

External Factors Influencing Software Developer Behavior and Their Contribution to Software Errors: A Phenomenological Study

Rahul Azmeera ^{1,✉}, James C. Hyatt ¹

¹. Department of Information Technology, University of the Cumberlands, Williamsburg, USA

Received: July 22, 2026 | Review began: July 29, 2026 | Review ended: September 07, 2026 | Published: October 15, 2026

© Copyright 2026

This is an open access article distributed under the terms of the Creative Commons Attribution License CC-BY 4.0., which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

Abstract

This study investigates external factors shaping software developer behavior and predisposing them to introduce errors, an area receiving limited scholarly attention compared to technical root causes of defects. Software defects persist despite advances in development tooling, process maturity, and automated quality assurance. This study draws on the Technology-Organization-Environment framework to explore the organizational and environmental conditions underlying defect introduction. A qualitative phenomenological study was conducted with 15 professional software developers in the United States, each with 5 or more years of cross-domain experience. Semi-structured interviews were conducted and analyzed by six-stage thematic analysis. Seven thematic categories of external influence were identified: (T1) time constraints and dynamic requirements, (T2) communication barriers and team dependencies, (T3) knowledge gaps and testing strategies, (T4) resource constraints and industry awareness, (T5) development modalities and architecture, (T6) legacy systems, infrastructure, and security, and (T7) financial constraints and leadership vision. A total of 112 coded comments were reported, theme T1 was raised by 11 of 15 participants (73.3%) with 42 references and this was the major reported theme, followed by themes T3 and T4 reported by 8 participants (53.3%) with 15 and 10 references. Theme T2 was identified by 7 participants (46.7%) with 18 references. Theme T5 was identified by 6 participants (40%) with 11 references and theme T7 was identified by 5 participants (33.3%) with 11 references. Theme T6 was least identified with 5 references by 4 participants (26.7%). Beyond their relative frequency, these counts indicate the breadth of concerns across participants rather than a ranking of causal importance. The themes collectively describe how organizational and environmental conditions, rather than individual counts alone, shape developers' experience of error introduction. The study provides an integrated, evidence-based account of external pressures leading developers to introduce defects, extending prior research on organizational drivers of software quality. Software defect introduction is substantially shaped by organizational and environmental conditions rather than individual developer capability alone, requiring systemic responses from organizations to improve software reliability.

Categories: Software Development, Software Quality Assurance (SQA), Software Testing

Keywords: developer experience, human factors, organizational factors, phenomenological research, qualitative methods, software defects, software development lifecycle, software quality, toe framework

Introduction

Software now functions as critical infrastructure, deeply embedded in healthcare delivery, financial systems, transportation control, emergency response, and everyday consumer technology. The societal stakes of software failure have grown accordingly; a defect that previously caused a minor inconvenience can now result in financial loss, regulatory liability, or harm to human life in safety-critical domains [1], [2].

How to cite this article:

Azmeera R, Hyatt J C (October 15, 2026) External Factors Influencing Software Developer Behavior and Their Contribution to Software Errors: A Phenomenological Study. Cureus J Comput Sci 3 : es44389-026-00326-1. DOI <https://doi.org/10.7759/s44389-026-00326-1>

The software engineering community has invested extensively in defect prevention over several decades. Structured development methodologies, continuous integration and delivery pipelines, static analysis tools, formal code review processes, and comprehensive test automation frameworks are now widely deployed. Yet, software defect rates have not declined in proportion to these investments [3]. The gap between process maturity and defect prevalence suggests that existing models of software quality remain incomplete.

A substantial part of this gap concerns the human element. Software is produced by developers who operate within organizations subject to deadlines, dependent on teammates, constrained by budgets, guided by leaders, and shaped by architectural decisions made long before any given developer joined the project. These contextual forces are largely outside the developer's direct control, yet they substantially determine the conditions under which quality work is possible [4], [5].

This study employs a phenomenological qualitative methodology, which centers the first-person accounts of practitioners. This approach is well-suited to inquire into lived professional experience, where the goal is conceptual depth rather than statistical estimation [6]. The resulting analysis builds directly on the doctoral research of Azmeera [7], who documented the consequences of escalating software errors for product functionality and identified the need for deeper investigation into the factors that cause those errors.

Findings are organized using the Technology-Organization-Environment (TOE) framework [8], which provides a principled structure for categorizing the technological, organizational, and environmental forces that act on software development teams. This paper provides an integrated, evidence-based account of the external pressures that lead developers to introduce defects, advancing the understanding for practitioners seeking to reduce errors, managers seeking to create better working conditions, and researchers building the evidence base for software quality as a sociotechnical discipline.

