IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received February 2, 2022, accepted March 17, 2022, date of publication March 28, 2022, date of current version April 4, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3162878

Software Project Failure Process Definition

ALİ NİZAM
Department of Software Engineering, Fatih Sultan Mehmet Vakif University, 80523 İstanbul, Turkey
e-mail: ali.nizam@fsm.edu.tr

ABSTRACT Many researchers have attempted to identify the factors behind software project failures and their solutions from various perspectives. However, systematic and integrated process definitions of failure as process models for success are lacking. This study aims to build a process definition for software project failure as an anti-pattern by identifying the main phases and their relationships in terms of team behavior. We researched software engineering literature and case studies to gather information about critical incidents and repeating behaviors of teams in failed projects into a novel dataset. Grounded theory was employed to build a theoretical *foundation* for failure phase definitions from the collected data. The design structure matrix and Bayesian belief network were used for the quantitative assessment of the transitions between phases. The results revealed that common behavioral patterns occurred in approximately 89 percent of the case studies, supporting the decision to consider software project failure as a process. The proposed failure process definition has a simple structure that uses everyday concepts for phase names and reveals the critical behaviors leading a software project to failure Thus, it provides critical insights for software professionals, non-technical stakeholders, and managers to evaluate the progress of their projects and design strategies to avoid failure.

INDEX TERMS Bayesian belief network, grounded theory, qualitative process analysis, software process models.

I. INTRODUCTION

Successful software projects provide a competitive advantage to enterprises; therefore, the management of software and information technology projects is critical to the success of the organization and the careers of participating team members [1]. Conversely, the failure causes financial and moral loss, reduces or destroys expected gains [2]. Despite their importance and the spent resources, software project failure rates have remained high during the last ten years. The Standish Group CHAOS report found that the overall success rate was only 29 percent, while challenged projects accounted for 59 percent, and canceled projects for 19 percent; for big projects, the success rate is much lower and likely to be about 10 percent [3].

Many studies have investigated the top causes of failure as poor requirements, inadequate resources, unrealistic schedules, poor planning, unidentified risks [4], lack of top management commitment [5], inappropriately used technology [6], unrealistic or unarticulated project goals, inaccurate estimates, poorly defined system requirements, poor status reporting, and unmanaged risk [7]. Management techniques are generally the main cause of problems [8], [9]. Furthermore, the specific characteristics of a project, an organization, or an external environment play an essential role in failure [10].

From an integrated perspective, project failure is a combination of several factors and multiple interrelated problems [9] that can be defined as a mistake chain [8]. However, the causes of failure are often investigated in isolation [11]. The influence of failure factors on the result of a project and the relationship between them were evaluated by conducting root cause analysis (RCA) [11], logistic regression [12], grounded theory, and latent Dirichlet allocation [13]. Researchers have generally focused on defining the causal relationships between failure factors, not explaining the team behaviors behind the failure factors. Thus, some researchers argue that existing critical success (or failure) factor models have less concentration on communication, team, project management, and product-related issues [14].

The goal of this study is to identify patterns of team behavior in the context of software project failure in contrast to previous studies. *The main research questions (RQ) explored* are: "*RQ1*: *What are the critical and recurring team behaviors*

The associate editor coordinating the review of this manuscript and approving it for publication was Ricardo Colomo-Palacios.

that lead a software project to failure phase by phase?'' and ''*RQ2*: *What are the transition patterns between the failure phases*?''. Critical team behavior is a type of behavior that leads a project to fail by initiating an important deviation, preventing a solution, and negatively affecting overcoming events.

Because the reasons for failures create a complex network and it is difficult to address a software project failure in a linear and normative way, we first focused on identifying important team actions leading a software project to failure, and all the relationships between these actions. Then, for simplicity, we logically categorized team behaviors into failure phase definitions and highlighted the most likely transition between the phases to form a process definition.

The main contribution of this study is to formulate software project failure in the form of a process created by team behaviors behind failure factors. Identifying distinct breakpoints in the failure process supports more effectively measuring and controlling project progress and product quality and determining the future risks arising from the actions of the software team. In addition, we constructed a dataset that describes the relationship between software project failure factors and team behaviors (actions). Our failure process definition that is expressed in basic terms of behaviors supports simplifying communication between software developers, non-technical managers, and other stakeholders while enabling a concise description of higher-level concepts [11].

The remainder of this paper is organized as follows. Section 2 introduces related works from the software project management literature and defines the gap in the existing literature. Section 3 presents the methodology to define the failure process. Section 4 contains a systematic analysis of private and public case study results via qualitative grounded theory, quantitative design structure matrix (DSM), and Bayesian belief network (BBN) methods. Section 5 answers the research questions and discusses the results, the structure of the overall failure process, and threats to validity. Finally, Section 6 presents conclusions and future work.

II. BACKGROUND AND RELATED WORKS

In this section, the focus is to investigate existing work on failure progress in software projects and the prediction methods of software project failure to establish the justification for the contribution of the systemic failure process definition. We discussed basic concepts of existing methods such as risk management, root cause analysis, and anti-patterns that were used to define or evaluate software project progress and their effectiveness when failure arises in a project.

A. EXISTING FAILURE PROCESS DEFINITIONS

The mistake chain, which can be accepted as a process, is one of the main causes of failure for various project types. Mittelstaedt [8] defined mistake chains as follows: ''The root causes of project failure depend on similar human behaviors, biases, and blind spots''. He identified sequences of mistake chains as follows: ''1) An uncared initial problem, often minor in isolation, 2) Subsequent problems that increase the effect of the initial problem. 3) Inept corrective effects, 4) Disbelief that accelerates seriousness of the situation, 5) Generally, an attempt to hide the truth while attempting to remediate, 6) Sudden recognition that the situation is out of control 7) Finally, a total failure that causes significant loss''. However, software engineering requires a specific definition of the failure process, because it differs from other engineering disciplines owing to its essential difficulties such as complexity, relevance, mutability, and invisibility [15].

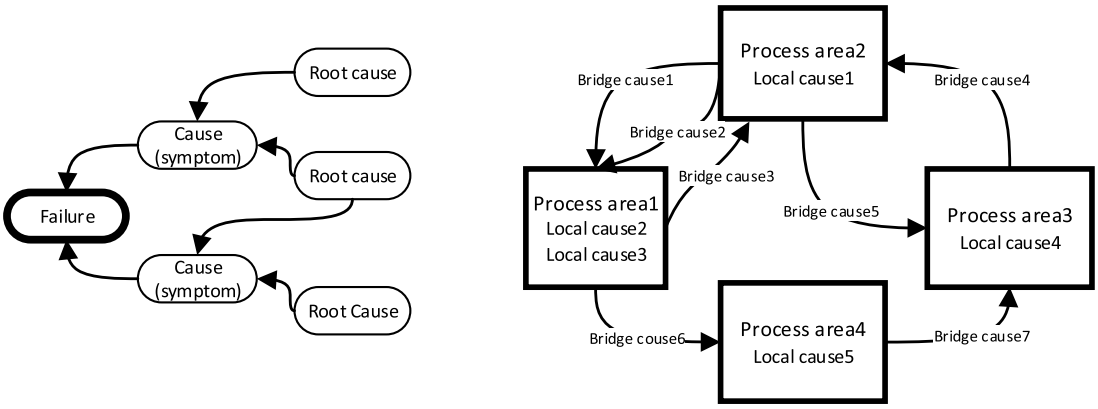
Even if there was no general process definition for software project failure, the researchers aimed to reveal notable phased definitions in the evaluation of failure. Jones stated [16] that ''most troubled projects, the trigger point for alerting clients and senior management occurs when a late milestone…is missed and…several well-known software disasters were not anticipated until the very day of expected deployment! Obviously, technical personnel and even many low-level managers were aware that things were not going according to plan, but the political pressures…are often so strong that no one wants to give bad news…When the problems finally become so serious that they can no longer be ignored, it may well be too late …, and hence the project is canceled or terminated''.

Glass combined some previous work [6] to create a process that consists of crunch mode, death march, and runaway steps. Boddie [17] defined *crunch mode* as there due to a threat reaching its original target, and a project team was working very hard to overcome this dilemma. Yourdon [18] defined *death march* as the parameters of a project that exceeded the norm by at least 50 percent, and the project had a nearly impossible schedule. KPMG characterized *runaway projects* as failing significantly to achieve their objectives, exceeding their original budget by at least 30 percent [19], and getting development efforts out of control.

The existing identification of failure modes provides important insights for understanding the status of software projects. However, even their literary values, these terms do not directly reflect the attitudes and behaviors of a team against deviation; their process definitions are independent and do not cover the whole process; some of their phases are intersected.

B. FAILURE PREDICTION METHODS

Failure prediction methods identify the factors and risks that affect a project's success or failure by reviewing and investigating resources and analyzing the relationship between predictors and outcomes [12]. A causal model for software project failure should completely determine the causal relationships affecting failure [11]. RCA allows the identification of causal relationships between the detected causes of project failures in basic or categorized forms as shown in Fig. 1a and 1b [20]. It produces root causes related to all process areas such as implementation, requirements, management, software testing, and deployment [11], [21]. Additionally, logistic regression is another method to help project managers to assess expected failures [12].



