement creep [83]. To decide software cost and schedule, the measure of revamping and how it is dealt with are essential components [84]. When estimating the cost of a project, it is hard to

TABLE 3. Summary: after-effects of rework.

Effects	Selected consequences with references
Time	Rework creates schedule overruns. A lot a time is needed to fix the code [4][5]
Effort	Rework increases the effort require to complete a project [4, 71]
Cost	Rework increases the cost required to complete a project [21, 72]
Maintainability	Rework decrease maintenance [57]
Fix defects	Rework fixes defects, improves understanding of developers by identifying mistakes and analyzing root causes of problem [73, 74]
Consistency	Rework removes inconsistencies during design and in specifications, Early rework is less costly [75]

TABLE 4. Summary: after-effects of negotiation.

Effects	Selected consequences with references
Time	Most common type of delay occurs due to information exchange and negotiation while resolving work issues [76]
Clashes /Conflicts	Negotiation makes trade-off between diverse stakeholders concerns and needs [77] Negotiations resolves clashes and get agreements among the multiple stakeholders requirements [78, 79] Negotiation unites all the stakeholders on a satisfactory set of requirements by making a common vision among partners and develops requirement definition for projects [80]
Rework	Negotiation picks right set of the requirements. By picking right requirements the client gets satisfied By negotiation, vague and unclear requirements get clear. It avoids costly rework or changes [79]

reflect numerous cost variables which influence the product cost, e.g., requirement changes [21].

Numerous activities in software development involve significant changes in the requirements, scope, and technology used. These changes are outside the development control and regularly happen inside a development life cycle [7]. Most design cycles result from issues identified as late as equipment or framework reconciliation time. At this point, when coordinated into a framework, these either do not perform the required functionality or execute the wrong functions [41]. There is a tremendous amount of change in a project regarding new and changed classes during its life cycle [85]. Change of requirements happens throughout a project. Function Points can be redesigned each time the use cases change so that they can determine the effect of a particular use case in the estimate of the complete project development [71].

There are some situations in the software development life cycle when a team turns the effort in a new direction due to a customer's change demand or the changing market. Then need to change the already developed portion to incorporate the changes [29]. Large organizations have resistance to change, and agile development welcomes the change. During software development, companies face the problem of rework and deferrals [44]. Unpredictable changes occur during development, and one must respond to those changes [86]. Small organizations find it challenging to manage time in light of changing customer requests [87]. Factors that contribute to rework are lack of expertise, lack of documentation, lack of communication, changes in requirements, lack of user involvement, and lack of adequate testing [88]. ''Rugby'' is an agile process model, and it permits response to change requirements [48]. Software engineers distinguish between artefacts that have more defects and, according to the defect density, take decisions related to rework [89]. Rework is the work implemented again because it was not appropriately accomplished for the first time. Short, medium and long term projects meet the different amount of rework. It exists in all phases of SDLC but the maximum amount in the requirement gathering phase.

The greater part of the outcomes of rework is negative, yet there are some positive angles as well, e.g., it enhances the understanding of developers [73]. Early rework is also less costly, and it removes inconsistencies during design and in specifications [75]. Rework identifies developers' mistakes, improves understanding of developers, and fix defects. It also diminishes maintenance [74]. Negative effects of rework are more obvious e.g., increases cost [21], increases effort [71], much time will be used in fixing code [5], increases overall development cycle [4]. Rework is the main reason for schedule delays, budget overrun, and risks even after delivery [88].

D. NEGOTIATION

Negotiation is an iteration perspective that describes the circumstance in which the trade-off is made between various members' objectives and constraints by understanding and negotiating their conflicting goals. Negotiation is utilized to consolidate the commitment from various members who have little information about each other's work, and they regularly have clashing targets. When too many conflicting parameters are involved, negotiation turns out to be excessively troublesome. The project comprises a dynamic chain of contracts, and iteration is moved by backtracking up to the hierarchy for decision making about what moves to make and growing another sub-chain of agreements to execute the decision [1]. Building a system that operates nicely with individuals of various backgrounds, in diverse places, and at different times is a significant challenge. There must be a requirements negotiation process that addresses the stakeholder heterogeneity.

The selective consequences of negotiation gained from the literature are presented in table 4 and discussed below.

Rather than rigorous requirements in contracts, now organizations require stakeholders with a shared vision and flexibility to rapidly renegotiate another solution once unanticipated issues or opportunities emerge [80]. Risk management comprises negotiation, and risks are overseen throughout the development life cycle. Risk analysis is a business-level decision support tool [38], [39]. It is essential to adjust the levels of simultaneousness and cycle appropriately to streamline the execution. The decision about when to release software is a business decision, and there is minimal information available regarding the business decisions in any software literature [84].

Requirement engineering involves a collaborative, interactive, and interdisciplinary negotiation process that includes diverse partners/ stakeholders. These stakeholders take part in a negotiation process so they can unite on a commonly satisfactory set of requirements [80]. To consolidate another change, one must first choose whether it should accept it or not and then decide to add this change in the previous or create another module. Negotiation occurs while doing risk analysis too [38], [39]. Negotiation makes a shared vision among partners and develops project requirements definition. It also participates in COTS (commercial off-the-shelf) development for obtaining and integrating, transition planning, COTS enhancement, and release planning. We should perform prototyping in advance of and throughout requirements negotiations [80]. Distributed software development may expand process duration, and there can be numerous possible reasons for this delay. The most common delay was resolving the work issues, e.g., if a chunk of the design or code needs to be altered or needs a sound understanding of the product. To resolve this kind of problem, the individuals at more than one site should be included in information exchange and negotiation [76]. Negotiation can also take part in the selection of tools that will be utilized in the project development [114]. While discovering an optimal solution to an issue in search-based software engineering, there is a need to trade-off between diverse stockholders' concerns and needs. The solution is assessed on various distinctive subjective criteria [77].

Prune the product tree (online game) is being utilized for eliciting requirements and buying a feature (online game) for prioritization and negotiation to manage communication and knowledge transfer issues during the requirement engineering in [79]. The fundamental objective of negotiation is to resolve clashes and agree between partners about the most critical requirements. Sometimes, the restricted budget accelerates the requirements negotiation to settle on each requirement's choices and needs. Stakeholders consult to set the priorities and update the requirement list according to the priority. Prototypes get refined based on the client's criticism. By picking the right set of requirements, the client may get satisfied. As a consequence of negotiation, the vague and unclear requirements get more precise, avoiding costly changes. During software development, there is a negotiation between customers and stakeholders. Working software and user involvement is more critical than lengthy documentation and contract negotiation [48]. We have verified the defined perspectives by mapping each publication against the iterative perspectives as shown in table 5.

III. MATHEMATICAL MODELING

This paper modified Braha and Bar-Yam's model [33] by incorporating the effects of exploration, refinement, rework, and negotiation perspectives on the iteration as discussed in the preceding section. Analysis in [33] is based on the overall density of the incomplete tasks α. Hence their model does not require investigation of the individual tasks separately. However, we develop our model for the individual tasks as the different kinds of iteration will occur in different tasks; hence, their effects will not be present uniformly in all tasks. We can then look at the overall density of the incomplete tasks to establish a comparison with the results of [33]. As a test example, we use a directed random graph which contains 10^5 tasks with connectivity $< k_{in} >=< k_{out} >= 12$ and assuming all tasks to be incomplete at the start. Initially the internal completion rate $r = 0.75$ and sensitivity value $\beta = 0.061$ for all the tasks.

In our model, a network of nodes represents the software development, and an individual node represents a particular task during the development life cycle. During the software development life cycle, the state of a task can either be "complete" or "incomplete". The state of each task is influenced by its incoming tasks, and it also influences the outgoing tasks. The state of a task changes from being complete to incomplete or vice-versa by some stochastic rules. If a task's state is complete and has more incoming edges, its probability of getting incomplete will be higher. If a task is incomplete, its state may become complete in the next instance, depending on its internal completion rate and incoming incomplete tasks.

Let $s_i(t)$ represent the state of the task i at time t, then in our model a task can be either complete (i.e.$s_i(t) = 1$) or incomplete (i.e.$s_i(t) = 0$). Let k_i^{in} be the number of incoming edges to the task i, then $k_i(t) = k_i^{in} - \sum_{j:(j,i)\in E} s_j(t)$ represents the number of incomplete tasks connected to the task i at time t. Following [33], the state of a task changes according to the stochastic rules given in (1), (6), (16), (20), (24), and (25). Corresponding to the case of task i being complete at time t and task i being incomplete at time t, the state of task changes according to (1) and (25) respectively.

$$s_i(t+1) = \begin{cases} 0 & \text{with probability } \tanh(\beta_i k_i(t)), \\ 1 & \text{with probability } 1 - \tanh(\beta_i k_i(t)) \end{cases} \quad (1)$$

In the above equations, β_i represents the coupling of task i to its neighboring incomplete tasks, r_i represents the internal completion rate of that task, and $\tanh(x)$ represents the

TABLE 5. Selected studies mapped against iterative perspective it discuss.

