Journal of Computer Science and Technology Studies
ISSN: 2709-104X
DOI: 10.32996/jcsts
Journal Homepage: al-kindipublisher.com/index.php/jcsts/index

JCSTS
AL-KINDI CENTER FOR RESEARCH AND DEVELOPMENT

| RESEARCH ARTICLE

Software Errors and Their Effects on Software Product Functionality: A Literature Review and Research Gap Analysis Through the Technology-Organization-Environment Lens

Rahul Azmeera[1]✉, James C Hyatt[2], Sravanthi Dontu[3], Qamarsultana Alad[4]

[1]Indipendent researcher, Department of Information Technology, University of the Cumberlands, Williamsburg, KY, USA
razmeera8848@ucumberlands.edu

ORCID: 0000-0003-4454-1426

[2]Adjunct Professor, Department of Information Technology, University of the Cumberlands, Williamsburg, KY, USA
james.hyatt@ucumberlands.edu

[3]Indipendent researcher, Department of Information Technology, University of the Cumberlands, Williamsburg, KY, USA
sdontu21733@ucumberlands.edu

[4]Indipendent researcher, Department of Information Technology, University of the Cumberlands, Williamsburg, KY, USA
qalad26887@ucumberlands.edu

Corresponding Author: Rahul Azmeera, **E-mail:** razmeera8848@ucumberlands.edu

| ABSTRACT

The pervasive integration of software into digital products has made software errors a first-order concern for users, developers, and organizations, because defects that survive development and testing directly degrade software product functionality. This paper presents a structured literature review of software errors and their effects on software product functionality, organized through the Technology-Organization-Environment (TOE) framework. Drawing on studies published primarily between 2019 and 2023, and retrieved from Google Scholar, IEEE Xplore, the ACM Digital Library, and ProQuest, the review synthesizes evidence across three directions: technological factors that introduce errors, including development tools, documentation quality, architecture, third-party dependencies, and cloud platforms; organizational factors and consequences, including security practice, regulatory compliance, requirements understanding, and resourcing; and external environmental factors that affect software developers, including market pressure, distributed teamwork, human values, and privacy regulation. The review then evaluates conventional, automated, and emerging artificial intelligence based testing approaches and their limitations in preventing errors from reaching production. The synthesis shows that prior work has extensively documented what software errors cost and how testing detects them, but has rarely examined the root causes of errors from the lived perspective of the software developers who introduce, detect, and repair them across all three TOE dimensions simultaneously. Five research gaps are identified and mapped to future research directions, including qualitative phenomenological inquiry into developer experiences and human-like, vision-based automated testing.

| KEYWORDS

Software errors, software defects, software product functionality, software quality, software testing, TOE framework, literature review, research gaps

| ARTICLE INFORMATION

ACCEPTED: 15 August 2026 **PUBLISHED:** 23 September 2026 **DOI:** 10.32996/jcsts.2026.8.9.10

1. Introduction

Software is a set of instructions and programs that controls the behavior of a computer system and enables it to perform specific tasks (Englander & Wong, 2021). When technology designers create software, they expect it to produce the desired output; however, coding errors and weaknesses introduced during development cause the software to deviate from expected results, leading to failures that range from minor inconveniences to critical system malfunctions (Bojanova & Galhardo, 2023; Kumari et al., 2019). The problem motivating this review is that an increase in software errors has led to functional issues in software products (Alami & Krancher, 2022; Khaliq, 2022; Thirumoorthy & Britto, 2022), with consequences that extend from individual users to entire organizations and, in safety-critical domains, to human lives (Johnston & Harris, 2019; Travis, 2019).

The economic stakes are substantial. The Consortium for Information and Software Quality estimated the cost of poor software quality in the United States at 2.08 trillion US dollars for 2020, with software technical debt continuing to grow (Krasner, 2021). The market for automated software quality analysis was projected to grow from 8.52 billion US dollars in 2018 to 19.27 billion US dollars by 2023, a compound annual growth rate of 17.7 percent (Tao et al., 2019). These figures indicate both the scale of the software error problem and the intensity of industrial investment in detecting errors before release.