(a) Basic

(b) Categorized [11]

FIGURE 1. RCA methods.

In risk analysis and RCA, the root causes at the deepest level may or may not be related to team behaviors, although some studies categorize failure factors and examine management as a separate category [11]. Additionally, one team action may drive and assign to many failure factors. Thus, the identification of team behavior behind the failure factor and the connections between team behaviors leading a project to failure become unclear and ambiguous. For example, Boehm's top ten risk items contain many risks (failure factors) such as unrealistic schedules and budgets, developing the wrong functions or user interface, gold-plating, and straining computer-science capabilities [22]. The definitions of root causes are also similar such as unrealistic project objectives, team technical problems, lack of user involvement, requirement instability, problematic technology, etc. [12].

Another key aspect driving our research is that existing failure models, such as RCA-based models, have complex representations for non-technical people as shown in Fig. 1. Whereas, process quality models should provide explicit viewpoints and meet the needs of diverse stakeholder groups for a wide range of applications [23]. Management looks at the connection between the team structure and the problem domain to understand who owned the problem instead of complex explanations [24]. Thus, a new perspective is required to enlighten managers and customers about the progress of failure in software projects.

C. ANTI-PATTERN DEFINITIONS AND FAILURE PROCESS

Failure process definition can be considered an anti-pattern. Anti-patterns provide a common vocabulary for identifying problems, defective processes, and implementations. They provide useful insights into understanding the relationship between failure and its causes. However, from an integration perspective, common anti-patterns were presented as lists where they were isolated from one another [11]. The relationship between project progress and the time when the anti-patent symptoms appear is not clear except for rare and independent examples such as escalation commitment (continue against failure) [25], fire drill (forcing immediate delivery after slack time) [11], and death march (trying to catch impossible schedule) [18].

III. METHODOLOGY

We aimed to identify and extract behavioral failure patterns from case study data. Thus, we required methods to collect case study data, detect important patterns in the data, and qualitatively and quantitatively identify and analyze the connections between them. The main steps of the proposed methodology are shown in Fig. 2: (1) We researched relevant literature to derive the structured questions. (2) For qualitative analysis, we collected private and public case study data and analyzed them with grounded theory to identify team behaviors behind failure factors for building the failure process definition. We investigated teams' responses to each failure factor by examining failure causes and their context until reaching the team behaviors behind failure. It allows further analysis to understand how teams' failure responses were represented in terms of concepts, categories, and relations. (3) We applied quantitative research based on DSM and BBN to evaluate the transitions between phases.

We reviewed the literature and noted the similarities between case studies and anti-patterns with our code definitions to validate our findings. We also benefited from the definition, causes, symptoms, and consequences of similar anti-patterns [26] such as escalation failure and death march.

A. QUALITATIVE ANALYSIS

Our grounded theory-based case study data analysis contains the following interweaving steps: building memo questions from a literature analysis, theoretical sampling, constant comparison, coding, and saturation [27]. We preferred the Straussian grounded theory approach because it allowed for a general idea of where to begin, forcing the theory with structured questions [28], providing rigorous procedures, and techniques for analyzing interpretive case study data [29].

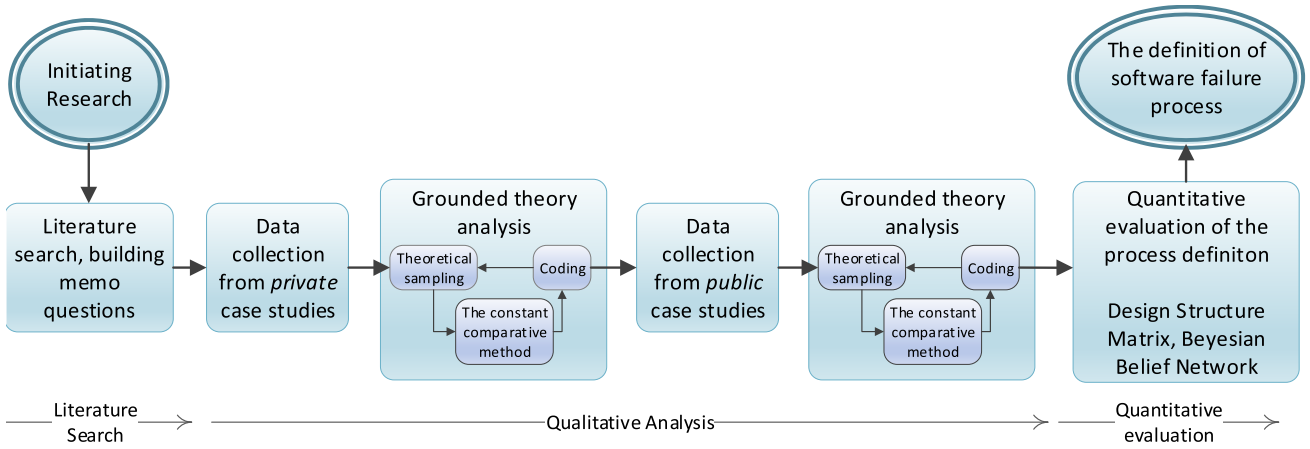


FIGURE 2. The main steps of our methodology.

We evaluated the results employing credibility, transferability, conformability, and dependability criteria [30], [29].

Corbin and Strauss [31] claimed that some surveys in the literature supported researchers to initiate a research project with some knowledge of the phenomenon being studied. Moreover, literature can be used to derive questions that largely determine the research methods and establish the boundaries of the study. These questions represent memos in grounded theory and are called memo questions. We built memo questions and linked them to pertinent expressions [32] to identify the context of team activities. The answers to the memo questions referred to previous and subsequent actions against failure factors that identified the code, connections, and categories.

Theoretical sampling [31] supports identifying what data to collect, where to find it, when it appears, and the selection of data sources when choosing the sample case studies in our study [33]. Thus, we filtered out case studies whose chronological order was unreachable because of the order required to detect transitions between phases. Therefore, because the failure process should be traceable and auditable to identify key points, we excluded failed projects that even if careful change management procedures were conducted, a suddenly appeared internal or external factor such as unexpected customer or manager decision led to a huge deviation and termination.

To detect failure factors, we accepted each person involved or contributing to the software project progress as a member of the software team, including the project manager, developer, sponsor, customer, or top manager.

The definition of software project failure is also important to choose appropriate case studies; however, it is imprecise and varied. Project management literature has many different failure definitions [34]. Generally, they are based on the degree of match or mismatch between design and reality of process and objectives [35]. We accepted the definition of failure as failed software projects are over budget and not delivered on time [36]; they do not create the impact they should have delivered [37], abandoned, or canceled with loss of investment [38]; they do not create client benefits or are excessively deviated from its budget and schedule [39].

Grounded theory-based data investigation involves incremental and iterative applications of three types of *coding*: open (identifying, naming, categorizing, and describing phenomena), axial (relating codes to each other), and selective (choosing a core category and relating it to other categories) [29]. We evaluated raw data of one case study on a sentence-by-sentence basis in each iteration to derive codes, concepts, and categories. The main steps are as shown in Fig. 3: (1) detecting the sections where visibility and scale of deviation vary considerably and matching them corresponding memo questions, (2) creating open codes using key points representing team behavior in the sections (3) creating axial code by categorizing open codes logically, and (4) building failure definition based on chronology in selective coding.

Each iteration investigating a case study requires the *constant comparison and update* of incidents, codes, and categories with previously collected data [40]. The constant comparison allowed us to control the data collection process and ensure the consistency of codes with data and other codes.

We considered different perspectives and actions of team members involved or contributing to the project's progress to conduct an integral and persuasive analysis [40]. We chose the most dominant action as a key point leading a project to failure even if there were objections from other members.

In open coding, we detected key points and assigned them with labels to build codes by summarizing incidents via team behaviors; the labels make concepts that support capturing and collecting the main objects, instances, incidents, and actions on the data [41].

The outcome of open coding was categories that supported the identification of concepts from the data by grouping similar concepts into categories or subcategories. We noted