Studies	Exploration	Refinement	Rework	Negotiation
[29, 90, 91]		✓		✓
[24, 31, 40, 42]	✓			
[6, 84, 92]			✓	✓
[27, 28, 45, 47, 93, 94, 95, 96, 97]		✓		
[77]	✓	✓		✓
[30, 35]	✓	✓	✓	✓
[3, 4, 25, 71, 75, 81, 85, 88, 98, 99, 100, 101, 102, 103, 104]			✓	
[78, 79, 105, 106]				✓
[21, 32, 44, 72, 73, 82, 107, 108]		✓	✓	
[7, 37, 43, 86]	✓	✓	✓	
[1, 2, 5, 48, 83]		✓	✓	✓
[38, 39, 41, 109]	✓			✓
[22, 34, 36, 89, 110, 111, 112, 113]	✓	✓		

hyperbolic tangent function defined as

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2}$$

We are updating the internal completion rate r, whereas the coupling between tasks β, also get updated. In particular, $\beta_i = 0$ corresponds to the case where all the tasks are independent, low β_i value represents that tasks are not much affected by its neighboring incomplete tasks and $\beta_i \to \infty$ means that all the tasks are completely dependent on its neighbors.

A. ANALYTICAL RESULTS

Adhering to the nomenclature of [33], we define the density of incomplete tasks at any time t as:

$$\alpha(t) = 1 - \sum_{i} s_i(t)/N, \tag{3}$$

where N is the total number of the tasks. Keeping the assumptions of Braha and Bar-yam i.e. $\beta_i = \beta$, $r_i = r$ for all tasks and the homogeneity condition. $k_i(t) \cong < k_{in} > \alpha(t)$, where $k_i(t)$ represents the number of incomplete tasks and $< k_{in} >$ represents the average number of incoming edges. The rate equation for the evolution of overall density of incomplete tasks is given by

$$\frac{d\alpha(t)}{dt} = (1 - \alpha(t))\tanh(\beta < k_{in} > \alpha(t)) - \alpha(t)r \times (1 - \tanh(\beta < k_{in} > \alpha(t))). \tag{4}$$

The time evolution of the state of an individual task is

$$\frac{ds_i}{dt} = -s_i(t)\tanh(\beta_i k_i(t)) + (1 - s_i(t))r_i \times (1 - \tanh(\beta_i k_i(t))) \tag{5}$$

To look at the overall density of the incomplete tasks, one can recover (4) by using (3) in (5).

B. EXPLORATION MODEL

In the beginning, we assume that coupling between tasks is β and the completion rate is r for all the tasks. Tasks are going to complete once the project gets started. Expanding or diminishing coupling will not impact the exploration of a task, but it will influence the completion rate if exploration occurs. This is because a task being explored will take more time to finish. Later on, during SDLC, the coupling between the tasks will be less because the explored solution is straightforward and less unpredictable. Since tasks are now straightforward and less complex, rework will be minimum, the completion rate will increase, and all tasks will be resolved in time. If exploration occurs, then the convergence rate will be less. After exploration, the convergence rate will increase, and all the tasks will be completed within time. In short, exploring a task will decrease the convergence rate for a short time. In the long run, the completion rate of tasks will be higher than their coupling, and all tasks will be converged timely. Increasing early exploration will increase the probability that the project will converge. To illustrate, for a given β and r, exploration adds simplicity, minimizes complexity to a project network, and all tasks converge to resolve state within time globally.

When initial state of task i is incomplete at time t and exploration occurs, the state changes according to the following stochastic rules.

$$s_i(t+1) = \begin{cases} 0 & \text{with probability } 1 - (r_i + \dfrac{N_e}{N}\alpha_e f_e(t)) \\ & (1 - \tanh(\beta_i k_i(t))), \\ 1 & \text{with probability } (r_i + \dfrac{N_e}{N}\alpha_e f_e(t)) \\ & (1 - \tanh(\beta_i k_i(t))) \end{cases} \tag{6}$$

Under the uniformity and homogeneity conditions stated in the beginning, we can write (5) as

$$\frac{ds_i}{dt} = -s_i(t)\tanh(\bar{\beta}\alpha) + (1 - s_i(t))r(1 - \tanh(\bar{\beta}\alpha)), \tag{7}$$

where $\bar{\beta} = \beta < k_{in} >$.

Furthermore, modify as follows:

$$\frac{ds_i}{dt} = -s_i(t)\tanh(\bar{\beta}\alpha) + (1 - s_i(t))(r + f_e(t)) \times (1 - \tanh(\bar{\beta}\alpha)), \tag{8}$$

where $f_e(t) = -w_\circ \mathrm{erfc}(t - t_\circ) + 0.75w_1$ is the exploration function which decreases the overall internal completion rate

during the exploration time and increases the overall internal completion rate after the exploration, $w_\circ, w_1$ are the exploration weights which needs to be specified a priori depending upon the nature of the task and the exploration work required, t_e is the time until which the exploration takes place. It should be noted that the choice of the exploration function, $f_e(t)$, is not unique. The particular form used here is motivated by the smooth properties of the complementary error function [115].

$$\text{erfc} = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-t^2} dt. \tag{9}$$

Let N be the total number of tasks in the development life cycle, $N_\circ$ be the normal task i.e., those that are not being explored, and N_e be the tasks that are being explored,then the density of incomplete normal tasks ($\alpha_\circ$) and tasks that are being explored (α_e) is defined as

$$\alpha_\circ = \frac{N_\circ - \sum_1^{N_\circ} S_i(t)}{N_\circ}, \quad \alpha_e = \frac{N_e - \sum_1^{N_e} S_i(t)}{N_e} \tag{10}$$

respectively.

The rate equation for the evolution of the density of incomplete normal tasks and incomplete tasks that are being explored becomes

$$\frac{d\alpha_\circ}{dt} = \tanh(\bar{\beta}\alpha)(1-\alpha_\circ) - \alpha_\circ r(1-\tanh(\bar{\beta}\alpha)), \tag{11}$$

$$\frac{d\alpha_e}{dt} = \tanh(\bar{\beta}\alpha)(1-\alpha_e) - \alpha_e(r+f_e(t))(1-\tanh(\bar{\beta}\alpha)), \tag{12}$$

respectively.

Combining (11) and (12), with the use of (10), we get the rate equation for the evolution of the overall density of incomplete tasks,

$$\frac{d\alpha}{dt} = (1-\alpha)\tanh(\bar{\beta}\alpha) - \alpha r(1-\tanh(\bar{\beta}\alpha)) - \frac{N_e}{N}\alpha_e f_e(t) \times (1-\tanh(\bar{\beta}\alpha)) \tag{13}$$

Note that if none of the tasks are being explored (i.e. $N_e = 0$), our model (13) reduces to that of Braha and Bar-Yam [33]. For $N_e \neq 0$, our mathematical model (13) captures the effects of exploration not being incorporated in the Braha and Bar-Yam's model. We can calculate the asymptotic solution of (13) to analyze the system behavior in the end state (i.e. when $t \to \infty$). Note that in the end state the slope, $\frac{d\alpha}{dt}$, representing the rate of change in the density of incomplete tasks will be zero i.e.

$$0 = (1-\alpha)\bar{\beta}\alpha - \alpha r(1-\bar{\beta}\alpha) - \frac{N_e}{N}\alpha_e f_e^*(1-\bar{\beta}\alpha) \tag{14}$$

where f_e^* is the asymptotic limit of the exploration function $f_e(t)$ i.e. $f_e^* = \lim_{t\to\infty} f_e(t)$. The above relation can be simplified by noting that in the end stage $\alpha_e \approx \alpha$, hence simplifying (14), we get

$$\alpha = \frac{\bar{\beta} - r - \frac{N_e}{N}f^*}{\bar{\beta}(1 - r - \frac{N_e}{N}f^*)}, \tag{15}$$

which reduces to the results of Braha and Bar-yam for $N_e = 0$. Notice that for $N_e \neq 0$, the value of α (obtained using (15)) is smaller compared to the analogous results of [33] which is due to the increased values of internal completion rate after the exploration as discussed earlier.

C. REFINEMENT MODEL

Refinement is the process of removing impurities and improvement of something by the making of small changes. It commonly occurs at end of each phase of development process i.e, UI, analysis, design, and implementation. During refinement, the value of the coupling parameter, β, will not change, but the internal completion rate of that task,however, will be little slow. During that time, the task is being refined hence the rate of completion, during that time, will be zero.