Background and literature review

Conceptualizing Software Errors

In software engineering, a software error, also called as a defect or fault, occurs when a system's observable behavior deviates from its specified requirements [9]. Defects appear across a three-level severity range. At the lowest level are cosmetic anomalies, such as visual inconsistencies or minor formatting irregularities, which do not impair any functional behavior. At the intermediate level are partial functional failures that reduce the quality of the user experience without blocking core workflows. At the highest level are critical defects, application crashes, data corruption, security breaches, or incorrect outputs that cause failures in dependent systems [10].

The organizational consequences of high-severity defects extend well beyond the immediate technical failure. Azmeera [7] documented that escalating defect rates degrade product functionality, erode user trust, disrupt business operations, and damage the reputation of the responsible organization. In safety-sensitive domains including avionics, autonomous vehicles, medical devices, and power grid management, the consequences can be fatal.

Human and Organizational Factors in Software Quality

A growing body of scholarship has established that software quality is not determined by technical factors alone. Team composition and experience, project management practices, interpersonal dynamics, and organizational communication culture are all associated with measurable differences in defect rates [11]. Research on developer cognition has demonstrated that cognitive load, the total mental demand placed on a developer at any given moment, directly affects error susceptibility, with excessive load reliably producing higher error rates [5].

Studies examining the psychological and interpersonal dimensions of software work have found that developer personality, stress levels, and team cohesion predict not only productivity but also the types and frequencies of defects that reach production [12]. These findings collectively point to organizational and environmental conditions, not just individual technical skill, as critical leverage points for improving software quality.

The Technology-Organization-Environment Framework

How to cite this article:

Azmeera R, Hyatt J C (October 15, 2026) External Factors Influencing Software Developer Behavior and Their Contribution to Software Errors: A Phenomenological Study. *Cureus J Comput Sci* 3 : es44389-026-00326-1. DOI <https://doi.org/10.7759/s44389-026-00326-1>

The TOE framework was developed by Tornatzky and Fleischer [8] to explain how organizations adopt and integrate new technologies by attending to three interacting dimensions. The technological dimension covers the full range of tools, platforms, and systems that are both deployed internally and available externally that define what is technically achievable. The organizational dimension addresses internal structural features: team size and skill composition, available resources, communication norms, and management processes. The environmental dimension captures conditions outside the organization itself: market pressures, competitive dynamics, regulatory requirements, and the broader industry context.

Originally developed for technology adoption research, the TOE framework has since been applied to a range of software engineering questions, including agile method adoption, DevOps practices, and cloud infrastructure integration [13]. Its multi-level structure makes it particularly effective for organizing the diverse categories of external influence that emerge in the qualitative studies of developer experience.

Phenomenological Research in Software Engineering

Phenomenology as a research methodology focuses on understanding human experience from the perspective of those who live it, capturing what a phenomenon means to participants in their own words and context [14]. Within software engineering research, phenomenological and interpretive qualitative approaches have illuminated practitioner experiences of code review, pair programming, technical debt management, and developer burnout in ways that quantitative methods alone cannot capture [15].

The key strength of phenomenology in this context is the ability to reveal the meaning that practitioners attach to their experiences, rather than simply counting behaviors or outcomes. This is especially valuable when the phenomenon of interest - in this case, the introduction of software errors - is not a single measurable event but an outcome shaped by complex, context-dependent decisions made under pressure.

Research Gap

Existing literature has examined specific stressors such as time pressure [16] and communication breakdown [17] in relative isolation, and broader empirical studies on agile and DevOps adoption [13], knowledge management practices [17], and behavioral software engineering more generally [15] have independently established that organizational and interpersonal conditions shape defect-relevant outcomes. This body of research, together with the doctoral research of Azmeera [7] documenting the downstream consequences of escalating software errors for product functionality, establishes that individual external stressors matter but leaves their combined, lived effect on developers largely unexamined as a single, integrated phenomenon. The doctoral dissertation by Azmeera [7] provided the first comprehensive phenomenological account of how escalating software errors affect product functionality and identified the underlying causes of those errors as a priority area for further research. The present study responds to this gap by providing a holistic, thematically integrated analysis of the multiple external forces that together create error-prone development conditions; while individual stressors such as time pressure and communication breakdown are independently documented, their combined treatment within a single phenomenological, multi-factor account remains limited.

Case Presentation

Research question

This investigation is organized around one primary research question:

What external factors do software developers perceive as influencing their tendency to introduce software errors within organizational settings?

The deliberate breadth of this question is methodologically appropriate for phenomenological inquiry, which allows conceptual categories to emerge from participant accounts rather than imposing them in advance. The focus on external factors distinguishes this inquiry from the studies of individual developer attributes such as skill level, personality, or

How to cite this article:

Azmeera R, Hyatt J C (October 15, 2026) External Factors Influencing Software Developer Behavior and Their Contribution to Software Errors: A Phenomenological Study. *Cureus J Comput Sci* 3 : es44389-026-00326-1. DOI <https://doi.org/10.7759/s44389-026-00326-1>

cognitive style and highlights the organizational, structural, and environmental forces that operate beyond any individual developer's direct control.