Despite decades of progress in software testing methods, frameworks, and tools (Ahmad et al., 2019; Vadan & Miclea, 2023), errors continue to escape into production, where their severity ranges from low-impact display anomalies to critical failures that crash applications, corrupt data, and break core functionality (Liu et al., 2023; Ni et al., 2022). Understanding why this happens requires looking beyond testing technology alone. Errors originate in a sociotechnical system in which developer skill, tool quality, organizational process, and external pressures interact (Lopez et al., 2021; Baham & Hirschheim, 2022). A framework capable of organizing evidence across these interacting layers is therefore needed.

This paper reviews the literature on software errors and their effects on software product functionality through the lens of the Technology-Organization-Environment (TOE) framework (Baker, 2011; Horani et al., 2023). The contributions are fourfold. First, the review consolidates dispersed evidence on how technological, organizational, and environmental factors contribute to the introduction of software errors. Second, it synthesizes findings on the consequences of software errors for products, organizations, users, and developers. Third, it evaluates the state of conventional, automated, and artificial intelligence (AI) based testing approaches and their documented limitations. Fourth, it derives explicit research gaps and maps them to future research directions, including the need for qualitative inquiry into developers' lived experiences of software errors.

The remainder of the paper is organized as follows. Section 2 describes the review methodology. Section 3 introduces the TOE framework as the organizing lens. Section 4 summarizes the background on software pervasiveness and the software development life cycle (SDLC). Section 5 characterizes software errors, their life cycle, and severity. Sections 6, 7, and 8 review technological, organizational, and environmental factors, respectively. Section 9 reviews testing approaches and their limitations. Section 10 synthesizes the effects of errors on products and stakeholders. Section 11 presents the identified research gaps and future directions, and Section 12 concludes.

2. Review Methodology

This review followed a structured narrative approach. Literature was retrieved from Google Scholar, IEEE Xplore, the ACM Digital Library, and ProQuest using the keywords software errors, software developer challenges, and software security errors and their effects, together with combinations of individual terms. Initial keyword searches in Google Scholar returned approximately 5800, 16000, and 15800 results, respectively, confirming an active research trend on the challenges and effects of software development and errors during the last five years and the need to narrow the corpus systematically.

Only peer-reviewed journal articles, conference proceedings, scholarly books, and doctoral dissertations were considered; general websites and blogs were excluded. Most included studies were published between 2019 and 2023; older sources were retained only when a clear explanation of a foundational concept was not available in recent research, for example foundational treatments of software verification and validation (Boisvert et al., 2005) and aviation software challenges (Knight, 2002). Table 1 summarizes the inclusion and exclusion criteria, and Figure 1 shows the distribution of the cited studies by publication year, which concentrates in the 2019 to 2023 window consistent with the search strategy. Figure 2 in Section 3 shows how the retained literature is organized in this review.

Table 1. *Inclusion and Exclusion Criteria of the Review*

Criterion	**Specification**
Sources	Google Scholar, IEEE Xplore, ACM Digital Library, ProQuest
Publication types	Peer-reviewed journals, conference proceedings, scholarly books, doctoral dissertations
Time window	Primarily 2019 to 2023; earlier seminal works retained when no recent equivalent exists
Included topics	Software errors and defects, developer challenges, error effects on functionality, security consequences, software testing, AI-based testing
Excluded	Websites, blogs, non-scholarly gray literature, studies unrelated to software errors or quality