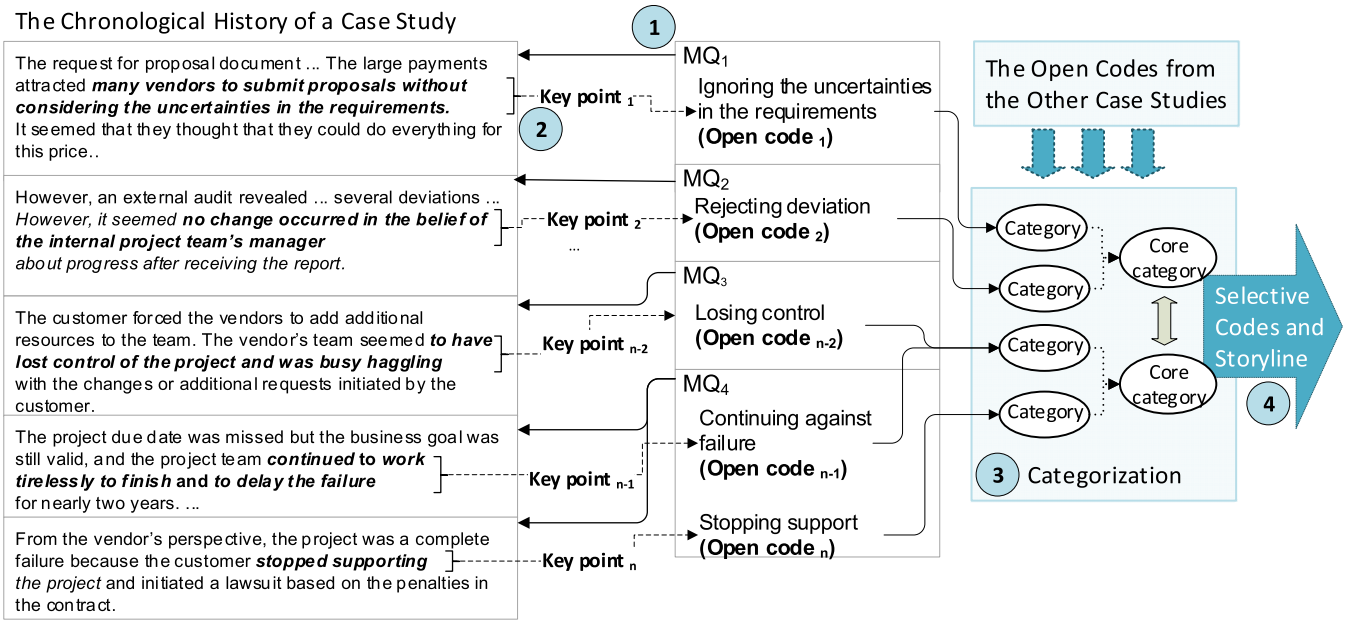


FIGURE 3. Analyzing case studies data with grounded theory.

the chronology of events to use the order of codes in axial and selective coding. *In axial coding*, the open codes were compared and categorized into code groups by looking for similarities and differences to construct axial codes. *In selective coding*, core categories defined by axial codes and their chronological order were refined to the conceptual and narrative description of *the storyline of the failure process definition*.

Theoretical saturation was evaluated to determine whether adding new data resulted in an update of the established theory [42] by examining the pattern of emergence of new concepts and transitions throughout the research process [23]. We calculated and visualized the cumulative sum of newly identified concepts and transitions to control saturation.

We first applied grounded theory to information from private failed case studies in various software development areas using memo questions. For enhancing credibility and decreasing subjectivity, we wrote down the speeches and evaluations of team members or reporters directly by avoiding interpretation. The main parameters used to define key failure incidents and behaviors were as follows:

- *Important event*: A critical incident, behavior or action creates a milestone in project progress toward failure by affecting the overcoming events. The attributes of the event are the cause of failure, the initiator, the time of the event, the next event, and the problem scale. The sources of information were meetings, reports, and emails.
- *The ratio between the actual and expected number of users*: Significantly lower numbers than the expected user numbers indicate failure. We evaluated high user numbers considering user satisfaction and productivity, because a manager may force end-users to use the system unwillingly. The sources of information were on-site observations, meetings, and system-usage logs.
- *User satisfaction*: It demonstrates the quality and content of interaction between the information produced by the system and the recipients [35]. On-site observations and meetings were the main resources for understanding a user's satisfaction level.
- *Developer viewpoint*: On-site observations and meetings were the main sources for obtaining a detailed understanding of developers' perspectives on project progress.

After completing the private case study analysis, further sampling was required to fill the gaps in the credibility and transferability of the emergent theory. This led to the public case study search and analysis. The data collection methods were different for private and public case studies; thus, we conducted the grounded theory coding process for them separately. However, we employed the same parameters and methods in analysis to maintain the standard grounded theory approach, which required an integrated coding process.

The public case study analysis was provided to improve the credibility and transferability of previous codes by saturating them using well-known failed and overbudget software project lists [6], [43]. To increase credibility, we gathered data from multiple sources such as public records, articles, newspapers, published reports, and official statements [44]. To ensure transferability and applicability, we evaluated the generalization of the findings by comparing the outcomes of private and public case studies and checking whether the same results were obtained in other contexts with similar properties and codes [29].

B. QUANTITATIVE ANALYSIS

Modeling a process requires decomposing the process into steps, determining transitions among the steps, and analyzing the sequence of the steps into a process flow [45]. *The quantitative assessment of process definition* via DSM and BBN unveiled the probability of relationships between core categories in selective coding. The inputs of quantitative assessment were the codes, occurrence numbers, and chronological order of codes identified by grounded theory. The output was the transition patterns between failure phases.

DSM is a system modeling tool that can represent system elements and their relationships in a compact way that highlights important patterns in the data [45]. It is a square matrix with identical row and column labels. A quantification scheme (not categorical) and time-based DSM were utilized for analysis. In time-based DSMs, the ordering of rows and columns indicates a flow-through time: upstream activities in a process precede downstream activities. In our work, the DSM represented the general characteristics of the system model via the chronological order of phases. However, an additional model was required to express the probabilistic relationships in the entire system.

Thus, we employed the BBN to represent the conditional dependence of phases in a directed graph that encodes probabilistic relationships among variables of interest [46]. A BBN utilizes a probability table that is associated with each node, providing the probabilities of each state of a variable and it provides a graphical representation of the causal structure using directed acyclic graphs [47].

IV. RESULTS

This section reports the grounded theory analysis results of case studies, followed by quantitative assessment results of the DSM and BBN methods. Four questions arise from the literature when considering team actions against failure, as shown in Table 1. The literature suggests a categorization of problems associated with their visibility and scale as minor, medium, and large [8], [16], [17]. Thus, the first three questions represent the team's answers to the progress of a problem as initiating, getting bigger, and serious or out of control. The fourth question address the actions of software teams when a failure has a high probability or is inevitable.

A. PRIVATE CASE STUDIES FOR INITIAL PROCESS DEFINITION

We gathered chronological failure data and critical incidents from seven failed software projects from different business sectors and software types to evaluate the failure progress in software projects. The context of these case studies varies from in-house software development to an extensive range of software applications such as document management, prototyping, and outsourcing. Private case study histories, matching memo questions, open codes, and resources are briefly presented in Supplementary Material A. The projects' general attributes are the goal, team structure, and team experience.

TABLE 1. Failure progress and memo questions.

Scale of deviation	Visibility of deviation	Resources	Memo questions (MQs) for failed software projects
Small	visible to the team as a minor problem	the minor problem [8], missing trigger point [16]	(MQ1) What is the primary action of a software team against problems that initiate failure?
Medium	visible to the team as a serious problem	accelerating seriousness [8], problems finally become so serious [16]	(MQ2) How does a software team react to a deviation when they understand its seriousness?
Large (hurt progress)	Visible to customers, managers, and other stakeholders	out of control [8], too late to solve the problem [16], a threat to reaching the original target [17]	(MQ3) How do a software team behave when they miss an important milestone or finish date?
Devastating	Visibility of a high failure possibility	impossible schedule [18], runaway projects [19]	(MQ4) What are the major actions leading to the termination of a project?

Additionally, the approximate values of numerical attributes were given as team size, duration, budget, and company size. Analysis software (ATLAS.ti) was used to map the logically related open codes to the code groups and code groups to axial codes. More details of the coding activities for private case studies can be found in Supplementary Material. B. The axial code analysis is presented in Table 2. The real names of the companies and projects were omitted to protect company information.

Open codes represent the direct behavioral responses of a team to a failure factor. The mental preparation and intermediate processes were not the main focus of interest. For example, in case study 1, MQ1 related to the problem expression as ''The large payments attracted many vendors to submit proposals without considering inconsistencies.''. It can be evaluated as ''wishful thinking'' or ''desire for money'' for the vendor. However, our open code definition was a combination of the failure factor or problem source as ''inconsistencies in analysis'' and the team action against the problem as ''ignoring''.

In axial coding, we attempted to assign the most comprehensive term covering specific variants of team behaviors to axial codes by considering the occurrence frequencies and logical relations between open codes. The open code count refers to the sum of the occurrence of open codes in case study analysis. It was calculated from Table 1-7 in Supplementary Material A.

Table 3 contains the transitions between axial codes that were obtained from the notes in the open code section of Tables 1-7 in Supplementary Material A. The number in a cell corresponds to the next axial code. The occurrence rate was calculated by dividing the number of occurrences by the total private case studies number (7). EP was omitted because all failed projects ended up with the same phase.

The selective coding revealed the generic process and phase definitions via axial codes. The narrative description of the storyline: The failure process starts with small slippages of the tasks; however, the team ignores warnings or deviations. As the harmful effects of deviations appear, the team

TABLE 2. Open and axial codes in private case studies.

Open code (occurrence number)	Axial code	Comment
Ignoring deviation (1) Ignoring the problems (1) Ignoring the requirements difficult to implement (1) Ignoring the uncertainties in the requirements (1) Ignoring warnings (3)	Ignoring warnings (IW)	Ignorance is the keyword expressing a team's action when they encounter a problem the first time or early stages. We chose "warnings" because it is a high-frequency and inclusive term. The alternative names can be "Ignoring deviation" or "Ignoring early warning signs".
Rejecting deviation (3) Rejecting the problems (2)	Rejecting deviations (RD)	The deviation is an umbrella term that refers to many negative situations including time delays, quality issues, and unsatisfied customer expectations.
Getting into panic (4) Losing control (2) Blaming team members (1) Firing or departure of the key members (1)	Getting into panic (GIP)	The usage frequency of getting into panic is higher than the other codes.
Continuing against failure (2) Escalating failure (4)	Escalating failure (EF)	Escalating failure provides a clearer definition of continuing investment in failed projects.
Losing customer interest (3) Losing support (2) Stopping support (3)	Losing support (LS)	The effect of losing support is detrimental to projects.
Failed partially (4) Terminating with failure (3)	Ending project (EP)	"End" is an umbrella term covering termination with failure and partial failure by exceeding limits.