Methodology

Research Design

A descriptive phenomenological research design was adopted. This approach, rooted in the philosophical tradition of setting aside the researcher's own assumptions so that the participant's experience can be understood on its own terms [6], was chosen because the research objective - understanding what external conditions developers associate with software error introduction - requires access to the subjective, interpretive dimension of professional experience rather than behavioral observation or statistical measurement. Qualitative inquiry is particularly appropriate when the phenomenon under investigation is embedded in complex social and organizational contexts that cannot be reduced to numerical variables [18].

Participant Selection and Sampling

Participants were selected through purposive sampling, a strategy that prioritizes the depth and direct relevance of participant experience over demographic representativeness [19]. Inclusion required that candidates (1) hold an active software developer role in the United States; (2) possess a minimum of five years of professional development experience; and (3) have worked in at least two of the following areas: mobile application development, web application development, and server-side or backend systems. These criteria ensured sufficient practitioner experience to generate meaningful accounts of organizational life with software errors.

A total of 40 participants were approached via LinkedIn and 17 participants matched the inclusion criteria. A final 15 participants were recruited and assigned coded identifiers (CD01-CD15) to protect confidentiality; the acceptance rate was 37.5%. The sample size reflects established norms in phenomenological research, where smaller purposive samples yield richer, more analytically productive data than larger heterogeneous ones [19]. Data collection continued until thematic saturation was reached, defined as the point at which additional interviews produced no new conceptual categories. Saturation was iteratively assessed rather than assumed: transcripts were coded and reviewed in successive batches of approximately five to six interviews, and the emergence of new codes was tracked across batches, consistent with established procedures for evaluating thematic saturation in qualitative research [18]. Saturation was reached when CD14 and CD15 of interviews produced no additional conceptual categories beyond the existing coding structure, after which no further participants were recruited. Demographic characteristics of the sample are summarized in Table 7.

How to cite this article:

Azmeera R, Hyatt J C (October 15, 2026) External Factors Influencing Software Developer Behavior and Their Contribution to Software Errors: A Phenomenological Study. Cureus J Comput Sci 3 : es44389-026-00326-1. DOI <https://doi.org/10.7759/s44389-026-00326-1>

Code	Education	Experience (years)	Age (years)	Gender	Sector	Int. Duration (minutes)	Primary Domain
CD01	Master's	13	38	Male	Retail	37	Mobile/Web/Backend
CD02	Bachelor's	17	41	Male	Convenience	27	Web/Backend
CD03	Master's	8	31	Male	Convenience	26	Mobile/Web
CD04	Bachelor's	13	38	Male	Technology	40	Backend/Architecture
CD05	Bachelor's	15	45	Female	Convenience	24	Web/Mobile
CD06	Master's	12	36	Male	Technology	24	Mobile iOS
CD07	Master's	24	50	Male	Health Care	33	Architect Mobile/ Web
CD08	Master's	7	30	Male	Technology	28	Backend
CD09	Master's	11	35	Female	Technology	34	React-Native Mobile
CD10	Bachelor's	20	44	Male	Technology	29	Mobile Android / iOS
CD11	Bachelor's	19	43	Female	Technology	36	Web /Mobile
CD12	Master's	15	40	Male	Asset Mgmt.	33	Architect Web/Mobile
CD13	Master's	12	38	Female	Retail	35	Web / Backend
CD14	Master's	8	33	Male	Technology	40	Mobile Android
CD15	Bachelor's	10	36	Male	Banking	38	Web /Mobile
Average	N/A	14.5	38.5	11 male / 4 female	Diverse	30.9	Cross-domain

TABLE 1: Participant demographic characteristics

Note. Participants are identified by coded designations to preserve confidentiality. Int. = Interview.

How to cite this article:

Azmeera R, Hyatt J C (October 15, 2026) External Factors Influencing Software Developer Behavior and Their Contribution to Software Errors: A Phenomenological Study. Cureus J Comput Sci 3 : es44389-026-00326-1. DOI <https://doi.org/10.7759/s44389-026-00326-1>

Data Collection

The primary data collection instrument was a semi-structured interview protocol developed from the relevant literature and validated through two pilot interviews prior to full data collection. Interviews were scheduled for sessions of 30-60 minutes; completed interviews ranged from 24 to 40 minutes ($M = 30.9$ minutes), as summarized in Table 7. The interviews were conducted via video conferencing to accommodate participants distributed across the United States and audio-recorded with explicit participant consent. Follow-up probes were used throughout to draw out elaboration and concrete examples, consistent with phenomenological interviewing practice.