When initial state of task i is incomplete at time t and refinement occurs, the state changes according to the following stochastic rules.

$$s_i(t+1) = \begin{cases} 0 & \text{with probability } 1 - (r_i + \frac{N_r}{N}\alpha_r w_r f_r(t)) \\ & (1 - \tanh(\beta_i k_i(t))), \\ 1 & \text{with probability } (r_i + \frac{N_r}{N}\alpha_r w_r f_r(t)) \\ & (1 - \tanh(\beta_i k_i(t))) \end{cases} \tag{16}$$

The overall internal completion rate r will be less than the previous overall completion rate. For the tasks that are being refined, the state changes as

$$\frac{ds_i}{dt} = -s_i(t)\tanh(\bar{\beta}\alpha) + (1 - s_i(t)(r + w_r f_r(t)) \times (1 - \tanh(\bar{\beta}\alpha)), \tag{17}$$

where w_r is the refinement weight and $f_r(t)$ is a rectangular function whose value is one for the times when the refinement takes place and stays zero otherwise. The value of refinement weight w_r is chosen to ensure that the internal completion rate of the task stays zero during refinement.

When refinement occurs, the overall density of incomplete tasks evolves according to

$$\frac{d\alpha}{dt} = (1-\alpha)\tanh(\bar{\beta}\alpha) - \alpha r(1-\tanh(\bar{\beta}\alpha)) - \frac{N_r}{N}\alpha_r w_r f_r(t)(1-\tanh(\bar{\beta}\alpha)), \tag{18}$$

where N_r is the number of tasks that are being refined.

A similar analysis, as presented in the preceding section, gives the asymptotic solution of (18) in the end state,

$$\alpha = \frac{\bar{\beta} - r}{\bar{\beta}(1-r)} \tag{19}$$

Notice that the above solution (19) is the same as Braha and Bar-Yam's which is due to the fact that the refinement effects the convergence rate locally i.e., when the task is being refined, however, the exploration effects are global i.e., its effects are present for all time.

D. REWORK MODEL

When project discloses, state of all tasks is incomplete. At this stage, β is coupling between tasks and r is completion rate of all tasks. We are also assuming that all tasks start at same time. Rework is task attempting again due to changed input. Source of change input can be internal or external. Increasing or diminishing coupling will impact on the amount of rework.

Whenever a task gets complete, rework is determined by its impact on successor completed tasks. If the successor tasks are incomplete then there will be no rework due to input by current completed task. If the successor tasks are incomplete then output of the current completed task can make them incomplete again and create rework. The amount of rework created depends on how many incoming links are in the task, those are coming from incomplete task, and coupling weight of each link. By combination of both these factors (i.e. number of incoming incomplete links, and weight of each link) rework is calculated. At a time a task that is going to be complete, can make its successor tasks incomplete by changing input of successors. It is the situation that we have performed a task with some assumptions and after completion of some other task; the assumption get updated, so have to perform it again. When a task gets complete, it can make as many task incomplete as number of outgoing links to completed tasks.

When initial state of task i is incomplete at time t and rework occurs, the state changes according to the stochastic rules given in 20.

$$s_i(t+1)=\begin{cases}0 & \text{with probability } 1-(r_i+\dfrac{N_{rw}}{N}\alpha_{rw}w_{rw}f_{rw}(t))\\ & (1-\tanh(\beta_i k_i(t))),\\ 1 & \text{with probability } (r_i+\dfrac{N_{rw}}{N}\alpha_{rw}w_{rw}f_{rw}(t))\\ & (1-\tanh(\beta_i k_i(t)))\end{cases} \tag{20}$$

When there are too many outgoing links from the task that is going to be complete, then at that time, number of tasks those are going to be complete is less than number of tasks going to be incomplete. Because, rework is going to perform a task again due to changed input, the internal completion rate of the task r greater than previous internal completion rate, when it was first performed. Coupling (β) value will be high for successor completed tasks. When there will be rework then density of unresolved tasks ($\alpha(t)$) will be high leading towards slow convergence.

Hence whenever rework occurs, the state of a task changes according to

$$\begin{aligned}\frac{ds_i}{dt} &= -s_i(t)\tanh(\bar{\beta}\alpha)+(1-s_i(t)(r+w_{rw}f_{rw}(t))\\ &\quad\times(1-\tanh(\bar{\beta}\alpha)),\end{aligned} \tag{21}$$

where w_{rw} is the rework weight and $f_{rw}(t)$ is a step function which stays zero everywhere except at the time when rework occurs ($f_{rw}=1$).The rework weight w_{rw} is chosen such that the state of the task, in which rework comes, becomes incomplete ($s_i=0$) at that particular time.

The overall density of the incomplete tasks evolves according to

$$\begin{aligned}\frac{d\alpha}{dt} &= (1-\alpha)\tanh(\bar{\beta}\alpha)-\alpha r(1-\tanh(\bar{\beta}\alpha))\\ &\quad-\frac{N_{rw}}{N}\alpha_{rw}w_{rw}f_{rw}(t)(1-\tanh(\bar{\beta}\alpha)),\end{aligned} \tag{22}$$

where N_{rw} is the number of tasks in which rework occurs. In the end state, the asymptotic solution of (22) is given by

$$\alpha=\frac{\bar{\beta}-r}{\bar{\beta}(1-r)}, \tag{23}$$

which is same as Braha and Bar-Yam's showing that the rework effects are local, similar to the refinement effects, as discussed in the preceding section.

E. NEGOTIATION MODEL

Negotiation either leads to rework, refinement, or exploration. Convergence slowdowns whenever negotiation occurs. Completion rate of the tasks become too slow i.e., (new completion rate r is much less than previous completion rate).

$$s_i(t+1)=\begin{cases}0 & \text{with probability } 1-(r_i+\dfrac{N_n}{N}\alpha_n w_n f_n(t))\\ & (1-\tanh(\beta_i k_i(t))),\\ 1 & \text{with probability } (r_i+\dfrac{N_n}{N}\alpha_n w_n f_n(t))\\ & (1-\tanh(\beta_i k_i(t)))\end{cases} \tag{24}$$

When initial state of task i is incomplete at time t and negotiation occurs, the state changes according to the stochastic rules given in (24).

F. CUMULATIVE EFFECTS

During SDLC, different perspectives on iteration occur at different stages. In the preceding sections, we have modeled these effects individually. We, now present a model which incorporate all of these effects. When initial state of task i is incomplete at time t, the state changes according to the stochastic rules given in (25).

$$s_i(t+1)=\begin{cases}0 & \text{with probability } 1-(r_i+\dfrac{N_e}{N}\alpha_e f_e(t)+\\ & \dfrac{N_{rw}}{N}\alpha_{rw}w_{rw}f_{rw}(t)+\dfrac{N_r}{N}\alpha_r w_r f_r(t)+\\ & \dfrac{N_n}{N}\alpha_n w_n f_n(t))(1-\tanh(\beta_i k_i(t)))\\ 1 & \text{with probability } (r_i+\dfrac{N_e}{N}\alpha_e f_e(t)+\\ & \dfrac{N_{rw}}{N}\alpha_{rw}w_{rw}f_{rw}(t)+\dfrac{N_r}{N}\alpha_r w_r f_r(t)+\\ & \dfrac{N_n}{N}\alpha_n w_n f_n(t))(1-\tanh(\beta_i k_i(t)))\end{cases} \tag{25}$$

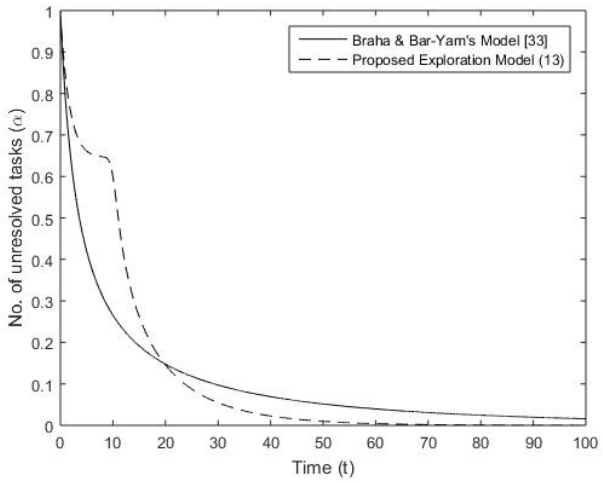

FIGURE 1. The time evolution of the density of incomplete tasks α using Braha and Bar-Yam's model (solid line) and our proposed exploration model (dashed line) given by (13). For the results presented it is assumed that 20% of the total tasks are being explored for $t \leq 10$.