TABLE 3. The phase transition matrix for private case studies.

	IW	RD	GIP	EF	LS
Case study 1	2	3	4	5	6
Case study 2	2	3	4	6	
Case study 3	3	4	2	5	6
Case study 4	3		4	5	6
Case study 5	2	3	4	5	6
Case study 6	5				6
Case study 7	2	4	5	3	6
The occurrence rate %	100	71,4	85,7	85,7	85,7

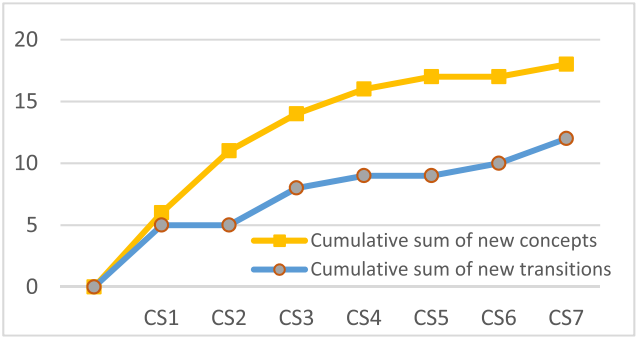


FIGURE 4. The saturation of concepts.

tries to reject the deviation first; however, when the deviation exceeds the limits, panic begins to catch the plan. After the committed finish date is missed, the project team attempts to save investments and continue the project against negative conditions. When corrective actions are not implemented on time, stakeholders lose interest and decrease their support; thus, termination is inevitable.

The cumulative sum of identified new concepts for each case study diminished quickly, as shown in Fig. 4. This indicates the maturity of the proposed model was high for phase definitions; therefore, adding further case studies would be unlikely to discover significant numbers of additional concepts [23]. However, the number of transitions between phases (axial codes) was more gradual, indicating a lower level of theoretical saturation. Thus, we further examined the transitions in evaluating the combined codes of private and public case studies using DSM and BBN methods.

B. SATURATING THE PHASE NAMES AND TRANSITIONS WITH PUBLIC CASE STUDIES

The main purpose of the public case study analysis with grounded theory was to further saturate the initial codes using the well-known, failed, and overbudget software project case study lists [6], [43]. Additionally, we searched the databases IEEE Xplore, ScienceDirect, ACM, and Google Scholar using the following combinations of keywords: software project failure, failed case study, failure process, phase of failure, and failure phase. The data collection principles were (1) using multiple sources of evidence, (2) creating a case study dataset, and (3) validating data [48]. The literature search resulted in a collection of more than 100 academic research papers, resources, and books, including case studies of software project failures. We investigated 15 projects with a failure history in the case study resources, as shown in Table 4. To represent the relationship between the gathered information and its context, a full set of data and transitions are provided in Supplementary Materials C and D, not to overextend the article.

Nine new open codes emerged from public case studies. The conceptual meaning of these open codes allowed us to map them to the existing axial codes as (IW - misunderstanding requirements, starting with unreachable requirements, ignoring constantly changing requirements, starting with known problems, ignoring too long plan issue), (RD - hiding deviation, continue as there is no problem, underestimating deviation), and (EF - indecision to continue or terminate, slipping scope or schedule constantly). Because the usage frequency of *losing control* was higher in public

TABLE 4. The open codes in public case studies.

	IW	RD	Losing control (LC)	EF	LS	EP
The CONFIG project	misunderstanding requirements [49], starting with known problems [25]	rejecting deviation, hiding deviation [25]	blaming (sales) team [25]	continuing against failure [49]	losing support [49], losing customer interest [25]	terminating with failure [25]
Denver Airport Baggage handling system	misunderstanding (importance and significance) requirements [6]	hiding deviation [50]	got into panic mode, (customer and vendor) blame each other [6]	continuing against failure [51], indecision to continue or terminate [6]	losing support [6].	partially failed [52]
Florida fiasco(welfare)	starting with unreachable requirements [6], ignoring the inconsistencies in the requirement analysis [53]	continue as there is no problem [6]	losing control, firing, or departure of the key members [6]			partially failed [6]
FAA Automation System	ignoring constantly changing requirements [6]	underestimating deviation [6]	losing control[6]	continue against problems [6]	losing support [6]	totally failed [6]
Confirm project	starting with known problems [6]	hiding deviation [6] rejecting deviation [54]	blame team members, firing or departure of the key members [54]	escalating failure[6]	losing (partner) support [6]	terminated with a lawsuit [6]
Kapor ON location Project	starting with known problems [6]	underestimating deviation [6]	losing control, conflict in team [6]	slipping schedule constantly (further and further) [6]	losing (partner) support [6]	totally terminated [6]
New Jersey Division of Motor Computer System	ignoring warnings [6]	continue as there is no problem [6] [55]		escalating failure [6]		totally failed (rewriting) [6]
BMBC e-procurement project	ignoring warning (signs) [56]	Rejecting deviation [56]	conflict in team [56]	continue against failure[56]	losing (user manager) support [56]	failed partially [56]
Taurus project	ignoring warning (signs) [49].	hiding deviation [7]	losing control[57] , [58]	slipping schedule constantly [57], continue against failure [59]	losing (major supplier support) [57]	totally failed [60]
BOLIT	ignoring the inconsistencies in the requirement analysis [61]	continue as there is no problem [61]		escalating failure [61]		totally failed [61]
CSIO portal	ignoring unreachable requirements [62], ignoring too long plan issue [63]	hiding deviation[64].	losing control [65]	indecision to continue or terminate [65], escalating failure [66]	losing support (a funding partners) [64], stopping support [66]	totally failed [66]
IS project in British Utilities	ignoring warning (signs) [10]	hiding deviation [10].	firing or departure of the key members, losing control [10].	indecision to continue or terminate [10]	losing customer interest [10]	totally failed [10]
NHS Connecting for Health	ignoring the inconsistencies in the requirement analysis [67], ignoring warning (signs) [68]	rejecting deviation [68]		continuing (some module) against failure [69]	losing support [67]	partially failed [67]
Polsag	ignoring warning (about feasibility) [70]	hiding deviation [70], [71]	conflict in team [70]	escalating failure [71]	losing support (of a supplier) [72]	totally failed [71]
Cover Oregon	ignoring warning [73], [74]	rejecting deviation [75], hiding deviation [74]	conflict in team [73]	slipping scope or schedule constantly [73]	losing support [76]	totally failed [76]

case studies, and it had a more comprehensive meaning to map with *conflict in team*, it was replaced with *getting into panic*.

Table 5 contains the transition information and occurrence rates of public case studies. The number of new open codes was relatively low. This indicates that our process definition was near saturation; therefore, an additional case study analysis would not produce significant differences.

C. QUANTITATIVE EVALUATION

The occurrence rates of the phases are shown in Fig. 5. They are higher than 80 percent for private or public case studies on average. The only exception is the occurrence of rejecting deviations phase in private case study analysis.

The total number of transitions between the phases is given by the DSM in Table 6. The matrix as a quantification scheme facilitates weighting the interactions relative to each other. n_{ij} is the total number of transitions from phase i to phase j and reveals the other phases followed by i. The marks below the diagonal are called feedback marks and the marks above the diagonal are called feedforward.

TABLE 5. The phase transition matrix for public case studies.

	IW	RD	LC	EF	LS
The CONFIG project	2	3	4	5	6
Denver Airport Baggage handling system	2	3	4	5	6
Florida fiasco(welfare)	2	3	6		
FAA Advanced Automation System	2	3	4	5	6
Confirm project	2	3	4	5	6
Kapor ON location Project	2	4	5	3	6
New Jersey Division of Motor new Computer System	2	4		6	
BMBC e-procurement project	2	3	4	5	6
Taurus project	2	3	4	5	6
BOLIT	2	4		6	
CSIO portal	2	3	4	5	6
IS project in British Utilities	2	3	5	6	4
NHS Connecting for Health	2	4		5	6
Polsag	3	4	2	5	6
Cover Oregon	2	3	4	5	6
The occurrence rate percentage	100	100	80.0	93.3	80.0

The probabilistic relationships of the transitions between the phases were evaluated using the BBN. The standard BBN representation contains a detailed table with each node providing the probabilities of each state of a variable.

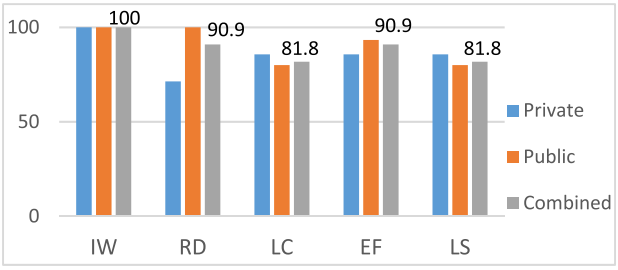


FIGURE 5. The total occurrence rates for case studies.