Data Analysis

All interview recordings were transcribed verbatim and analyzed using the six-stage thematic analysis procedure described by Braun and Clarke [20]. The process moved from deep familiarization with the data, through systematic code generation and iterative grouping of codes, to theme consolidation, definitional refinement, and production of the final thematic account. Coding proceeded in two passes: an initial open, inductive pass generated codes directly from participants' own language, followed by a deductive pass in which recurring patterns were consolidated against the emerging code structure. Open coding produced 21 discrete codes across the transcript corpus. These were compiled into a working codebook, iteratively refined through constant comparison, and grouped into candidate code clusters based on the frequency of occurrence and distribution across participants; the clusters were then consolidated into the seven final themes reported here.

The two researchers independently coded a random 20% subsample of the transcripts using the same codebook. Coding was compared line-by-line, and percentage agreement across the subsample was 96%. Discrepancies were resolved through discussion until full consensus was reached; the reconciled codebook was then applied by the primary coder to the remaining transcripts. Member checking was conducted with a subset of participants to confirm the accuracy of transcripts and the representativeness of the identified themes.

Ethical considerations

The study protocol received Institutional Review Board (IRB) approval (approval no. #0324-96827, University of the Cumberland's IRB) before data collection began. Every participant completed a written informed consent process covering voluntary participation, the right to withdraw at any time without consequence, and the data anonymization procedures applied throughout. No personally identifying information was retained in any analytic record.

AI and software tool disclosure

Grammarly was used solely for grammar checking and minor language corrections during manuscript preparation. No AI or LLM tools were used for drafting content, conducting analysis, or developing the research methods. The authors reviewed all edits and take full responsibility for the accuracy and integrity of the manuscript.

Discussion

Findings

Analysis of the 15 interview transcripts identified seven thematic categories representing the external factors that developers linked to their experience of introducing software errors. Table 2 summarizes each theme, the number of coded references, and the number of participants whose accounts contributed to it.

How to cite this article:

Azmeera R, Hyatt J C (October 15, 2026) External Factors Influencing Software Developer Behavior and Their Contribution to Software Errors: A Phenomenological Study. *Cureus J Comput Sci* 3 : es44389-026-00326-1. DOI <https://doi.org/10.7759/s44389-026-00326-1>

#	Theme	Refs (n)	Participants (n)	TOE Dimension
T1	Time Constraints & Dynamic Requirements	42	11	Organizational
T2	Communication Barriers & Team Dependencies	18	7	Organizational
T3	Knowledge Gaps & Testing Strategies	15	8	Organizational / Environmental
T4	Resource Constraints & Industry Awareness	10	8	Organizational
T5	Development Modalities & Architecture	11	6	Technological
T6	Legacy Systems, Infrastructure & Security	5	4	Technological
T7	Financial Constraints & Leadership Vision	11	5	Organizational

TABLE 2: Summary of identified external factor themes

T1: Time Constraints & Dynamic Requirements

The most frequently cited external factor, reported by 11 of the 15 participants, was the combination of compressed delivery timelines and unpredictably changing requirements. Business-driven timelines routinely outpaced technical feasibility, and quality-compromising trade-offs became structural necessities rather than individual choices. Participants described how unrealistic schedules forced them to bypass quality controls, shorten testing cycles, and choose implementation approaches optimized for speed rather than correctness. CD01 captured the core trade-off:

“If you plan it correctly, you can scale better by balancing time, speed, and quality. It is often said that time and quality cannot go together, and tight deadlines definitely affect quality. The tight deadline creates time pressure, which often leads developers to put wrong code and introduce issues in the software.”

Time pressure also eliminated opportunities for structured code review, the practice most reliably associated with catching defects before they reach production. CD02 described the experience of too many tasks and too little time: “When you have limited time and need to accomplish a lot of things, that can lead to errors.” Frequently changing requirements made the problem worse by forcing developers to revisit and rewrite code mid-cycle. CD03 described the consequence plainly: compressing the period of a 3-month project into 1 or 2 months does not reduce the work it transfers into defects.

T2: Communication Barriers and Team Dependencies

How to cite this article:

Azmeera R, Hyatt J C (October 15, 2026) External Factors Influencing Software Developer Behavior and Their Contribution to Software Errors: A Phenomenological Study. Cureus J Comput Sci 3 : es44389-026-00326-1. DOI <https://doi.org/10.7759/s44389-026-00326-1>

Communication failures and cross-team dependencies were identified by seven participants, generating 18 coded references. Software development depends on the accurate transfer of business intent into technical requirements; participants perceived breakdowns in communication between stakeholders and developers as a direct contributor to the defects they described. Participants described a range of failure modes: vague verbal feature requests, incomplete user stories, and requirements spread across informal communication channels that left critical details unresolved. Participants described that these gaps are forcing them into assumption-based implementation. CD01 described the dynamics directly: "There is a communication barrier, so understanding what the ask is versus what you deliver."