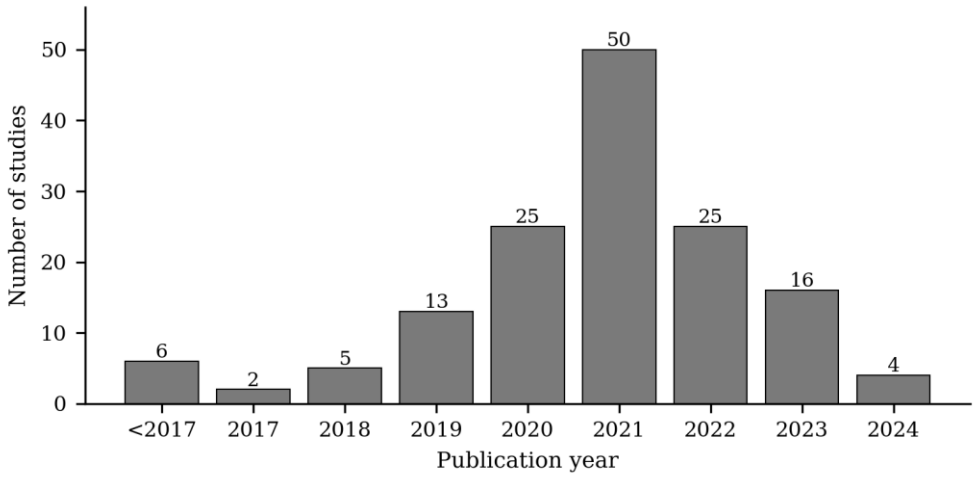

Figure 1. *Distribution of the studies cited in this review by publication year. The corpus concentrates in the 2019 to 2023 window, with earlier works retained for foundational concepts.*

Two limitations of the methodology should be acknowledged. First, as a structured narrative review rather than a fully protocol-driven systematic review, the study selection involved researcher judgment, which was mitigated by using explicit criteria and multiple databases. Second, the emphasis on recent literature means that long-run historical trends in software failure receive less coverage than contemporary evidence.

3. Theoretical Lens: The TOE Framework

The TOE framework explains technology-related phenomena in organizations through three interacting contexts: the technological context, which covers the characteristics of the technology itself, including its functionality, complexity, compatibility with existing systems, and ease of use; the organizational context, which covers the internal conditions, structures, and resources of the organization; and the environmental context, which covers external conditions such as market dynamics and regulatory requirements (Baker, 2011; Horani et al., 2023). The framework has been applied to a broad range of adoption and innovation problems, including cloud technology adoption (Hadwer et al., 2021) and organizational AI adoption (Horani et al., 2023).

Software development is itself a process in which technology, organization, and environment interact continuously (Hassan & Khan, 2021). Technological context issues manifest in the codebase, architecture, tools, development environments, scalability, and documentation. Organizational context issues manifest in development processes, methodology, testing management, budgets, security posture, and version control practices. Environmental context issues manifest in user expectations, market pressure, policies, and regulatory compliance. Analyzing software errors along all three constructs offers a multidimensional understanding of hidden gaps that single-perspective analyses miss. Table 2 maps the TOE constructs to the error-related factors reviewed in Sections 6 through 8.

Table 2. *TOE Constructs Mapped to Software Error Factors Reviewed in This Paper*

Construct	**Definition (Baker, 2011; Horani et al., 2023)**	**Error-related factors reviewed**
Technological	Characteristics of the technology: functionality, complexity, compatibility, ease of use	Tools and IDEs, documentation, architecture, open-source dependencies and SDKs, cloud platforms, hardware compatibility (Sec. 6)
Organizational	Internal conditions, culture, operations, and resources of the organization	Security practice, compliance, requirements and business knowledge, resourcing and hiring, stakeholder management (Sec. 7)
Environmental	External context: market conditions and regulatory requirements	Market pressure, distributed teams, human values, privacy regulation, knowledge attrition, user diversity (Sec. 8)