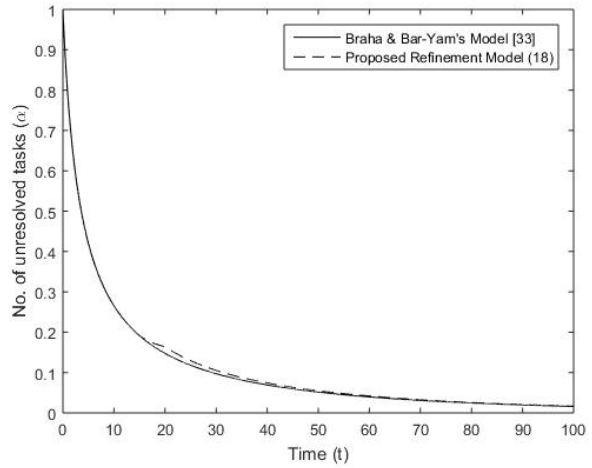

FIGURE 2. The density of the incomplete tasks α plotted as a function of time t. The solid curve corresponds to the Braha and Bar-Yam's model whereas the dashed curve represents our proposed refinement model (18). In refinement model (i.e. dashed line) it is assumed that 20% of the total tasks are being refined, refinement occurred between 15 and 20 time units (i.e., refinement duration; $15 \leq t \leq 20$).

$$\frac{d}{dt}\alpha = (1-\alpha)\tanh(\bar{\beta}\alpha) - [\alpha r + \frac{N_e}{N}\alpha_e f_e(t) + \frac{N_r}{N}\alpha_r w_r f_r(t) + \frac{N_{rw}}{N}\alpha_{rw} w_{rw} f_{rw}(t) + \frac{N_n}{N}\alpha_n w_n f_n(t)](1-\tanh(\bar{\beta}\alpha)) \quad (26)$$

The overall density of incomplete tasks, as a result of combined effects of different iteration perspectives, evolves according to (26)

IV. SIMULATIONS RESULTS AND VALIDATION

In order to validate the models given by (13), (17), (22), (24), and (26), we have conducted simulations with following test

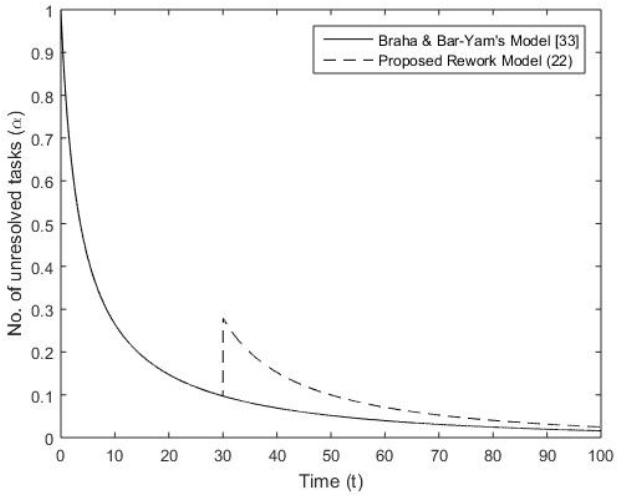

FIGURE 3. The evolution of the overall density of incomplete tasks α. The plotted results are from simulations of Braha and Bar-Yam's model and our proposed rework model, on the test example (see discussion). The solid line represents the results of the Braha and Bar-Yam's model [33] whereas dashed line corresponds to our proposed rework model as given in equation (22). In rework model (i.e., dashed line) it is assumed that 20% of the total tasks are being reworked, rework occurred at $t = 30$.

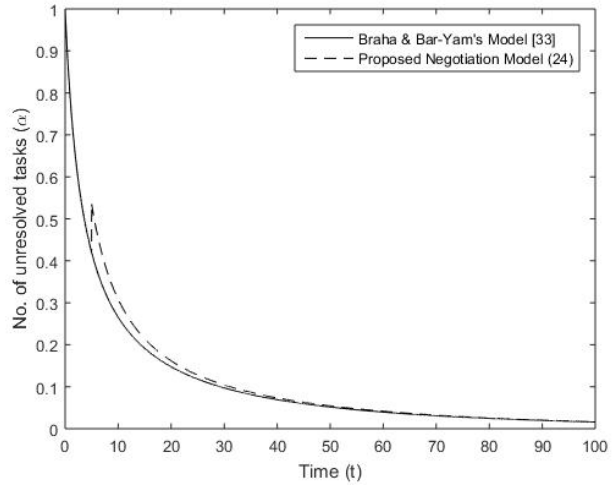

FIGURE 4. The evolution of the overall density of incomplete tasks α. Solid line represents the results of the Braha and Bar-Yam's model [33] whereas dashed line corresponds to our model as given in equation (22). In negotiation model (i.e., dashed line) it is assumed that 20% of the total tasks are negotiated, negotiation occurred at $t = 5$.

example (random graph with $N = 10^5$, $< k_{in} >=< k_{out} >= 12$, $N_\circ = 8 \times 10^4$,$N_e = 2 \times 10^4$, $w_\circ = 0$, $w_1 = 0.7$). Figure 1 shows the simulation results for [33] and (13). In this simulation it was assumed that 20% of the total tasks are being explored for the initial 10 time units. We therefore see that during the exploration, the rate of completion in our model is slower compared to Braha and Bar-Yam's model (with no exploration) whereas after the exploration, the completion rate increases in our model. On the other hand, the curve due to Braha and Bar-Yam's model shows a continuously decreasing trend in time with no effects from the tasks

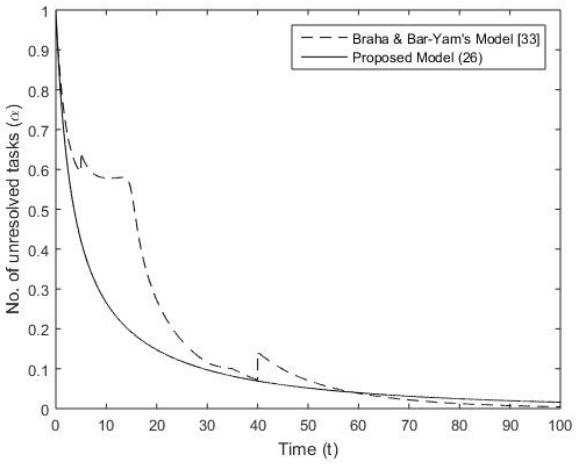

FIGURE 5. The evolution of the overall density of incomplete tasks α. The solid line represents the results of the Braha and Bar-Yam's model [33] whereas dashed line corresponds to our model as given in equation (26). In our proposed model (i.e., dashed line) negotiation takes place at $t = 5$, exploration continues for the first till 15 time units, refinement occurs at $t = 30$ and its duration is 5 time units, and reworks occurs at $t = 40$ time unit.

being explored. As mentioned above, the values of α, in the end state, obtained through proposed model (13) are smaller compared to the values of α obtained through Braha and Bar-Yam's Model consistent with our argument that the internal completion rate accelerates after the exploration. Also notice that due to higher internal completion rate(owing to the exploration effects), in our proposed model (13), the system has reached the end state ($t \approx 50$) earlier than Braha and Bar-Yam's model.

Figure 2 shows the simulation results (for the same test example) using our proposed refinement model (18) and [33].

For the presented simulation results, it is assumed that 20% of the tasks are being refined i.e. $N_r = 2 \times 10^4$ for $15 \leq t \leq 20$. It can be seen (dashed curve) that during the refinement, the overall internal completion rate is slower consistent with the theoretical effects of the refinement as explained earlier. Notice that during the refinement ($15 \leq t \leq 20$), the solid curve due to Braha and Bar-Yam's model continue to decrease uniformly showing no change in the pattern expected due to ongoing refinement in some of the tasks.