CD01 also described a practice that addressed this directly: a company culture requiring all requirements to be written in complete, fully specified statements before development could begin. External team dependencies added further risk: when developers needed data access, API specifications, security credentials, or infrastructure setup from other teams, delays and gaps in those responses directly disrupted development timelines and product quality.

CD05 expressed that "while developing shared modules I need to depend on other team members and as they have their own priorities, my request gets delayed and lead to write assumed code." This shows that developers are writing code with limited information and this practice is producing errors by missing out edge case handling in their code.

T3: Knowledge Gaps and Testing Strategies

Eight participants contributed 15 coded references describing how gaps in technical or domain knowledge, combined with inadequate testing practices, led to software errors. Two types of knowledge gaps were identified. The first involved insufficient understanding of a system's external dependencies, integration requirements, and architectural constraints. The second involved gaps in the knowledge of specific frameworks, libraries, or language features. CD05 described how incomplete scope assessment consistently caused mid-project problems: "Whether it might be an announcement on the current feature or might be a new feature, these are the two main things which I found is having a major gap on the scope assessment."

CD09 supported this finding, noting that initial estimates were routinely optimistic. Tasks projected to take 5 days frequently expanded due to unforeseen obstacles, deprecated library dependencies, network infrastructure issues, and undocumented system behaviors. Inadequate testing compounded the effect by failing to catch defects early in the software development lifecycle. Insufficient unit test coverage and failure to test boundary conditions allowed defects to pass through into later stages where the cost of fixing them is substantially higher [3].

T4: Resource Constraints and Industry Awareness

Eight participants across 10 references identified two distinct resource-related pressures. The first was human resource availability: team attrition, unplanned absences, and chronic understaffing placed disproportionate workloads on the developers who remained. Participants linked this pattern to elevated cognitive load and reduced time for quality-oriented practices (see Discussion). CD08 described the effect as follows: "Let's say five people are working on a project, and suddenly, one person drops out. In this case, the remaining developers have to pick up that person's work. All these factors can indirectly affect the developers to rush through tasks, which may lead to overlooking edge cases."

The second pressure was the gradual outdateding of developers' awareness of current tools, libraries, and technology practices. Developers who did not actively keep up with changes in the technology landscape were more likely to implement solutions using deprecated approaches or to miss better alternatives. CD01 identified staying updated with the industry as a professional quality responsibility, not an optional activity.

T5: Development Modalities and Architecture

Six participants produced 11 references describing problems caused by inappropriate methodology selection and poor architectural planning. Participants repeatedly observed a gap between how agile was described in their organizations and how it was actually practiced. CD04 captured the gap: "I have never seen a company that implements Agile 100%. When we consider doing development and QA in parallel, in reality, that has never taken a good shape."

How to cite this article:

Poor design decisions during the early phases of a project often made without consulting experienced architects or domain experts introduced structural weaknesses that created ongoing problems. CD03 described the consequence: "Without a clear design system in place, attempting to develop a system based on limited knowledge often leads to missing pieces and introduces errors in the system." Technology selection decisions, once made, were very difficult to change, and learning curves for unfamiliar technologies directly reduced code quality during the transition period.

T6: Legacy Systems, Infrastructure, and Security

Four participants described five instances in which legacy system constraints, infrastructure limitations, or external security threats imposed additional burdens. Legacy systems built on outdated technology stacks required developers to build fragile workarounds. CD03 described the situation: "We end up using the legacy systems, and with some of these systems, we cannot easily plug and play new technologies. In such cases, we must depend on patching solutions. Those systems are not up to date with current technologies, and we cannot even deprecate them."

Infrastructure limitations including limited computing capacity, lack of system redundancy, and budget-constrained tooling restricted the environment in which quality software could be built and tested. Security attacks on production systems, including penetration attempts and exploitation of known vulnerabilities, also created urgent, unplanned remediation work that diverted attention from planned development and compressed already tight schedules.

T7: Financial Constraints and Leadership Vision

Five participants, contributing 11 references, described how budget limitations and leadership quality shaped development conditions. Financial constraints appeared in two main forms: restricted hiring budgets that prevented teams from recruiting highly skilled developers and tool acquisition constraints that prevented adoption of enterprise-grade platforms. CD01 noted: "The reason why companies like Google and Facebook are among the top on the Forbes list is that they hire only high-quality developers. They are ready to pay top dollar for the right candidate."

Leaders who excluded developers from planning discussions, accepted externally imposed timelines without technical input, or failed to advocate for adequate resources created conditions in which developers operated under compounding pressures. CD04 described the distinction: "The leader should have the mindset of working with the team rather than making the team work." Participatory leadership characterized by open communication, realistic expectation-setting, and active support for developer needs was consistently described as a factor that partially offset the other external pressures documented throughout the findings.

Discussion

The seven themes collectively demonstrate that software defect introduction is substantially shaped by organizational and environmental conditions rather than by individual developer capability alone. This conclusion extends findings in the human factors and software quality literature [11], [21] and reinforces the thesis advanced by Azmeera [7] that software error escalation is a systemic, organizationally embedded problem that requires systemic responses.