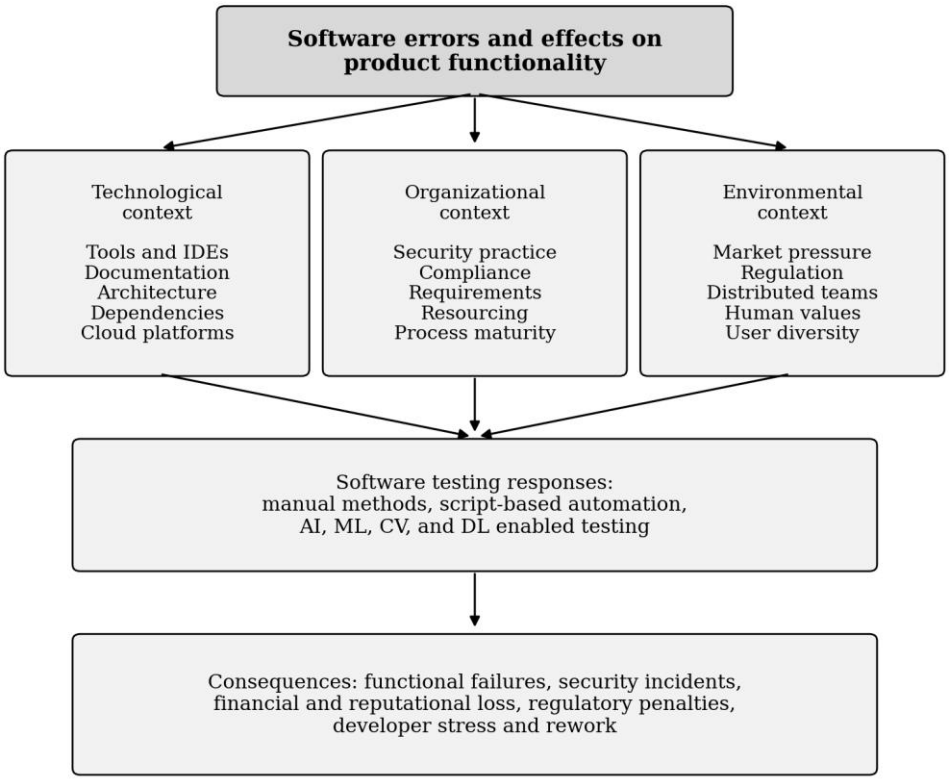

Figure 2. *Conceptual organization of the reviewed literature. Technological, organizational, and environmental factors jointly contribute to software errors; testing approaches attempt to intercept them; residual errors produce downstream consequences.*

4. Background: Software Pervasiveness and the SDLC

4.1 Growth of Software-Dependent Digital Products

Software has evolved from a supporting utility into the control center of most digital products, with platform-based software businesses reshaping entire industries (Cusumano et al., 2020). Digitalization has increased the market value of firms with high digital proximity, and information technology integration into products and back-office processes has improved outcomes and enabled further digital product development (Rahmati et al., 2021). Organizations use digital products and platforms to improve business models and market capabilities, although creating disruptive digital products demands difficult organizational and process changes (Simonsson et al., 2020; Bosch & Olsson, 2021).

The dependency is deep in specific domains. Smart home devices integrate software for functionality control (Chen et al., 2017; Seo et al., 2020). Healthcare uses software-driven Internet of Things (IoT) devices for glucose monitoring, remote surgery, and patient telemetry (Dantu et al., 2020), and smartphone-based systems such as the eCAALYX application track elderly patients with chronic diseases through wearable sensors (Boulos et al., 2011). Modern automobiles embed extensive software and sensors to collect driving data and control vehicle systems, a trend described as chipification (Forelle, 2022), and artificial intelligence increasingly supports path planning and hardware control in automotive systems (Gheorghe, 2022; Nakrani & Joshi, 2021). Aviation depends on onboard and ground digital systems for monitoring, communication, and autopiloting (Knight, 2002), unmanned aerial vehicles rely on onboard software for autonomous maneuvering (Koumaras et al., 2021), and ethical development of such systems is essential because glitches can create disastrous situations (Biable et al., 2022).