The simulation results for the proposed rework model (22), for $N_{rw} = 2 \times 10^4$, are shown in figure 3. At $t = 30$, the dashed curve jumps from 0.1 to 0.28 showing an increase in overall density of the incomplete tasks resulting from the rework. On the other hand, the solid curve due to

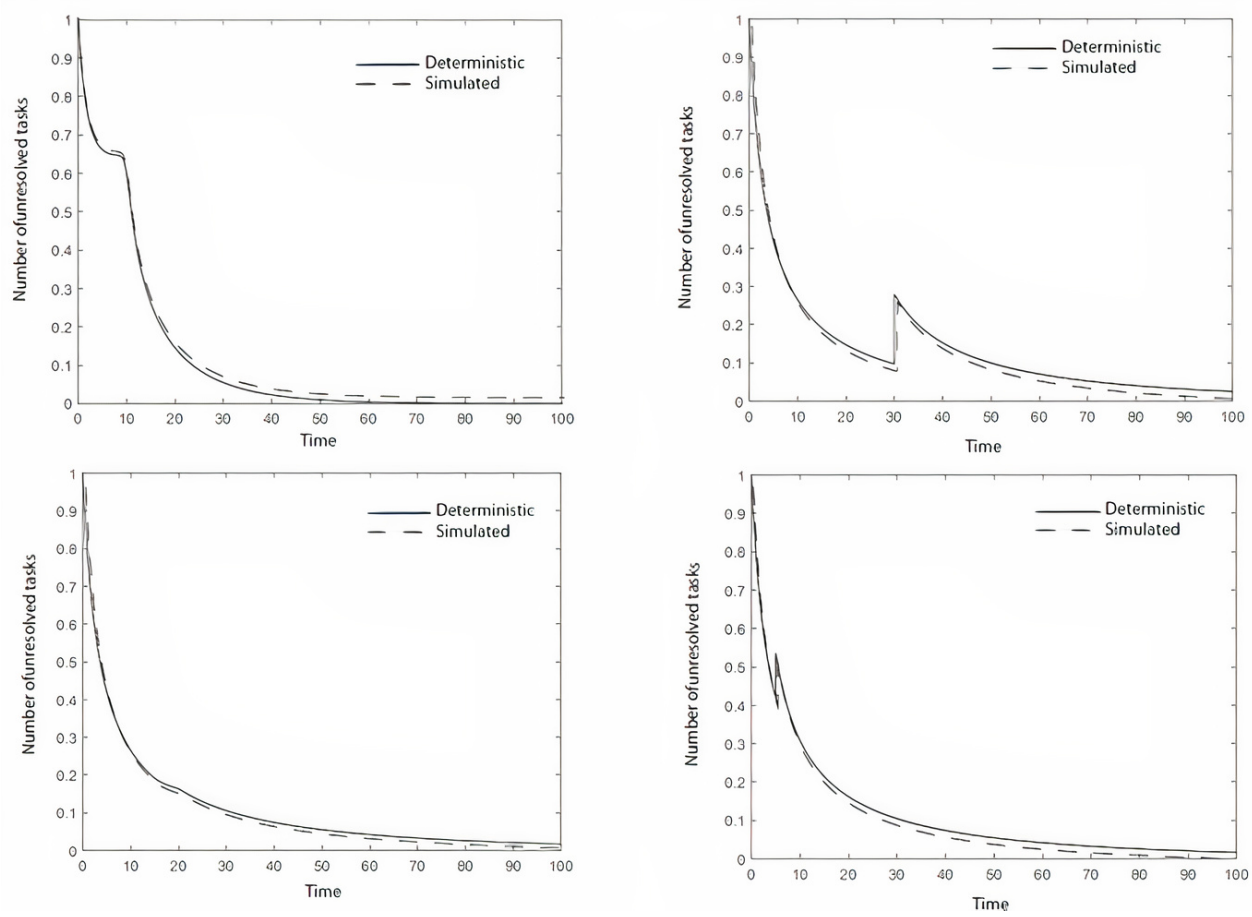

FIGURE 6. Comparison of typical simulation run with deterministic solution.

Braha and Bar-yam's continues to decrease showing that the overall density of the incomplete tasks decrease continuously without any influence from the rework.

To elaborate the effects of negotiation, we assume that the rework occurs in 20% of the tasks as a results of the negotiation. Hence the modeling for this particular case is the same as for the rework presented in the preceding sub section. We show the simulation results, for the same test example, for $N_n = 2 \times 10^4$, negotiation time $t_n = 5$ in figure 4. The dashed curve shows a sudden increase at $t = 5$ due to the rework arising as a consequence of the negotiation. As noted in the results presented in 4, the curve due to the Braha and Bar-yam's model shows no influence due to the negotiation at $t = 5$.

The simulation results for the combined effects of all perspectives on iteration (26) during the complete software development life cycle are shown in figure 5. In figure 5, the negotiation occurs at $t = 5$ showing sudden increase in fraction of incomplete tasks at that time. We can see in the dashed curve that the tasks are being explored till 15 time units. During the exploration time, the rate of convergence in our proposed model is slower as compared to Braha and Bar-Yam's model (i.e. solid curve). It can be seen (dashed curve) that tasks are being refined between 30 to 35 time units. The overall convergence rate is slower during the refinement time which is consistent with theoretical effects of refinement. At $t = 40$, the dashed curve jumps from 0.8 to 1.5 showing an increase in the fraction of incomplete tasks resulting from the rework. On the other hand, the solid curve due to Braha and Bar-Yam's model continues to decrease showing that there is no effect on the fraction of incomplete tasks due to the different perspectives on iteration.

We have demonstrated the effects of the different perspectives on iteration during SDLC. The proposed mathematical models capture the effects of individual perspectives of iteration on the density of incomplete tasks. The developed mathematical models for exploration and rework are entirely consistent with the theoretical effects; however, there are some limitations in the model of refinement and negotiation. The refinement's theoretical effect is that the convergence rate becomes slow during the refinement period but speeds up after refinement. We have modeled the refinement to slow down the internal completion rate r during refinement time, but the model does not exhibit the accelerating rate of convergence after the refinement. In software engineering literature, rework can be external and internal. In this paper, we have successfully modeled the effects of external rework, but internal rework implementation is future work.

Finally, we have performed simulations and compared the simulation results with the deterministic solution. The simulation runs followed the deterministic solution as shown in figure 6.

V. CONCLUSION AND FUTURE WORK

Different authors use different terms to refer to iteration in the software engineering literature. To resolve the issues related to unplanned iteration, we are contributing in two ways. First, we have defined different perspectives on iteration to clarify the different viewpoints through a literature review on iteration in the software engineering discipline. The second contribution is the modeling of iteration perspectives to determine the impact on project completion time. We have modeled the impact of each iterative situation on project completion time. Understanding different iterative situations and their after-effects play an essential role in success. It increases the visibility into processes by simulating different iterative models to predict outcomes, see future risks, forecast how much time can be delayed, and results in identifying improvements based on those. We have evaluated our results by comparing simulation runs with the deterministic solution.

The developed mathematical models have some limitations. Models for exploration and rework are entirely consistent with the theoretical effects. However, there are some limitations in the model of refinement and negotiation. In software engineering literature, rework can be external and/or internal. In this paper, we have successfully modeled the effects of external rework, but internal rework implementation is future work. Furthermore, negotiation can either lead to rework, refinement, or exploration. Here we are only considering the negotiation during the requirement phase of the software development life cycle, and we have modeled the case of rework as a consequence of negotiation. However, we can improve the negotiation model by randomly selecting the type of iteration (i.e. rework, refinement, or exploration) or by using some average effect combining the effects of all three perspectives as a consequence of negotiation. We will create a taxonomy of iteration in software engineering through a systematic literature review to validate different perspectives on iteration.

Acknowledgment

The authors would like to express their sincere gratitude to the Computer Science Department, COMSATS University Islamabad, for providing a research oriented environment to complete this research. The authors are also thankful to Prof. Aun Haider for his kind suggestions.

REFERENCES

[1] M. Dowson, "Iteration in the software process; review of the 3rd international software process workshop," in *Proc. 9th Int. Conf. Softw. Eng.*, 1987, pp. 36–41.
[2] J. Ferreira, J. Noble, and R. Biddle, "Agile development iterations and UI design," in *Proc. AGILE (AGILE)*, Aug. 2007, pp. 50–58.
[3] S. Ferreira, J. Collofello, D. Shunk, and G. Mackulak, "Understanding the effects of requirements volatility in software engineering by using analytical modeling and software process simulation," *J. Syst. Softw.*, vol. 82, no. 10, pp. 1568–1577, Oct. 2009.
[4] A. Gopal, T. Mukhopadhyay, and M. S. Krishnan, "The role of software processes and communication in offshore software development," *Commun. ACM*, vol. 45, no. 4, pp. 193–200, Apr. 2002.
[5] C. R. Cooley, "Daily iterations: Approaching code freeze and half the team is not agile," in *Proc. Agile Develop. Conf. (ADC)*, 2003, pp. 162–164.
[6] V.-P. Eloranta, K. Koskimies, and T. Mikkonen, "Exploring ScrumBut—An empirical study of scrum anti-patterns," *Inf. Softw. Technol.*, vol. 74, pp. 194–203, Jun. 2016.