Mapped onto the TOE framework [8], the themes fall coherently across three contextual dimensions. T5 and T6 are primarily technological in character, reflecting how architecture choices, development tooling, infrastructure, and legacy system constraints define the technical conditions of development. T1, T2, T4, and T7 are predominantly organizational, reflecting how management processes, communication culture, resource allocation, team staffing, and financial decisions shape the human conditions of development. T3 spans organizational and environmental dimensions, since knowledge currency gaps may reflect both insufficient organizational investment in continuous learning and the rapid pace of change in the external technology landscape.

The prominence of T1 time constraints and dynamic requirements reported with the highest reference count and broadest participant coverage of the seven themes aligns with experimental literature showing that time pressure is among the strongest predictors of developer performance degradation [16]. The phenomenological contribution of this

How to cite this article:

Azmeera R, Hyatt J C (October 15, 2026) External Factors Influencing Software Developer Behavior and Their Contribution to Software Errors: A Phenomenological Study. *Cureus J Comput Sci* 3 : es44389-026-00326-1. DOI <https://doi.org/10.7759/s44389-026-00326-1>

study is interpretive: participants' accounts suggest that tight deadlines do not simply reduce the time available for quality work but are perceived by developers as reshaping how they make decisions, orienting them toward faster and less rigorous approaches.

The finding that participants consistently linked communication failures (T2) to software errors they described reframes requirements engineering as a quality issue, not merely a project management concern. Unclear or incomplete requirements represent a form of technical debt incurred before a single line of code is written: debt that developers must repay through guess-based implementation, which participants associated with a higher likelihood of defects [17].

Themes T5 and T6 highlight structural and architectural factors that individual developers cannot resolve on their own. The persistence of legacy system dependencies and the widespread failure to implement agile methodologies as designed are organizational-level problems. Quality improvement programs that target only developer behavior through training, certification, or performance management treat the symptom while leaving root causes untouched.

Recommendations

For Software Developers

Developers should proactively seek written clarification of requirements before beginning implementation, explicitly documenting assumptions and requesting resolution from the appropriate stakeholders. Thorough scope assessment including identification of all external dependencies at the planning stage reduces the unexpected problems caused by knowledge gaps documented in T3. Developers should also commit to staying current with the technology landscape, treating awareness of current tools, frameworks, and security vulnerabilities as a professional responsibility that directly affects the quality of their work.

For Project Managers

Project managers should establish requirements documentation standards that define clear completeness criteria before development begins. Sprint planning should explicitly account for dependency resolution activities and build in realistic time buffers. Capacity planning should be based on actual team availability, adjusted for expected attrition, rather than on theoretical full-team size. When external pressure requires scope or timeline compression, managers should formally document the associated quality risks and escalate those trade-off decisions to appropriate stakeholders rather than absorbing them silently into developer workload.

For Technology Leaders and Organizations

Technology leaders should require architectural review by experienced practitioners before implementation begins on complex systems or features. Investment in legacy system modernization should be treated as quality debt reduction with a measurable return on investment through lower defect rates. Organizations should adopt a systemic view of software quality in which defect rates are treated as organizational health indicators. Tracking defect origins against the external factor categories identified in this study - schedule compression, communication gaps, resource shortfalls, architectural weaknesses, and leadership failures - enables targeted, evidence-based interventions. Sustained investment in talent development, knowledge-sharing infrastructure, and inclusive team culture should be understood as quality investments with measurable returns in reduced defect rates.

Limitations and future research

Several limitations affect the scope and generalizability of these findings. The sample was restricted to the United States; software development cultures and organizational norms differ significantly across countries, and findings may not apply directly in other national or industry contexts. Purposive sampling, while appropriate for phenomenological research, does not support statistical generalization to the broader population of software developers. Participant recruitment through LinkedIn may have introduced bias toward developers with more established professional profiles and more reflective attitudes toward their work experiences.

How to cite this article:

Azmeera R, Hyatt J C (October 15, 2026) External Factors Influencing Software Developer Behavior and Their Contribution to Software Errors: A Phenomenological Study. *Cureus J Comput Sci* 3 : es44389-026-00326-1. DOI <https://doi.org/10.7759/s44389-026-00326-1>

The retrospective interview format depends on participants recalling past experience rather than observing behavior in real time. Recall bias may cause particularly memorable incidents to be over-represented while routine, everyday error-contributing conditions receive less attention. Future research using longitudinal designs or experience sampling methods would complement and strengthen these retrospective accounts.

The most productive directions for future research include: (1) developing validated survey instruments based on the seven themes identified here, for testing at large scale across diverse populations; (2) conducting multi-country comparative studies to examine whether these findings vary across different national and organizational contexts; (3) designing controlled intervention studies to measure whether organizational practices that target the identified themes produce measurable reductions in defect rates; (4) analyzing how combinations of external factors interact and compound; and (5) conducting further phenomenological investigation into variation within themes.