Two properties of this landscape amplify the consequences of software errors. First, digital products are tightly coupled with their software, meaning the product cannot behave independently of software behavior (Hein et al., 2018). Tight software-hardware coupling creates new attack surfaces, including hardware trojans that exploit software-controlled circuits (Šišejković et al., 2019) and control logic injection attacks on industrial controllers (Yoo & Ahmed, 2019), while software is also tightly coupled to business processes and data, so exploitation can halt operations and destroy organizational trust (Wieringa et al., 2021). Second, software increasingly takes control over human actions. Autopilot systems assume navigation responsibility, and increased human trust in such automation means that minor glitches can place people in dangerous situations with little time to react (Gillath et al., 2021;

Madison et al., 2021). In healthcare wearables, glitches in sensor data can trigger false alarms or missed alerts with potentially fatal consequences (Manninger et al., 2021).

4.2 The Software Development Life Cycle

The SDLC is a structured process for creating software systems with high efficiency, consisting of requirements gathering and analysis, design and architecture, development, testing, and deployment with feedback (Arrey, 2019). In the waterfall methodology the phases are sequential, whereas Agile permits requirement and design changes throughout the life cycle with continuous parallel improvement (Arrey, 2019; Waja et al., 2021). Each stage contributes to, or defends against, error introduction: unclear requirements propagate into wrong functionality, architectural decisions constrain quality attributes, coding introduces defects, testing intercepts a subset of them, and deployment feedback reveals what escaped (Waja et al., 2021). Agile adoption itself faces open theoretical and practical challenges that affect quality outcomes (Baham & Hirschheim, 2022), and empirical evidence shows that Scrum practices add value to software quality primarily through collaboration and process discipline rather than automatically (Alami & Krancher, 2022).

5. Software Errors: Nature, Life Cycle, and Severity

5.1 Definitions and Origins

Terminology in this space is often conflated. Recent work distinguishes bugs, faults, errors, and weaknesses in the context of software security vulnerabilities (Bojanova & Galhardo, 2023), and ontological analyses formalize the relations among defects, errors, and failures and their propagation to dependent systems (Duarte et al., 2018). In operational terms, when software fails to work as expected in a complex system environment, that failure is referred to as a bug, and in large-scale systems such as autonomous ships, errors left in the design and coding phases combine with user-driven errors produced by poor interface design (Chang et al., 2021). Software must traverse circuits, digital design, computer architecture, operating systems, and programming languages to execute, and it must avoid errors at every layer to produce correct results; resource and energy trade-offs during computation can themselves perturb outcomes (Stanley-Marbell et al., 2021). Accuracy and precision are the key elements determining software reliability, and verification and validation compare computed results against real-world references and mathematical models (Boisvert et al., 2005).

5.2 Simple Bugs with Large Consequences

Analysis of the simple stupid bugs (SStuB) dataset shows that trivial single-statement defects, which compile and run without complaint, silently alter functional behavior and have caused million-dollar losses; the majority were introduced by the original author during the initial coding phase rather than through later modification by others (Mosolygo et al., 2021). The time to fix these bugs had a median of 58 days and a mean of approximately 240 days with a standard deviation of 440 days, indicating that even simple defects hide effectively in large codebases (Mosolygo et al., 2021). Lack of rigorous testing and inadequate code review allow such errors to remain unnoticed for extended periods (Mosolygo et al., 2021).