TABLE 6. DSM for transitions between phases.

	IW	RD	LC	EF	LS	EP
IW	0	18	3	0	1	0
RD	0	0	13	7	0	0
LC	0	2	0	12	3	1
EF	0	0	2	0	14	4
LS	0	0	0	1	0	17
EP	0	0	0	0	0	0

However, for simplicity, we represent only the joint probability of transition on the edges calculated using the following equations:

$$Probability\ of\ node\ i = P_i = \frac{n_i}{n} \tag{1}$$

$$Probability\ of\ transition\ i\,j = PT_{ij} = \frac{n_{ij}}{t_i} \tag{2}$$

where n is the total number of case studies and n_i is the number of occurrences of phase i. n_{ij} is the number of transitions from phase i to phase j, and t_i is the total number of transitions initiated from phase i. $PT_{(ij/i)}$ is the joint probability of a transition from phase i to phase j which is defined as:

$$PT(ij/i) = P(ij)\,xP(i) \tag{3}$$

Fig. 6 shows the BBN analysis results of the proposed failure process model. The nodes in the graph are phase names, while the edges represent the joint probabilities of the transitions between phases. The transition with the highest probability value between two phases is highlighted by a thick line to illustrate its importance. The feedback transitions are represented by dashed lines.

V. DISCUSSION

In this section, we discuss the outcomes of our work concerning the research questions and compare our findings with those of previous studies on project failure. We have defined the phase details and highlighted the threats to the validity of our conclusions.

A. ANSWERING THE RESEARCH QUESTIONS

RQ1: What are the critical and recurring team behaviors that lead a software project to failure phase by phase?

Answer 1: Our findings suggest that similar team behaviors are perceived in failed software projects, whereas the apparent failure symptoms are different. The phase definitions created by grouping and categorizing codes have high occurrence rates in the case studies as shown in Fig. 6. On average, similar phase definitions appeared in 89 percent (standard deviation 6.8 percent) of case studies.

RQ2: What are the transition patterns between the failure phases?

Answer 2: The joint probability of transitions indicates that the order of IW → RD → LC → EF → LS → EP is more frequent than that of other paths as shown in Fig. 6. The probabilities in this path have at least twice as much frequency as backward and forward jumps although some have low probability values. Thus, we accept this as the primary path for defining the failure process.

Our findings contribute to previous studies on building an integrated software-specific failure process model with standard phase and transition definitions. Table 7 summarizes the intersections and shared concepts between existing definitions from the literature and the proposed failure process; they indicate a reasonable consistency of content. However, existing software-specific anti-patterns have isolated definitions and the general failure process does not include some of them. For example, escalating failure and losing support phases did not exist in the mistake chain [8] while rejecting deviations was emphasized in the passive form as the mum effect [77] or hiding the truth [8]. The central role of these differences is the essential difficulties of software development that make measuring the correlation between outcomes, objectives, and progress difficult especially, for non-technical people. In other engineering disciplines, such as civil or mechanical, the team can see the whole product at once; hiding problems is not easy and may not result in extreme situations [78]. In a construction project, the visibility of the outcome causes a delay to be realized almost immediately [79]. Minimal technical knowledge can be sufficient to evaluate the overall progress.

B. DEFINITION OF SOFTWARE FAILURE PROCESS

The main outcome of this study is the definition of a process model for failed software projects. At the beginning of the process, early warning signs arise; however, there is no visible indication of failure. The project team does not understand the overall problem and *ignores warnings*. In the *rejecting deviations* phase, the significant deviation is visible to the team; however, the majority of members do not accept the importance of the problem and tend to mask the problem hoping to overcome it without attracting the scrutiny of executives and other stakeholders [2]. When deviation from the schedule exceeds the nominal limits, the team *loses control* and unconsciously attempts to rescue the project. *Escalating failure* occurs when the team continues to invest even through negative events and missed results. After a long unproductive escalation cycle, the project *loses the support* of clients or sponsors, causing its termination.

We prefer the term "*the software project failure process*" to express the process leading a project to failure. Another possible term may be inspired by nature, such as the waterfall

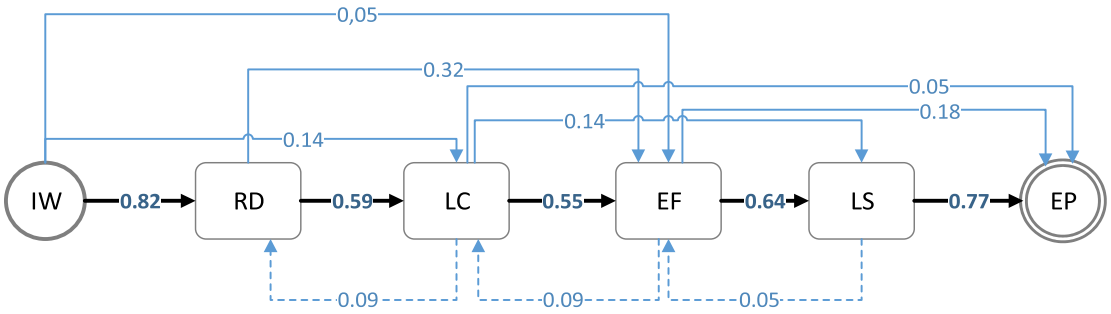


FIGURE 6. The probabilities of the transitions between failure phases.

TABLE 7. The proposed and existing failure phase definitions.

	IW →	RD →	LC →	EF →	LS
Mittelstaedt [8]	disbelief seriousness	attempt to hide the truth	sudden recognition out of control		
Jones [16]	missed alert	no one wants to give bad news	until the very day of the expected deployment		
Glass [6] (combined phases)			crunch mode[17], death march[18]	runaway projects[19]	
Keil and Mann [25]				escalating failure	
Blumen [37]					lost the support

process as "*the swamping process*". The more you struggle unconsciously, the more you sink until you drown. More importantly, you will lose your chance to rescue. "Sink" was also used to identify runaway projects in the literature [6].

1) IGNORING WARNINGS

This phase refers to a situation in which the software team underestimates or ignores warning signs and the initial emergence of deviations, leading to project failure in the future. Brooks [15] asked the question, "How does a large software project get to be one year late?" and answered it as "One day at a time! Incremental slippages on many fronts eventually accumulate to produce a large overall delay". All project tasks are linked in a complex decision network; thus, total failure can be initiated by the first wrongdoing in the tasks and/or decisions that aggregate and accumulate into more critical problems [24].

Early warning signs arise as predictions, cautions, or alerts to possible or impending problems. They provide an assessment of risks, future difficulties, and failures during project development [1]. They do not directly represent a major crisis but indicate a need for further investigation to prevent the start of a mistake series [10]. Lui and Chan [79] reported that the dynamic and multifactorial nature of many software project problems may prevent immediate or even timely identification of root causes. Moreover, the intangible nature of software makes it difficult to estimate the proportion of work that has been completed [77]. Project managers are often missing the appropriate response to early warnings in many cases [80].

2) REJECTING DEVIATIONS

In this phase, the project team intentionally rejects problems or withholds bad news about the project status, as the scale and visibility of deviation increase. The reluctance of people to report bad news about a troubled project is called the "Mum Effect" [77]. This could be a major contributor to the phenomenon of uncontrollable (runaway) software projects [81]. As a passive version of rejecting deviation, it may disrupt communication between team members [82] and occurs very frequently [82], [83]. Snow *et al.* [83] claimed that biased, generally optimistic status reporting occurs in more than 60 percent of projects.

3) LOSING CONTROL

The losing control phase represents the unconscious attempts of a software team to rescue a project after skipping an important milestone or finish date. It is similar to the crunch mode reported in the literature. Huang and Han [84] stated that as the date to deliver the slipping milestone approached, teams tried to compensate for the lost time by forcing a team to work more days and hours per week, developers sleeping on the floor for days on end. The exhaustion leads to more bugs that need to be fixed and causes wasting even more time. The delay may affect the management to initiate multiple forms of coping strategies aiming for the same goal [85]: ownership problems, no owner or more than one owner can create political fighting and convert a small problem into a big issue; moreover, team members may be firefighting with or even withholding their problems without considering how these problems might create other problems [79]. The team

members stop moving according to the plan and try to catch the schedule unconsciously by focusing on only their tasks.

4) ESCALATING FAILURE

Keil and Mann's [25] description of the escalating failure is parallel to our findings as ''continued commitment against negative information'' and ''an escalation cycle starts following a series of negative project events and the commitment of more resources when decision-makers neither decide to abandon the project nor take corrective actions despite unambiguous negative feedback''. Keil *et al.* [86] used it to explain ''troubled projects were continued instead of being abandoned or redirected''. According to Keil and Mann [87], 30-40 percent of all information system (software) projects involve some degree of project escalation, the average escalation time is 21 months, and less than 25 percent of the escalated projects are completed or implemented. The causes of escalation are more than simple mismanagement of projects and can be psychological, social, and organizational [87].