[7] J. Highsmith and A. Cockburn, "Agile software development: The business of innovation," *IEEE Comput.*, vol. 34, no. 9, pp. 120–127, Sep. 2001.
[8] H.-B. Jun and H.-W. Suh, "A modeling framework for product development process considering its characteristics," *IEEE Trans. Eng. Manag.*, vol. 55, no. 1, pp. 103–119, Feb. 2008.
[9] D. Unger and S. Eppinger, "Improving product development process design: A method for managing information flows, risks, and iterations," *J. Eng. Des.*, vol. 22, no. 10, pp. 689–699, Oct. 2011.
[10] N. Bhuiyan, D. Gerwin, and V. Thomson, "Simulation of the new product development process for performance improvement," *Manage. Sci.*, vol. 50, no. 12, pp. 1690–1703, Dec. 2004.
[11] T. Taylor and D. N. Ford, "Tipping point failure and robustness in single development projects," *Syst. Dyn. Rev.*, vol. 22, no. 1, pp. 51–71, Mar. 2006.
[12] M. Haller, W. Lu, L. Stehn, and G. Jansson, "An indicator for superfluous iteration in offsite building design processes," *Archit. Eng. Des. Manage.*, vol. 11, no. 5, pp. 360–375, Sep. 2015.
[13] D. C. Wynn, C. M. Eckert, and P. J. Clarkson, "Modelling iteration in engineering design," in *Proc. 16th Int. Conf. Eng. Design (ICED)*, Paris, France, 2007.
[14] J. Clarkson and C. Eckert, *Design Process Improvement: A Review of Current Practice*. London, U.K.: Springer-Verlag, 2010.
[15] S.-H. Cho and S. D. Eppinger, "A simulation-based process model for managing complex design projects," *IEEE Trans. Eng. Manage.*, vol. 52, no. 3, pp. 316–328, Aug. 2005.
[16] R. Costa, "Productive iteration in student engineering design projects," Ph.D. dissertation, College Eng., Montana State Univ., Bozeman, MT, USA, 2004.
[17] D. C. Wynn and C. M. Eckert, "Perspectives on iteration in design and development," *Res. Eng. Des.*, vol. 28, no. 2, pp. 1–32, 2016.
[18] Y. Jin and P. Chusilp, "Study of mental iteration in different design situations," *Des. Stud.*, vol. 27, no. 1, pp. 25–55, Jan. 2006.
[19] M. J. Safoutin, "A methodology for empirical measurement of iteration in engineering design processes," Univ. Washington, Seattle, WA, USA, Tech. Rep. 3102710, 2003.
[20] R. Costa and D. K. Sobek, "Iteration in engineering design: Inherent and unavoidable or product of choices made?" in *Proc. ASME Int. Design Eng. Tech. Conf. Comput. Inf. Eng. Conf.* New York, NY, USA: American Society of Mechanical Engineers, 2003, pp. 669–674.
[21] S. Kang, O. Choi, and J. Baik, "Model-based dynamic cost estimation and tracking method for agile software development," in *Proc. IEEE/ACIS 9th Int. Conf. Comput. Inf. Sci. (ICIS)*, Aug. 2010, pp. 743–748.
[22] J. D. Blackburn, G. D. Scudder, and L. N. V. Wassenhove, "Improving speed and productivity of software development: A global survey of software developers," *IEEE Trans. Softw. Eng.*, vol. 22, no. 12, pp. 875–885, Dec. 1996.
[23] B. J. Moore, "Achieving software quality through requirements analysis," in *Proc. IEEE Int. Eng. Manage. Conf. (IEMC)*, Oct. 1994, pp. 78–83.
[24] K. Wnuk, D. Pfahl, D. Callele, and E.-A. Karlsson, "How can open source software development help requirements management gain the potential of open innovation: An exploratory study," in *Proc. ACM IEEE Int. Symp. Empirical Softw. Eng. Meas.*, Sep. 2012, pp. 271–280.
[25] R. Moser, P. Abrahamsson, W. Pedrycz, A. Sillitti, and G. Succi, "A case study on the impact of refactoring on quality and productivity in an agile team," in *Balancing Agility and Formalism in Software Engineering*. Berlin, Germany: Springer, 2008, pp. 252–266.
[26] A. Mavridis, A. Ampatzoglou, I. Stamelos, P. Sfetsos, and I. Deligiannis, "Selecting refactorings: An option based approach," in *Proc. 8th Int. Conf. Quality Inf. Commun. Technol. (QUATIC)*, Sep. 2012, pp. 272–277.
[27] C. K. Roy, M. F. Zibran, and R. Koschke, "The vision of software clone management: Past, present, and future (keynote paper)," in *Proc. Softw. Evol. Week-IEEE Conf. Softw. Maintenance, Reeng., Reverse Eng. (CSMR-WCRE)*, Feb. 2014, pp. 18–33.
[28] J. Chen, J. Xiao, Q. Wang, L. J. Osterweil, and M. Li, "Perspectives on refactoring planning and practice: An empirical study," *Empirical Softw. Eng.*, vol. 21, no. 3, pp. 1397–1436, Jun. 2016.
[29] J. Yli-Huumo, A. Maglyas, and K. Smolander, "How do software development teams manage technical debt?—An empirical study," *J. Syst. Softw.*, vol. 120, pp. 195–218, Oct. 2016.
[30] H. van Vliet and A. Tang, "Decision making in software architecture," *J. Syst. Softw.*, vol. 117, pp. 638–644, Jul. 2016.
[31] U. van Heesch, A. Jansen, H. Pei-Breivold, P. Avgeriou, and C. Manteuffel, "Platform design space exploration using architecture decision viewpoints—A longitudinal study," *J. Syst. Softw.*, vol. 124, pp. 56–81, Feb. 2017.
[32] K. Dikert, M. Paasivaara, and C. Lassenius, "Challenges and success factors for large-scale agile transformations: A systematic literature review," *J. Syst. Softw.*, vol. 119, pp. 87–108, Sep. 2016.
[33] D. Braha and Y. Bar-Yam, "The statistical mechanics of complex product development: Empirical and analytical results," *Manage. Sci.*, vol. 53, no. 7, pp. 1127–1145, Jul. 2007.
[34] V. R. Basil and A. J. Turner, "Iterative enhancement: A practical technique for software development," *IEEE Trans. Softw. Eng.*, vol. SE-1, no. 4, pp. 390–396, Dec. 1975.
[35] N. Ahituv, S. Neumann, and M. Zviran, "A system development methodology for ERP systems," *J. Comput. Inf. Syst.*, vol. 42, no. 3, pp. 56–67, 2002.
[36] D. D. Gajski and F. Vahid, "Specification and design of embedded hardware-software systems," *IEEE Des. Test. Comput.*, vol. 12, no. 1, pp. 53–67, Spring 1995.
[37] H. Kaindl, E. Arnautovic, D. Ertl, and J. Falb, "Iterative requirements engineering and architecting in systems engineering," in *Proc. 4th Int. Conf. Syst. (ICONS)*, 2009, pp. 216–221.
[38] V. N. L. Franqueira, Z. Bakalova, T. T. Tun, and M. Daneva, "Towards agile security risk management in RE and beyond," in *Proc. Workshop Empirical Requirements Eng. (EmpiRE)*, Aug. 2011, pp. 33–36.
[39] D. Verdon and G. McGraw, "Risk analysis in software design," *IEEE Security Privacy*, vol. 2, no. 4, pp. 79–84, Jul. 2004.
[40] M. Larusdottir, J. Gulliksen, and Å. Cajander, "A license to kill—Improving UCSD in agile development," *J. Syst. Softw.*, vol. 123, pp. 214–222, Jan. 2017.
[41] G. Mancini, D. Yurach, and S. Boucouris, "A methodology for HW-SW codesign in ATM," in *Proc. 32nd Annu. ACM/IEEE Design Autom. Conf.*, Dec. 1995, pp. 520–527.
[42] H. Munir, K. Wnuk, and P. Runeson, "Open innovation in software engineering: A systematic mapping study," *Empirical Softw. Eng.*, vol. 21, no. 2, pp. 684–723, 2016.
[43] B. Swaminathan and K. Jain, "Implementing the lean concepts of continuous improvement and flow on an agile software development project: An industrial case study," in *Proc. Agile India*, Feb. 2012, pp. 10–19.
[44] E. Kupiainen, M. V. Mäntylä, and J. Itkonen, "Using metrics in agile and lean software development—A systematic literature review of industrial studies," *Inf. Softw. Technol.*, vol. 62, pp. 143–163, Jun. 2015.
[45] T. J. Lehman and A. Sharma, "Software development as a service: Agile experiences," in *Proc. Annu. SRII Global Conf.*, Mar. 2011, pp. 749–758.
[46] J. Bosch and P. Molin, "Software architecture design: Evaluation and transformation," in *Proc. IEEE Conf. Workshop Eng. Comput.-Based Syst. (ECBS)*, Mar. 1999, pp. 4–10.
[47] D. Liu, D. Liu, L. Zhang, M. Hou, and C. Wang, "The prediction of code clone quality based on Bayesian network," *Int. J. Softw. Eng. Appl.*, vol. 10, no. 4, pp. 47–56, Apr. 2016.
[48] B. Bruegge, S. Krusche, and L. Alperowitz, "Software engineering project courses with industrial clients," *ACM Trans. Comput. Educ.*, vol. 15, no. 4, p. 17, 2015.
[49] L. Williams, K. Rubin, and M. Cohn, "Driving process improvement via comparative agility assessment," in *Proc. Agile Conf.*, Aug. 2010, pp. 3–10.
[50] T. Chow and D.-B. Cao, "A survey study of critical success factors in agile software projects," *J. Syst. Softw.*, vol. 81, no. 6, pp. 961–971, Jun. 2008.
[51] M. Fowler, *Refactoring: Improving the Design of Existing Code*. London, U.K.: Pearson, 2009.
[52] S. W. Ambler, "Survey says: Agile works in practice," *Doctor Dobbs J.*, vol. 31, no. 9, pp. 62–64, 2006.
[53] T. Mens and T. Tourwé, "A survey of software refactoring," *IEEE Trans. Softw. Eng.*, vol. 30, no. 2, pp. 126–139, Feb. 2004.
[54] M. Pizka, "Straightening spaghetti-code with refactoring?" in *Proc. Int. Conf. Softw. Eng. Res. Pract. (SERP)*, Las Vegas, NV, USA, Jun. 2004, pp. 846–852.
[55] E. van Emden and L. Moonen, "Java quality assurance by detecting code smells," in *Proc. 9th Work. Conf. Reverse Eng.*, 2002, pp. 97–106.
[56] M. Fowler and K. Beck, *Refactoring: Improving the Design of Existing Code*. Reading, MA, USA: Addison-Wesley, 1999.
[57] 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.