Conclusions

This study used a phenomenological qualitative methodology to investigate how external organizational and environmental factors shape software developer behavior in ways that contribute to software errors. Fifteen experienced professional developers provided detailed, experience-centered accounts that were analyzed into seven thematic categories of external influence. These themes reveal a consistent finding: software error introduction is substantially determined by conditions that developers neither choose nor control - organizational schedules, communication cultures, resource levels, architectural foundations, legacy technical debt, and leadership practices rather than by the competence or effort of individual developers. Across the results, time constraints and changing requirements (T1) emerged as the most consistently reported external factor, cited nearly twice as often as any other theme (see Table 2 and Findings section for full frequencies). Knowledge gaps and testing strategies (T3), resource constraints and industry awareness (T4), and communication barriers (T2) were also broadly represented across participants. The remaining themes T5, T6, and T7, though reported by fewer participants, still reflected substantive challenges and should not be read as secondary in practical importance. Taken together, the seven themes point less to a single dominant cause of software errors and more to a layered set of organizational and environmental conditions, with schedule pressure as the most pervasive, though not the only, significant force shaping error-prone development.

The study contributes to an emerging research agenda, initiated in part by the doctoral work of Azmeera, that treats software quality as an organizational systems property rather than an individual performance attribute. Achieving lasting improvements in software reliability requires that organizations address the systemic conditions identified here: realistic scheduling, disciplined requirements communication, adequate resourcing, sound architectural governance, and developer-inclusive leadership. This evidence base supports this reorientation and provides a thematically structured foundation for both future research and practical organizational change.

Additional Information

Author Contributions

All authors have reviewed the final version to be published and agreed to be accountable for all aspects of the work.

Concept and design: Rahul Azmeera, James C. Hyatt

Acquisition, analysis, or interpretation of data: Rahul Azmeera, James C. Hyatt

Drafting of the manuscript: Rahul Azmeera, James C. Hyatt

Critical review of the manuscript for important intellectual content: Rahul Azmeera, James C. Hyatt

Supervision: Rahul Azmeera, James C. Hyatt

How to cite this article:

Azmeera R, Hyatt J C (October 15, 2026) External Factors Influencing Software Developer Behavior and Their Contribution to Software Errors: A Phenomenological Study. *Cureus J Comput Sci* 3 : es44389-026-00326-1. DOI <https://doi.org/10.7759/s44389-026-00326-1>

Disclosures

Human subjects: Consent was obtained from all participants in this study. University of The Cumberlands issued approval #0324-96827. Principal Investigator: Rahul Azmeera From: Institutional Review Board Subject: IRB Approved Project title: The Effects of Increased Software Errors on Software Product Functionality IRB Approval Number: #0324-96827 Approval Date: 2024-03-07 Thank you for submitting your materials to the IRB office. The above referenced human-subjects research project has been approved by the University of the Cumberlands Institutional Review Board. This approval is limited to the approved protocols described in the application which have been reviewed as acceptable activities described by the Office of Human Research Protections (HHS.org). It has been determined that your study meets federal criteria to qualify as an expedited study in accordance with the requirements set forth in 45 CFR 46.110 finding that 1) the research is minimal risk, 2) that if identification of the participants and/or their responses reasonably place them at risk of criminal or civil liability or could be damaging to the participants' financial standing, employability, insurability, or reputation, or be stigmatizing there are reasonable and appropriate protections that will be implemented so that risk related to invasion of privacy and breach of confidentiality are no greater than minimal, and 3) that the research is not classified or does not involve prisoners, with the exception that the expedited review of minor amendments for approved studies involving prisoners may be used. However, if there are changes to research project in the following areas a modification form must be submitted to the IRB office: Substantial change to recruitment materials or consent documents Change in the data collection process Change in the location of the study Change in key personnel Change in instrumentation Principal investigators are responsible for ensuring that studies are conducted according to University protocol. As a principal investigator, you have multiple responsibilities to the IRB, the research subjects and the faculty partner. If you have questions, please feel free to email me at IRB@ucumberlands.edu Please continue to work with your dissertation advisor as you proceed. IRB Office University of the Cumberlands 6429 College Station Drive | Williamsburg, KY 40769.

Conflicts of interest: In compliance with [the ICMJE uniform disclosure form](#), all authors declare the following:

- **Payment/services info:** All authors have declared that no financial support was received from any organization for the submitted work.
- **Financial relationships:** All authors have declared that they have no financial relationships at present or within the previous three years with any organizations that might have an interest in the submitted work.
- **Other relationships:** All authors have declared that there are no other relationships or activities that could appear to have influenced the submitted work.

Data Availability Statements

The datasets (and/or code) supporting this study are available from the corresponding author upon reasonable request.