5) LOSING SUPPORT

This phase refers to losing support of customers, managers, or other stakeholders because the development process does not meet their requirements and expectations on time. The key reason is the loss of sponsorship support in 80 percent of project abandonment [37] similar to our findings. The management of stakeholders leads a project to success or failure [14]. The withdrawal of a sponsor or stakeholder already contributing to the project has a more detrimental effect on perceptions of project success than starting without a sponsor [88]. Without the strong support of the organization's managers and sponsor commitment, developers perceive little chance of project success [88], [89].

6) ENDING PROJECT

If there is no way for a project to be completed, the terminating project is natural and even healthy [90]. In this phase, the business goal shifts to terminate the project with minimal losses. The primary actions are preparing a termination plan, managing public relations crises, and learning from failures.

C. THREATS TO VALIDITY

In this section, we address increasing the validity of our work by minimizing the risk of potential threats. For dependability, we conducted a systematic and well-documented research process. Therefore, we have provided the raw data and analysis procedures with the intermediate results of the qualitative and quantitative analyses to ensure confirmability. Other researchers can evaluate the code, phase, and transition definitions from different perspectives using raw data and analysis notes to unveil new concepts.

The credibility of this research inherits some internal limitations from the grounded theory. Grounded theory might restrict the potential of the data and the creativity of the qualitative analyst, and it may be influenced by the personality, experience, or perspectives of researchers [41], [91]. To overcome this bias and increase credibility, we collected data from multiple sources using various methods. Individuals and companies withhold information about their failed project with feeling guilt or shame; hence, some details may have been missed. To overcome this problem, we examined 22 case studies (private 7 + public 15) that exceeded the ideal limit to build a theory defined between 4 and 10 [92]. In addition, we researched many papers or websites for each case study to gather data for cross-validation.

A threat to the transferability of our process definition was the effect of bias in case study selection and the data collection process. We increased diversity by researching projects of different scales and domains to mitigate this threat. Moreover, we saturated our process definition by adding public case study data and comparing them with private case study data. The comparative analysis facilitates a decrease in a threat related to the generalizability of the findings that appears the selected sample of case studies is not sufficient to represent different perspectives to be generally applicable [23]. We assessed that the effect of this threat decreased significantly in the final version of the process definition as relatively few new concepts arose in public case studies.

Another validation technique for results discovered using grounded theory is a literature search [41]. The parallel definitions in the literature and our process model indicate consistency between existing studies and our process model as demonstrated in Sections 5.1 and 5.2. The existing studies contain similar observations, symptoms, and outcomes; however, their definitions are isolated and not integrated into a failure process.

Employing an additional analytic strategy can produce credible and dependable research findings [91]. Thus, we supported grounded theory with quantitative DSM and BBN approaches. The quantitative methods highlighted the main path of failure process definition by analyzing the transition probabilities. However, the quantitative evaluation also revealed a significant number of feedbacks and feedforward jumps between RD, LC, and EF phases that may have weakened the failure process definition and external validity. The main reason behind this was the difficulty in finding the full chronology of failure stories because many studies have investigated failure causes in isolation [11]. Moreover, people, projects, and organization-specific reasons can differentiate the failure process. The exceptional feedback and feedforward transitions also appear in other development processes, and their probability values are too small to distort the failure process definition.

There are some other limitations to our study. First, informants' perceptions may have biased the collected data samples; although, we preferred to use multiple data sources for each case study. However, the addition of more failure cases may not have eliminated the perceptual effect. To reduce the effect of this limitation, surveys can be conducted to define the failure process through expert validation. Second, because there is a lack of research and datasets directly investigating

the relationships between team actions behind failure factors in the software engineering literature, for some case studies, we had to collect data from unusual sources such as newspapers or websites for scientific research to generate a novel chronological dataset on software project failure. This may have adversely affected the quality of this study.

VI. CONCLUSION

The principal contribution of this study is a failure process model for software projects. The model reveals critical team behaviors and transitions among them leading a software project to failure. It was derived from a comprehensive case study search, qualitative grounded theory analysis, quantitative DSM, and BBN methods.

The important theoretical and practical implications of the results can raise the awareness of researchers, software professionals, managers, and other stakeholders to evaluate a more realistic picture of project failures, failure milestones, real progress, and estimate future incidents. From a theoretical perspective, the researcher can use the methods, intermediate analysis notes, and dataset in our study as inputs for new research to further understand software failures and develop methods for successful projects. Additionally, thinking failure as a process can create a scientific background to develop and offer models for improving process monitoring and control tools. From a practical perspective, our process model, which has simple behavioral definitions helps software teams and managers to identify and manage the deviations and risks in software projects. Thus, they can handle and prevent the spread of failures in a project's progress by an in-depth understanding of initiating team action.

Although we believe that the results of this study will be useful to understand the progress of a failed software project, there are some requirements for further research on failure in software development process methodologies and phases. The first improvement would be to match failure phases with their solutions that support important insights for identifying solution methods. The second improvement is systematically elaborating the connection of the software project failure process to various software development process models, such as waterfall, incremental, agile models, and their phase definitions. Another future improvement is a domain-specific analysis of the failure process. There are important domains that can be investigated specifically, such as the back-end, system, and real-time software. Additionally, predictive modeling using computational intelligence for the effects of failure factors, personal roles, and personal characteristics on failure phases can reveal useful concepts. It will be interesting to extend this study by examining the subphases of the primary failure phases.

REFERENCES

[1] L. A. Kappelman, R. McKeeman, and L. Zhang, "Early warning signs of it project failure: The dominant dozen," *Inf. Syst. Manage.*, vol. 23, no. 4, pp. 31–36, Sep. 2006, doi: 10.1201/1078.10580530/46352.23.4.20060901/95110.4.

[2] C. L. Iacovou and A. S. Dexter, "Turning around runaway information technology projects," *California Manage. Rev.*, vol. 46, no. 4, pp. 68–88, Jul. 2004, doi: 10.2307/41166275.

[3] G. Standish, "Chaos report on software projects," Project Smart, Standish Group, Boston, MA, USA, 2014.

[4] Project Management Solutions. (2011). *Strategies for Project Recovery: A PM Solutions Research Report*. [Online]. Available: https://www.pmsolutions.com/audio/Strategies_for_Project_Recovery_Research_Report.pdf

[5] R. Schmidt, K. Lyytinen, M. Keil, and P. Cule, "Identifying software project risks: An international Delphi study," *J. Manage. Inf. Syst.*, vol. 17, no. 4, pp. 5–36, 2001, doi: 10.1080/07421222.2001.11045662.

[6] R. L. Glass, *Software Runaways: Monumental Software Disasters*. Upper Saddle River, NJ, USA: Prentice-Hall, 1998.

[7] R. N. Charette, "Why software fails [software failure]," *IEEE Spectr.*, vol. 42, no. 9, pp. 42–49, Sep. 2005, doi: 10.1109/MSPEC.2005.1502528.

[8] R. Mittelstaedt, *Will Your Next Mistake Be Fatal: Avoiding the Chain of Mistakes That Can Destroy Your Organization*. Upper Saddle River, NJ, USA: Prentice-Hall, 2004.

[9] J. Verner, J. Sampson, and N. Cerpa, "What factors lead to software project failure?" in *Proc. 2nd Int. Conf. Res. Challenges Inf. Sci.*, Jun. 2008, pp. 71–80, doi: 10.1109/RCIS.2008.4632095.

[10] G. Pan, S. L. Pan, and M. Newman, "Managing information technology project escalation and de-escalation: An approach-avoidance perspective," *IEEE Trans. Eng. Manage.*, vol. 56, no. 1, pp. 76–94, Feb. 2009.

[11] T. O. A. Lehtinen, M. V. Mäntylä, J. Vanhanen, J. Itkonen, and C. Lassenius, "Perceived causes of software project failures—An analysis of their relationships," *Inf. Softw. Technol.*, vol. 56, no. 6, pp. 623–643, 2014, doi: 10.1016/j.infsof.2014.01.015.

[12] M. A. Ibraigheeth and S. A. Fadzli, "Software project failures prediction using logistic regression modeling," in *Proc. 2nd Int. Conf. Comput. Inf. Sci. (ICCIS)*, Oct. 2020, doi: 10.1109/ICCIS49240.2020.9257648.

[13] D. A. Tamburri, F. Palomba, and R. Kazman, "Success and failure in software engineering: A followup systematic literature review," *IEEE Trans. Eng. Manag.*, vol. 68, no. 2, pp. 599–611, Apr. 2021, doi: 10.1109/TEM.2020.2976642.

[14] G. P. Sudhakar, "A model of critical success factors for software projects," *J. Enterprise Inf. Manage.*, vol. 25, no. 6, pp. 537–558, 2012, doi: 10.1108/17410391211272829.

[15] B. F. P. Brooks, *The Mythical Man-Month, Essays on Software Engineering*. Reading, MA, USA: Addison-Wesley, 1995.

[16] C. Jones, "Patterns of large software systems: Failure and success," *Computer*, vol. 28, no. 3, pp. 86–87, Mar. 1995.

[17] J. Boddie, *Crunch Mode: Building Effective Systems on a Tight Schedule*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1987.

[18] E. Yourdon, *Death March*. Upper Saddle River, NJ, USA: Prentice-Hall, 2004.

[19] *Report on IT Runaway Systems*, KPMG, Amstelveen, The Netherlands, 1995.