[58] A. van Deursen, "Program comprehension risks and opportunities in extreme programming," in *Proc. 8th Work. Conf. Reverse Eng.*, 2001, pp. 176–185.
[59] Y. Kataoka, T. Imai, H. Andou, and T. Fukaya, "A quantitative evaluation of maintainability enhancement by refactoring," in *Proc. Int. Conf. Softw. Maintenance*, 2002, pp. 576–585.
[60] C. Schofield, B. Tansey, Z. Xing, and E. Stroulia, "Digging the development dust for refactorings," in *Proc. 14th IEEE Int. Conf. Program Comprehension (ICPC)*, Jun. 2006, pp. 23–34.
[61] P. Abrahamsson and J. Koskela, "Extreme programming: Empirical results from a controlled case study," in *Proc. IEEE Int. Symp. Empirical Softw. Eng.*, Redondo Beach, CA, USA, Aug. 2004, pp. 73–82.
[62] S. Demeyer, S. Ducasse, and O. Nierstrasz, "Finding refactorings via change metrics," *ACM SIGPLAN Notices*, vol. 35, no. 10, pp. 166–177, 2000.
[63] R. W. Zmud, "Management of large software development efforts," *MIS Quart.*, vol. 4, pp. 45–55, Jun. 1980.
[64] H. Sondergaard, B. Thomsen, A. P. Ravn, R. R. Hansen, and T. Bogholm, "Refactoring real-time Java profiles," in *Proc. 14th IEEE Int. Symp. Object/Component/Service-Oriented Real-Time Distrib. Comput.*, Mar. 2011, pp. 109–116.
[65] E. Ammerlaan, W. Veninga, and A. Zaidman, "Old habits die hard: Why refactoring for understandability does not give immediate benefits," in *Proc. IEEE 22nd Int. Conf. Softw. Anal., Evol., Reeng. (SANER)*, Mar. 2015, pp. 504–507.
[66] Y. Brun, R. Holmes, M. D. Ernst, and D. Notkin, "Proactive detection of collaboration conflicts," in *Proc. 19th ACM SIGSOFT Symp. 13th Eur. Conf. Found. Softw. Eng. (SIGSOFT/FSE)*, 2011, pp. 168–178.
[67] S. Kaur and P. Singh, "How does object-oriented code refactoring influence software quality? Research landscape and challenges," *J. Syst. Softw.*, vol. 157, Nov. 2019, Art. no. 110394.
[68] J. Al Dallal and A. Abdin, "Empirical evaluation of the impact of object-oriented code refactoring on quality attributes: A systematic literature review," *IEEE Trans. Softw. Eng.*, vol. 44, no. 1, pp. 44–69, Jan. 2018.
[69] A. Almogahed, M. Omar, and N. H. Zakaria, "Categorization refactoring techniques based on their effect on software quality attributes," *Int. J. Innov. Techno, Logy Exploring Eng.*, vol. 8, pp. 439–445, Jun. 2019.
[70] M. Kim, D. Cai, and S. Kim, "An empirical investigation into the role of API-level refactorings during software evolution," in *Proc. 33rd Int. Conf. Softw. Eng.*, May 2011, pp. 151–160.
[71] J. A. Pow-Sang and E. Jolay-Vasquez, "An approach of a technique for effort estimation of iterations in software projects," in *Proc. 13th Asia Pacific Softw. Eng. Conf. (APSEC)*, Dec. 2006, pp. 367–376.
[72] R. E. Fairley and M. J. Willshire, "Iterative rework: The good, the bad, and the ugly," *Computer*, vol. 38, no. 9, pp. 34–41, Sep. 2005.
[73] D. Kasperek and M. Maurer, "Coupling structural complexity management and system dynamics to represent the dynamic behavior of product development processes," in *Proc. IEEE Int. Syst. Conf. (SysCon)*, Apr. 2013, pp. 414–419.
[74] S. Coronado and J. A. Jaén, "Incremental quality network," in *Proc. 2nd Asia–Pacific Conf. Quality Softw.*, 2001, pp. 59–64.
[75] A. G. Cass, S. M. Sutton, Jr., and L. J. Osterweil, "Formalizing rework in software processes," in *Proc. Eur. Workshop Softw. Process Technol.* Berlin, Germany: Springer, 2003, pp. 16–31.
[76] J. D. Herbsleb and A. Mockus, "An empirical study of speed and communication in globally distributed software development," *IEEE Trans. Softw. Eng.*, vol. 29, no. 6, pp. 481–494, Jun. 2003.
[77] G. R. Santhanam, "Qualitative optimization in software engineering: A short survey," *J. Syst. Softw.*, vol. 111, pp. 149–156, Jan. 2016.
[78] D. Robey, R. Welke, and D. Turk, "Traditional, iterative, and component-based development: A social analysis of software development paradigms," *Inf. Technol. Manage.*, vol. 2, no. 1, pp. 53–70, 2001.
[79] H. Ghanbari, J. Similä, and J. Markkula, "Utilizing online serious games to facilitate distributed requirements elicitation," *J. Syst. Softw.*, vol. 109, pp. 32–49, Nov. 2015.
[80] B. Boehm, P. Grunbacher, and R. O. Briggs, "Developing groupware for requirements negotiation: Lessons learned," *IEEE Softw.*, vol. 18, no. 3, pp. 46–55, May 2001.
[81] W. W. Royce, "Managing the development of large software systems," in *Proc. IEEE WESCON*, Los Angeles, CA, USA, 1970, vol. 26, no. 8, pp. 328–338.
[82] A. I. Anton, "Goal-based requirements analysis," in *Proc. 2nd Int. Conf. Requirements Eng.*, 1996, pp. 136–144.
[83] A. Powell, K. Mander, and D. Brown, "Strategies for lifecycle concurrency and iteration—A system dynamics approach," *J. Syst. Softw.*, vol. 46, nos. 2–3, pp. 151–161, Apr. 1999.
[84] G. A. Hansen, "Simulating software development processes," *Computer*, vol. 29, no. 1, pp. 73–77, Jan. 1996.
[85] H. M. Olague, L. H. Etzkorn, S. Gholston, and S. Quattlebaum, "Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes," *IEEE Trans. Softw. Eng.*, vol. 33, no. 6, pp. 402–419, Jun. 2007.
[86] F. S. Silva, F. S. F. Soares, A. L. Peres, I. M. de Azevedo, A. P. L. F. Vasconcelos, F. K. Kamei, and S. R. de Lemos Meira, "Using CMMI together with agile software development: A systematic review," *Inf. Softw. Technol.*, vol. 58, pp. 20–43, Feb. 2015.
[87] M.-L. Sánchez-Gordón and R. V. O'Connor, "Understanding the gap between software process practices and actual practice in very small companies," *Softw. Qual. J.*, vol. 24, no. 3, pp. 1–22, 2015.
[88] V. D. Ramdoo and G. Huzooree, "Strategies to reduce rework in software," *Int. J. Softw. Eng. Appl.*, vol. 6, no. 5, pp. 9–20, 2015.
[89] H. B. Yadav and D. K. Yadav, "A fuzzy logic based approach for phase-wise software defects prediction using software metrics," *Inf. Softw. Technol.*, vol. 63, pp. 44–57, Jul. 2015.
[90] Y. Shastri, R. Hoda, and R. Amor, "Spearheading agile: The role of the scrum master in agile projects," *Empirical Softw. Eng.*, vol. 26, no. 1, pp. 1–31, Jan. 2021.
[91] P. Hohl, J. Klünder, A. van Bennekum, R. Lockard, J. Gifford, J. Münch, M. Stupperich, and K. Schneider, "Back to the future: Origins and directions of the 'agile manifesto'–views of the originators," *J. Softw. Eng. Res. Develop.*, vol. 6, no. 1, pp. 1–27, 2018.
[92] J. Pernstål, T. Gorschek, R. Feldt, and D. Florén, "Requirements communication and balancing in large-scale software-intensive product development," *Inf. Softw. Technol.*, vol. 67, pp. 44–64, Nov. 2015.
[93] D. Firmenich, S. Firmenich, J. M. Rivero, L. Antonelli, and G. Rossi, "CrowdMock: An approach for defining and evolving web augmentation requirements," *Requirements Eng.*, vol. 23, no. 1, pp. 33–61, Mar. 2018.
[94] Y. Zhang, S. Shao, H. Liu, J. Qiu, D. Zhang, and G. Zhang, "Refactoring Java programs for customizable locks based on bytecode transformation," *IEEE Access*, vol. 7, pp. 66292–66303, 2019.
[95] Y. Yang, W. Ke, J. Yang, and X. Li, "Integrating UML with service refinement for requirements modeling and analysis," *IEEE Access*, vol. 7, pp. 11599–11612, 2019.
[96] A. Nilsson, L. M. Castro, S. Rivas, and T. Arts, "Assessing the effects of introducing a new software development process: A methodological description," *Int. J. Softw. Tools Technol. Transf.*, vol. 17, no. 1, pp. 1–16, Feb. 2015.
[97] A. A. S. Ivo, E. M. Guerra, S. M. Porto, J. Choma, and M. G. Quiles, "An approach for applying test-driven development (TDD) in the development of randomized algorithms," *J. Softw. Eng. Res. Develop.*, vol. 6, no. 1, pp. 1–31, Dec. 2018.
[98] E. Alégroth, R. Feldt, and P. Kolström, "Maintenance of automated test suites in industry: An empirical study on visual GUI testing," *Inf. Softw. Technol.*, vol. 73, pp. 66–80, May 2016.
[99] M. Hamid, F. Zeshan, A. Ahmad, S. Malik, M. Saleem, N. Tabassum, and M. Qasim, "Analysis of software success through structural equation modeling," *Intell. Autom. Soft Comput.*, vol. 31, no. 3, pp. 1689–1701, 2022.
[100] H. Villamizar, M. Kalinowski, A. Garcia, and D. Mendez, "An efficient approach for reviewing security-related aspects in agile requirements specifications of web applications," *Requirements Eng.*, vol. 25, no. 4, pp. 439–468, Dec. 2020.
[101] A. Sarwar, Y. Hafeez, S. Hussain, and S. Yang, "Towards taxonomical-based situational model to improve the quality of agile distributed teams," *IEEE Access*, vol. 8, pp. 6812–6826, 2020.
[102] M. W. Mkaouer, M. Kessentini, S. Bechikh, M. Ó. Cinnéide, and K. Deb, "On the use of many quality attributes for software refactoring: A many-objective search-based software engineering approach," *Empirical Softw. Eng.*, vol. 21, no. 6, pp. 2503–2545, Dec. 2016.
[103] M. Tufano, F. Palomba, G. Bavota, R. Oliveto, M. Di Penta, A. De Lucia, and D. Poshyvanyk, "When and why your code starts to smell bad," in *Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng. (ICSE)*, vol. 1, May 2015, pp. 403–414.
[104] C. Ebert and S. Brinkkemper, "Software product management—An industry evaluation," *J. Syst. Softw.*, vol. 95, pp. 10–18, Sep. 2014.