5.3 The Bug Life Cycle and Its Costs

Once discovered, a bug enters a workflow of creation, assignment, resolution, verification, and closure that consumes developer time and affects project timelines; duplicate bug reports filed by different users force redundant investigation of issues already under active work, a direct consequence of weak duplicate management (Kucuk & Tuzun, 2021). Defect severity classification, from low to critical, determines triage priority, and recent work applies multi-feature fusion and transformer-based models to automatically detect high-severity defects and predict vulnerability severity (Liu et al., 2023; Ni et al., 2022). Surveys of failures in the software development process identify lack of testing, weak project management, poor requirements analysis, time constraints, and poor functionality analysis as recurring causes (Marques et al., 2017), and emerging application classes remain undertested; virtual reality applications, for example, exhibit higher code smell density than non-VR software (Andrade et al., 2019). Figure 3 summarizes the pathway from error introduction through detection escape to operational failure and stakeholder impact.

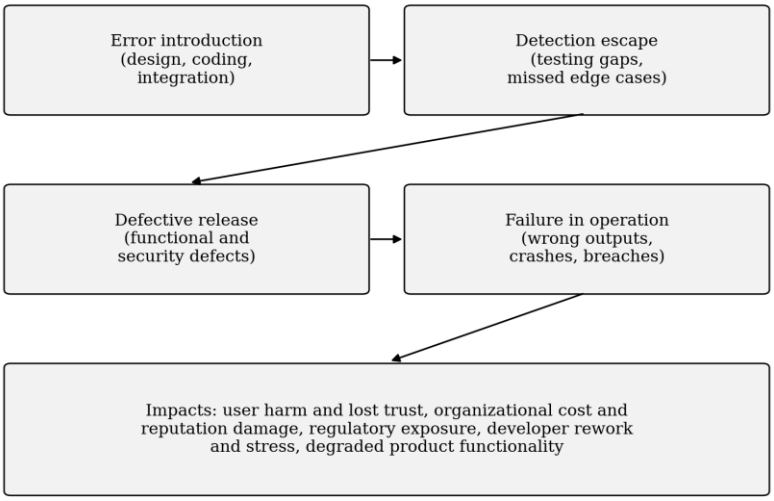

Figure 3. *Pathway from error introduction to stakeholder impact synthesized from the reviewed literature. Testing gaps allow defects to escape into releases, where operational failures generate functional, organizational, and human consequences.*

6. Technological Factors Contributing to Software Errors

This section reviews evidence on the technological context of the TOE framework: characteristics of tools, documentation, architecture, dependencies, and platforms that contribute to error introduction. Table 3 summarizes the factors and representative studies.

6.1 Development Tools and Environments

Developers depend heavily on tools, and tool quality shapes error handling. Empirical work shows that developers rely on tool-produced error messages as their first viewpoint on a problem, that learning to communicate with complex technologies and interpret their errors is a major difficulty, and that tool distractions directly cost time and budget; when tools themselves misbehave, developers resort to trial-and-error workarounds under time pressure while still being expected to write quality code (Lopez et al., 2021). Studies of DevOps adoption identify lack of technical infrastructure, limited tool stacks, and lack of tool expertise as major challenges (Jayakody & Wijayanayake, 2021), and systematic review evidence confirms that learning new technologies, tools, and methods requires significant effort, that teams often lack technical expertise, and that resistance to unfamiliar tools persists, motivating the development of more developer-friendly tooling (Khan et al., 2022). Container-based development introduces its own error surface: analysis of Stack Overflow questions on Docker found application development and framework integration issues to be the dominant concern at roughly 21 percent, followed by networking issues at approximately 13 percent, alongside memory management and configuration difficulties (Haque et al., 2020). Migrating and integrating legacy systems on which the business heavily depends poses additional persistent difficulty (Alexandrova & Rapanotti, 2019), and shared components maintained by very large distributed groups of developers slow debugging and small changes due to coordination overhead (Patel et al., 2020).

6.2 Documentation Quality

Documentation is a critical asset that reduces the time developers spend reading code and supports future integration work, yet practitioners report systematic deficiencies. A large survey found that lack of correctness and synchronization between documentation and code was reported by 59 percent of respondents, time constraints for producing documentation by 65 percent, and neglected contribution documentation acknowledged by 64 percent, alongside outdated content and missing information about tools and third-party configuration (Aghajani et al., 2020). Incorrect installation steps, wrong code comments, and erroneous example code directly mislead developers who rely on them, causing defects downstream (Laplante, 2024).