References

1. Leveson NG: [Engineering a Safer World: Systems Thinking Applied to Safety](#). The MIT Press, Cambridge, MA; 2012. [10.7551/mitpress/8179.001.0001](#)
2. Glass RL: [Frequently forgotten fundamental facts about software engineering](#). IEEE Software. 2001, 18:112-111. [10.1109/ms.2001.922739](#)
3. Jones C, Bonsignour O: [The Economics of Software Quality](#). Addison-Wesley Professional, Upper Saddle River, NJ; 2011.
4. LaToza TD, Venolia G, DeLine R: [Maintaining mental models: a study of developer work habits](#). ICSE '06: Proceedings of the 28th international conference on Software engineering. 2006, 492-501. [10.1145/1134285.1134355](#)
5. Iqbal ST, Horvitz E: [Disruption and recovery of computing tasks: field study, analysis, and directions](#). CHI '07: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. 2007, 677-686. [10.1145/1240624.1240730](#)

How to cite this article:

Azmeera R, Hyatt J C (October 15, 2026) External Factors Influencing Software Developer Behavior and Their Contribution to Software Errors: A Phenomenological Study. Cureus J Comput Sci 3 : es44389-026-00326-1. DOI <https://doi.org/10.7759/s44389-026-00326-1>

6. Creswell JW, Poth CN: [Qualitative Inquiry and Research Design: Choosing Among Five Approaches, 4th ed.](#) Sage Publications, Thousand Oaks, CA; 2016.
7. Azmeera R: [The effects of increased software errors on software product functionality \(Publication No.: 32173249\) \[Ph.D. dissertation, University of the Cumberlands, Williamsburg, KY\]](#). ProQuest Dissertations and Theses Global. 2025, 10.13140/RG.2.2.18820.80009
8. IEEE Computer Society: [IEEE standard glossary of software engineering terminology](#). IEEE Std 610.12-1990. 1990, 1-84. 10.1109/IEEESTD.1990.101064
9. Beizer B: [Software Testing Techniques, 2nd ed.](#) Van Nostrand Reinhold, New York, NY; 1990.
10. Boehm B, Turner R: [Balancing Agility and Discipline: A Guide for the Perplexed](#). Addison-Wesley Longman Publishing Co., Inc., Boston, MA; 2003.
11. Dybå T, Dingsøyr T: [Empirical studies of agile software development: A systematic review](#). Information and Software Technology. 2008, 50:833-859. 10.1016/j.infsof.2008.01.006
12. Capretz LF, Ahmed F: [Making sense of software development and personality types](#). IT Professional. 2010, 12:6-13. 10.1109/mitp.2010.33
13. Zhu K, Kraemer K, Xu S: [Electronic business adoption by European firms: a cross-country assessment of the facilitators and inhibitors](#). European Journal of Information Systems. 2003, 12:251-268. 10.1057/palgrave.ejis.3000475
14. van Manen M: [Researching Lived Experience: Human Science for an Action Sensitive Pedagogy, 2nd ed.](#) Routledge, New York, NY; 2016.
15. Lenberg P, Feldt R, Wallgren LG: [Behavioral software engineering: a definition and systematic literature review](#). Journal of Systems and Software. 2015, 107:15-37. 10.1016/j.jss.2015.04.084
16. Bergersen GR, Hannay JE, Sjøberg DIK, Dybå T, Karahasanovic A: [Inferring skill from tests of programming performance: combining time and quality](#). 2011 International Symposium on Empirical Software Engineering and Measurement. Banff, AB, Canada; 2011. 305-314. 10.1109/ESEM.2011.39
17. Bjørnson FO, Dingsøyr T: [Knowledge management in software engineering: a systematic review of studied concepts, findings and research methods used](#). Information and Software Technology. 2008, 50:1055-1068. 10.1016/j.infsof.2008.03.006
18. Miles MB, Huberman AM, Saldaña J: [Qualitative Data Analysis: A Methods Sourcebook, 3rd ed.](#) Sage Publications, Thousand Oaks, CA; 2014.
19. Patton MQ: [Qualitative Research and Evaluation Methods](#). Sage Publications, Thousand Oaks, CA; 2014.
20. Braun V, Clarke V: [Using thematic analysis in psychology](#). Qualitative Research in Psychology. 2006, 3:77-101. 10.1191/1478088706qp0630a
21. Tornatzky LG, Fleischer M, Chakrabarti AK: [The Process of Technological Innovation](#). Lexington Books, Lexington, MA; 1990.

How to cite this article:

Azmeera R, Hyatt J C (October 15, 2026) External Factors Influencing Software Developer Behavior and Their Contribution to Software Errors: A Phenomenological Study. Cureus J Comput Sci 3 : es44389-026-00326-1. DOI <https://doi.org/10.7759/s44389-026-00326-1>