[105] M. B. Julian, "Artefacts and agile method tailoring in large-scale offshore software development programmes," *Inf. Softw. Technol.*, vol. 75, pp. 1–16, Jul. 2016.
[106] T. Alsanoosy, M. Spichkova, and J. Harland, "Identification of cultural influences on requirements engineering activities," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng., Companion (ICSE-Companion)*, Jun. 2020, pp. 290–291.
[107] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," *J. Syst. Softw.*, vol. 101, pp. 193–220, Mar. 2015.
[108] E. Knauss, A. Yussuf, K. Blincoe, D. Damian, and A. Knauss, "Continuous clarification and emergent requirements flows in open-commercial software ecosystems," *Requirements Eng.*, vol. 23, no. 1, pp. 97–117, Mar. 2018.
[109] R. Snijders, F. Dalpiaz, M. Hosseini, A. Shahri, and R. Ali, "Crowd-centric requirements engineering," in *Proc. IEEE/ACM 7th Int. Conf. Utility Cloud Comput.*, Dec. 2014, pp. 614–615.
[110] F. Wang, Z.-B. Yang, Z.-Q. Huang, C.-W. Liu, Y. Zhou, J.-P. Bodeveix, and M. Filali, "An approach to generate the traceability between restricted natural language requirements and AADL models," *IEEE Trans. Rel.*, vol. 69, no. 1, pp. 154–173, Mar. 2020.
[111] V. Alizadeh, M. Kessentini, M. W. Mkaouer, M. Ocinneide, A. Ouni, and Y. Cai, "An interactive and dynamic search-based approach to software refactoring recommendations," *IEEE Trans. Softw. Eng.*, vol. 46, no. 9, pp. 932–961, Sep. 2020.
[112] P. Newman, M. A. Ferrario, W. Simm, S. Forshaw, A. Friday, and J. Whittle, "The role of design thinking and physical prototyping in social software engineering," in *Proc. IEEE/ACM 37th IEEE Int. Conf. Softw. Eng.*, May 2015, pp. 487–496.
[113] M. Glinz and S. A. Fricker, "On shared understanding in software engineering: An essay," *Comput. Sci.-Res. Develop.*, vol. 30, nos. 3–4, pp. 363–376, Aug. 2015.
[114] B. Boehm, "Anchoring the software process," *IEEE Softw.*, vol. 13, no. 4, pp. 73–82, Jul. 1996.
[115] L. C. Andrews, *Special Functions of Mathematics for Engineers*, vol. 49. Bellingham, WA, USA: SPIE, 1998.

MAMOONA MUMTAZ received the M.S. degree in software Engineering from COMSATS University Islamabad, Pakistan, in 2018. She is currently working as a Lecturer with the University of Management and Technology. Her research interests include change in software development, software process improvements, and human–computer interaction.

NAVEED AHMAD received the Ph.D. degree in engineering design from the University of Cambridge, in 2011. He joined the Faculty of Computing—National University of Computer and Emerging Sciences (FAST-NUCES) as a Professor, in January 2019. His research interests include modeling and simulation, understanding the behavior of complex systems, information systems and security, software engineering, and human–computer interaction (user experience).

M. USMAN ASHRAF received the Ph.D. degree in computer science from King Abdulaziz University, Saudi Arabia, in 2018. He has served as a HPC Scientist at the HPC Centre, King Abdulaziz University. He is an Assistant Professor and the Head of the Computer Science Department, Government College Women University, Sialkot, Pakistan. His research on exascale computing systems, high performance computing (HPC) systems, parallel computing, HPC for deep learning, and location-based services systems has appeared in IEEE Access, *IET Software*, the *International Journal of Advanced Research in Computer Science*, the *International Journal of Advanced Computer Science and Applications*, the *International Journal of Information Technology and Computer Science*, the *International Journal of Computer Science and Security*, and several international IEEE/ACM/Springer conferences.

AHMED ALSHAFLUT received the M.Sc. degree in IT (applied computing) from Edinburgh Napier University, U.K., in 2012, and the Ph.D. degree in computer science from King Abdulaziz University, Saudi Arabia, in 2019. He received the Best Postgraduate Student Award for his Ph.D. degree and the Second Best Paper Award at IEEE L&T 2018.

ABDULLAH ALOURANI (Member, IEEE) received the bachelor's degree in computer science from Qassim University, Saudi Arabia, the master's degree in computer science from DePaul University, Chicago, and the Ph.D. degree in computer science from the University of Illinois Chicago. He is an Assistant Professor at the Department of Computer Science and Information, Majmaah University, Saudi Arabia. His current research interests include cloud computing, software engineering, security, and artificial intelligence. He is a member of ACM.

HAFIZ JUNAID ANJUM received the M.S. and Ph.D. degrees from the Department of Applied Mathematics and Theoretical Physics, University of Cambridge, U.K. He is currently working as an Assistant Professor at the Department of Mathematics, COMSATS University Islamabad. His research interests include applied mathematics, computational fluid dynamics, and high performance computing.