[20] M. Jørgensen, T. Dybå, K. Liestøl, and D. I. K. Sjøberg, "Incorrect results in software engineering experiments: How to improve research practices," *J. Syst. Softw.*, vol. 116, pp. 133–145, Jun. 2016, doi: 10.1016/j.jss.2015.03.065.

[21] P. Savolainen, J. J. Ahonen, and I. Richardson, "Software development project success and failure from the supplier's perspective: A systematic literature review," *Int. J. Project Manage.*, vol. 30, no. 4, pp. 458–469, May 2012, doi: 10.1016/j.ijproman.2011.07.002.

[22] B. W. Boehm, "Software risk management: Principles and practices," *IEEE Softw.*, vol. 8, no. 1, pp. 32–41, Jan. 1991, doi: 10.1109/52.62930.

[23] T. A. Kroeger, N. J. Davidson, and S. C. Cook, "Understanding the characteristics of quality for software engineering processes: A grounded theory investigation," *Inf. Softw. Technol.*, vol. 56, no. 2, pp. 252–271, 2014, doi: 10.1016/j.infsof.2013.10.003.

[24] T. N. Nguyen, "Software project management towards failure avoidance," in *Proc. 9th Int. Conf. Softw. Eng. Appl. (ICSOFT-EA)*, Aug. 2014, pp. 560–567.

[25] M. Keil, "Pulling the plug: Software project management and the problem of project escalation," *MIS Quart.*, vol. 19, no. 4, pp. 421–447, 1995, doi: 10.2307/249627.

[26] I. Stamelos, "Software project management anti-patterns," *J. Syst. Softw.*, vol. 83, no. 1, pp. 52–59, Jan. 2010, doi: 10.1016/j.jss.2009.09.016.

[27] B. G. Glaser and A. L. Strauss, *The Discovery of Grounded Theory: Strategies for Qualitative Research*. London, U.K.: Aldine, 1967.

[28] P. E. W. Onions, "Grounded theory applications in reviewing knowledge management literature," in *Proc. Leeds Metrop. Univ. Innov. North Res. Conf.*, 2006, pp. 1–20.
[29] M. Halaweh, C. Fidler and S. McRobb, "Integrating the grounded theory method and case study research methodology within IS research: A possible 'road map,'" in *Proc. 29th Int. Conf. Inf. Syst.*, 2008. [Online]. Available: http://aisel.aisnet.org/icis2008/165
[30] Y. S. Lincoln and E. G. Guba, *Naturalistic Inquiry*. Newbury Park, CA, USA: Sage, 1985.
[31] J. Corbin and A. Strauss, *Basics of qualitative research: Techniques and procedures for Developing Grounded Theory*. Newbury Park, CA, USA: Sage, 2014.
[32] S. Friese. (2009). *Working Effectively With Atlas.TI*. [Online]. Available: http://downloads.atlasti.com/library/Friese_2009-09_1.pdf
[33] W. D. Fernández, "The grounded theory method and case study data in IS research: Issues and design," in *Information Systems Foundations Workshop: Constructing and Criticising*. Canberra, QLD, Australia: ANU Press, 2004, pp. 43–59.
[34] J. K. Pinto and S. J. Mantel, Jr., "The causes of project failure," *IEEE Trans. Eng. Manag.*, vol. 37, no. 4, pp. 269–276, Nov. 1990, doi: 10.1109/17.62322.
[35] A. Hawari and R. Heeks, "Explaining ERP failure in a developing country: A Jordanian case study," *J. Enterprise Inf. Manage.*, vol. 23, no. 2, pp. 135–160, Feb. 2010, doi: 10.1108/17410391011019741.
[36] J. Mishra, *Software Engineering*. New Delhi, India: Pearson, 2011.
[37] R. Blumen, "Jürgen Laartz and Alexander Budzier on why large IT projects fail," *IEEE Softw.*, vol. 33, no. 4, pp. 117–120, Jul./Aug. 2016, doi: 10.1109/MS.2016.102.
[38] R. Frese and V. Sauter, "Improving your odds for software project success," *IEEE Eng. Manag. Rev.*, vol. 42, no. 4, pp. 125–131, Fourth 2014, doi: 10.1109/EMR.2014.6966952.
[39] M. Jørgensen, "A survey on the characteristics of projects with success in delivering client benefits," *Inf. Softw. Technol.*, vol. 78, pp. 83–94, Oct. 2016, doi: 10.1016/j.infsof.2016.05.008.
[40] A. D. Andrade, "Interpretive research aiming at theory building: Adopting and adapting the case study design," *Qualitative Rep.*, vol. 14, no. 1, pp. 42–60, 2009.
[41] H. K. Gidey, D. Marmsoler, and J. Eckhardt, "Grounded architectures: Using grounded theory for the design of software architectures," in *Proc. IEEE Int. Conf. Softw. Archit. Workshops (ICSAW)*, Apr. 2017, pp. 141–148, doi: 10.1109/ICSAW.2017.41.
[42] R. Verdecchia, P. Kruchten, P. Lago, and I. Malavolta, "Building and evaluating a theory of architectural technical debt in software-intensive systems," *J. Syst. Softw.*, vol. 176, Jun. 2021, Art. no. 110925, doi: 10.1016/j.jss.2021.110925.
[43] Wikipedia. (2020). *List of Failed and Overbudget Custom Software Projects*. Accessed: Feb. 5, 2020. [Online]. Available: https://en.wikipedia.org/wiki/List_of_failed_and_overbudget_custom_software_projects
[44] A. Alami, "The UK e-borders project failure," *PM World J.*, vol. 5, no. 3, pp. 1–14, 2016.
[45] T. R. Browning, "Applying the design structure matrix to system decomposition and integration problems: A review and new directions," *IEEE Trans. Eng. Manag.*, vol. 48, no. 3, pp. 292–306, Aug. 2001, doi: 10.1109/17.946528.
[46] D. Heckerman, "Bayesian networks for data mining," *Data Mining Knowl. Discovery* vol. 1, no. 1, pp. 79–119, 1997, doi: 10.1023/A:1009730122752.
[47] N.-T. Nguyen, Q.-T. Huynh, and T.-H.-G. Vu, "A Bayesian critical path method for managing common risks in software project scheduling," in *Proc. 9th Int. Symp. Inf. Commun. Technol. (SoICT)*, 2018, pp. 382–388.
[48] J. M. Verner and L. M. Abdullah, "Exploratory case study research: Outsourced project failure," *Inf. Softw. Technol.*, vol. 54, no. 8, pp. 866–886, 2012, doi: 10.1016/j.infsof.2011.11.001.
[49] K. Lyytinen and D. Robey, "Learning failure in information systems development," *Inf. Syst. J.*, vol. 9, no. 2, pp. 85–101, Apr. 1999, doi: 10.1046/j.1365-2575.1999.00051.x.
[50] J. Swartz, "Simulating the Denver airport automated baggage system," *Dr. Dobb's J.*, vol. 22, no. 1, pp. 56–62, 1997.
[51] S. Dalal and R. Chhillar, "Case studies of most common and severe types of software system failure," *Int. J. Adv. Res. Comput. Sci. Softw. Eng.*, vol. 2, no. 8, pp. 341–347, 2012.
[52] K. R. Linberg, "Software developer perceptions about software project failure: A case study," *J. Syst. Softw.*, vol. 49, no. 2, pp. 177–192, 1999, doi: 10.1016/S0164-1212(99)00094-1.
[53] S. C. Rollins and R. Lanza, *Essential Project Investment Governance and Reporting: Preventing Project Fraud and Ensuring Sarbanes-Oxley Compliance*. FL, USA: J Ross Publishing, 2005.
[54] E. Oz, "When professional standards are lax: The CONFIRM failure and its lessons," *Commun. ACM*, vol. 37, no. 10, pp. 29–43, Oct. 1994, doi: 10.1145/194313.194319.
[55] P. A. Strassmann, *The Business Value of Computers: An Executive's Guide*. New Canaan, CT, USA: Information Economics Pr, 1990.
[56] G. S. C. Pan, S. L. Pan, and D. Flynn, "De-escalation of commitment to information systems projects: A process perspective," *J. Strategic Inf. Syst.*, vol. 13, no. 3, pp. 247–270, Sep. 2004, doi: 10.1016/j.jsis.2004.08.001.
[57] H. Drummond, "The politics of risk: Trials and tribulations of the taurus project," *J. Inf. Technol.*, vol. 11, no. 4, pp. 347–357, Dec. 1996.
[58] H. Drummond, "Are we any closer to the end? Escalation and the case of Taurus," *Int. J. Project Manage.*, vol. 17, no. 1, pp. 11–16, 1999, doi: 10.1016/S0263-7863(97)00074-4.
[59] M. Keil and D. Robey, "Turning around troubled software projects: An exploratory study of the deescalation of commitment to failing courses of action," *J. Manage. Inf. Syst.*, vol. 15, no. 4, pp. 63–87, 1999, doi: 10.1080/07421222.1999.11518222.
[60] M. H. B. Afzal, "Large scale IT projects: Study and analysis of failures and winning factors," *IETE Tech. Rev.*, vol. 31, no. 3, pp. 214–219, May 2014, doi: 10.1080/02564602.2014.906862.
[61] T. Cegrell. (2010). *BOLIT*. [Online]. Available: https://cio.idg.se/2.1782/1.326833/darfor-floppade-projektentre-svenska-it-fiaskon-under-lupp
[62] C. Harris. (2006). *2006 I.T. FOCUS: Reality Check*. Accessed: Dec. 5, 2019. [Online]. Available: https://web.archive.org/web/20150402130915/http://www.canadianunderwriter.ca/news/2006-i-t-focus-reality-check/1000202197/
[63] K. Westera, "CSIO portal update: Focus on Critical Mass," Canadian Underwriter, 2002.
[64] D. Glasgow. (2006). *CSIO Portal Abandoned Due to Lack of Insurer Support and Availability of Other Solutions*. Insurance Portal. [Online]. Available: https://insurance-portal.ca/article/csio-portal-abandoned-due-to-lack-of-insurer-support-and-availability-of-other-solutions/
[65] Canadian Underwriter. (2003). *CSIO Role in Future Portal Development Uncertain*. Can. Underwriter. Accessed: Jan. 29, 2020. [Online]. Available: https://www.canadianunderwriter.ca/insurance/csio-role-in-future-portal-development-uncertain-1000008354/
[66] *CSIO Closes Door on Portal Project*, Can. Underwriter, Toronto, ON, Canada, 2006.
[67] G. Dhillon and J. Backhouse, "Risks in the use of information technology within organizations," *Int. J. Inf. Manage.*, vol. 16, no. 1, pp. 65–74, 1996, doi: 10.1016/0268-4012(95)00062-3.
[68] R. Francis, *Independent Inquiry Into Care Provided by Mid Staffordshire NHS Foundation Trust January 2005-March 2009*, vol. 375. London, U.K.: The Stationery Office, 2010.
[69] G. Southon, C. Sauer, and K. Dampney, "Lessons from a failed information systems initiative: Issues for complex organisations," *Int. J. Med. Inform.*, vol. 55, no. 1, pp. 33–46, 1999, doi: 10.1016/S1386-5056(99)00018-0.
[70] S. Lauesen, "Damage and damage causes in large government IT projects," IT Univ. Copenhagen, Copenhagen, Denmark, 2018.
[71] *Extract From the Report to the Public Accounts Committee on the Danish Police's IT System POLSAG March*, Rigsrevisionen, Copenhagen, Denmark, 2013.
[72] I. Reporters. (2012). *Outsourcer Facing Challenges on Multiple Fronts*. Computerworld U.K. Accessed: Dec. 21, 2019. [Online]. Available: https://www.cio.co.U.K./it-strategy/large-csc-project-dumped-amid-growing-crisis-3335166/
[73] D. Lane. (2014). *'We Look Like Fools:' A History of Cover Oregon's Failure*. Investigators2. Accessed: Jan. 9, 2020. [Online]. Available: https://web.archive.org/web/20150128010047/http://www.katu.com/news/investigators/We-look-like-fools-A-history-of-Cover-Oregons-failure-239699521.html
[74] N. Budnick. (2016). *Documents Damning on Oracle's Cover Oregon Release*. Portland Tribune. Accessed: Jan. 9, 2020. Accessed: Jan. 9, 2020. [Online]. Available: https://pamplinmedia.com/pt/9-news/294405-171739-documents-oracle-doesnt-want-you-to-read-
[75] G. Friedman. (2016). *Cover OregonKitzhaber, Oracle Respond to Report Critical of Cover Oregon*. Statesman J. Accessed: Jan. 9, 2020. [Online]. Available: https://www.statesmanjournal.com/story/news/politics/2016/05/25/congressional-panel-releases-critical-cover-oregon-report/84901000/

[76] S. Gallaghe. (2014). *Oregon Attorney General Sues Oracle for 'Racketeering Activity'*. ArsTechnica. Accessed: Jan. 9, 2020. [Online]. Available: https://arstechnica.com/tech-policy/2014/08/oregon-attorney-general-sues-oracle-for-racketeering-activity/

[77] H. J. Smith, M. Keil, and G. Depledge, "Keeping mum as the project goes under: Toward an explanatory model," *J. Manage. Inf. Syst.*, vol. 18, no. 2, pp. 189–227, Oct. 2001, doi: 10.1080/07421222.2001.11045677.

[78] S. Ramingwong and L. Ramingwong, "A tale behind mum effect," *Int. J. Inf. Syst. Project Manage.*, vol. 1, no. 3, pp. 47–58, 2013, doi: 10.12821/ijispm010303.

[79] K. M. Lui and K. C. C. Chan, "Rescuing troubled software projects by team transformation: A case study with an ERP project," *IEEE Trans. Eng. Manag.*, vol. 55, no. 1, pp. 171–184, Feb. 2008, doi: 10.1109/TEM.2007.912933.

[80] S. Haji-Kazemi, B. Andersen, and O. J. Klakegg, "Barriers against effective responses to early warning signs in projects," *Int. J. Project Manage.*, vol. 33, no. 5, pp. 1068–1083, Jul. 2015, doi: 10.1016/j.ijproman.2015.01.002.

[81] B. C. Y. Tan, H. J. Smith, and M. Keil, "Reporting bad news about software projects: Impact of organizational climate and information asymmetry in an individualistic and a collectivistic culture," *IEEE Trans. Eng. Manage.*, vol. 50, no. 1, pp. 65–77, Feb. 2003. [Online]. Available: http://search.ebscohost.com/login.aspx?direct=true&db=epref&AN=ITEM.EJ.FE.TAN.RBNASP

[82] J. Natovich, R. Natovich, and Z. Derzy, "Withholding bad news in information technology projects: The effect of positive psychology," in *Proc. 15th Pacific Asia Conf. Inf. Syst. Qual. Res. Pacific (PACIS)*, 2011, pp. 139–150.

[83] A. P. Snow, M. Keil, and L. Wallace, "The effects of optimistic and pessimistic biasing on software project status reporting," *Inf. Manage.*, vol. 44, no. 2, pp. 130–141, Mar. 2007, doi: 10.1016/j.im.2006.10.009.

[84] S.-J. Huang and W.-M. Han, "Exploring the relationship between software project duration and risk exposure: A cluster analysis," *Inf. Manage.*, vol. 45, no. 3, pp. 175–182, Apr. 2008, doi: 10.1016/j.im.2008.02.001.

[85] M. S. Granlien, J. Pries-Heje, and R. Baskerville, "Project management strategies for prototyping breakdowns," in *Proc. 42nd Hawaii Int. Conf. Syst. Sci. (HICSS)*, Jan. 2009, pp. 1–10, doi: 10.1109/HICSS.2009.357.

[86] M. Keil, A. Rai, J. Ellen, C. Mann, and G. P. Zhang, "Why software projects escalate: The importance of project management constructs," *IEEE Trans. Eng. Manage.*, vol. 50, no. 3, pp. 251–261, Aug. 2003.

[87] M. Keil and J. Mann, "Understanding the nature and extent of IS project escalation: Results from a survey of IS audit and control professionals," in *Proc. 13th Hawaii Int. Conf. Syst. Sci.*, vol. 3, Jan. 1997, pp. 139–148, doi: 10.1109/HICSS.1997.661582.

[88] J. D. Procaccino, J. M. Verner, S. Overmyer, and M. E. Darter, "Case study: Factors for early prediction of software development success," *Inform. Softw. Tech.*, vol. 44, no. 1, pp. 53–62, Jan. 2002, doi: 10.1016/S0950-5849(01)00217-8.

[89] D. Viskovic, M. Varga, and K. Curko, "Bad practices in complex IT projects," in *Proc. 30th Int. Conf. Inf. Technol. Interface (ITI)*, Jun. 2008, pp. 301–306, doi: 10.1109/ITI.2008.4588425.

[90] B. Boehm, "Project termination doesn't equal project failure," *Computer*, vol. 33, no. 7, pp. 94–96, Sep. 2000, doi: 10.1109/2.868706.

[91] M. Mehmetoglu and L. Altinay, "Examination of grounded theory analysis with an application to hospitality research," *Int. J. Hospitality Manage.*, vol. 25, no. 1, pp. 12–33, Mar. 2006, doi: 10.1016/j.ijhm.2004.12.002.

[92] K. M. Eisenhardt, "Building theories from case study research," *Acad. Manage. Rev.*, vol. 14, no. 4, pp. 532–550, 1989.

ALİ NİZAM was born in Fatih, İstanbul, Turkey, in 1976. He received the B.S. degree in electronic engineering from Yıldiz Technical University, İstanbul, in 1997, and the M.S. and Ph.D. degrees in electronic-biomedical engineering from İstanbul Technical University, İstanbul, in 2000 and 2009, respectively.

From 1997 to 2011, he worked at ISKI as a Software Engineer, the Project Manager, and the Management Information Systems Manager. Since 2011, he has been an Assistant Professor with the Computer Engineering Department, Fatih Sultan Mehmet Vakif University, İstanbul. He is the author of four books, one chapter, and seven articles. His research interests include software engineering, relational database concepts, and data science.

Dr. Nizam's awards and honors include the Turkey Academy of Science (TÜBA) and the University Textbooks Award Program Best Original Book Award with Software Project Management Book.