6.3 Architecture and Complexity

Software architecture is the blueprint on which developers and stakeholders rely, and changes that introduce new tools and technologies impose learning curves and error risk (Taylor, 2019). Exploratory surveys report that misalignment between architecture and source code, often rooted in communication failures between technical and non-technical teams, disrupts developers and project timelines (Tian et al., 2022). Architectural complexity creates difficulties in scaling, understanding, and maintenance, and balancing rigidity against flexibility remains an unresolved trade-off (Nugraha & Talita, 2021; Silva et al., 2023).

6.4 Open-Source Dependencies and SDKs

Open-source components accelerate development but import risk. Open-source contributors report challenges with quality and usability, outdated tooling, globally distributed communication, and resource constraints, and many contributors prioritize features over usability (Wang et al., 2022). Vulnerable dependencies frequently remain out of sight and unpatched in downstream projects, introducing information disclosure and denial-of-service exposure (Prana et al., 2021). Qualitative evidence on dependency management shows that continuous library updates force constant compatibility and security monitoring, and that weak testing in upstream projects limits library reliability (Pashchenko et al., 2020). Software development kits (SDKs) present integration difficulties, dependency and version conflicts, debugging challenges, and security vulnerabilities when outdated, while poor SDK documentation leads to faulty implementations with functional and performance consequences (Frantz et al., 2016; Pashchenko et al., 2020). Reliability modeling of open-source software further shows that fault introduction follows heavy-tailed patterns that complicate stabilization (Wang, 2021). Conservative sectors respond by restricting technology choice: government projects favor established languages and avoid most open-source libraries for security reasons (Chen & Huang, 2022), and newly released libraries tend to carry more defects until several years of community testing mature them (Wang, 2021).

6.5 Cloud Platforms and Operations

Cloud technologies add flexibility but also error surface. Resource allocation across co-hosted applications, latency guarantees, and energy-efficient scheduling remain open challenges (Abid et al., 2020), and organizations resist cloud adoption due to complexity, security concerns, and loss of control over data (Hadwer et al., 2021). Fault tolerance is a central requirement, and debugging multi-cloud deployments is described as time-consuming and tedious (Bharany et al., 2022). Monitoring produces overwhelming log volume in which the actual error signal is comparatively small, consuming developer debugging time; deep learning based log ranking has been proposed to mitigate this (Lovas et al., 2023). Load balancing remains the major operational challenge, prompting organizations to adopt gameday-style fault injection exercises to expose weaknesses (Shahid et al., 2020). Emerging technology choices carry their own constraints; blockchain-based designs are resource and energy intensive, and no mature automated testing tooling exists for them (Sedlmeir et al., 2021).

6.6 Hardware and Software Compatibility

Efficient operation requires mutual compatibility between hardware and software; glitches can invoke the wrong hardware resources and cause system failure, while incompatible hardware forces redevelopment for developers, testers, and organizations, making compatibility validation and benchmarking a necessary part of development and testing (Englander & Wong, 2021).

Table 3. *Technological Factors Contributing to Software Errors and Representative Evidence*

Factor	**Key findings**	**Studies**
Tools and IDEs	Error messages guide debugging; tool complexity and failures cost time and induce workarounds; expertise gaps persist	(Lopez et al., 2021; Jayakody & Wijayanayake, 2021; Khan et al., 2022; Haque et al., 2020)
Documentation	Out-of-sync, incorrect, and missing documentation reported by majorities of practitioners; wrong examples mislead developers	(Aghajani et al., 2020; Laplante, 2024)
Architecture	Architecture-code misalignment, complexity, and rigidity-flexibility trade-offs disrupt development	(Taylor, 2019; Tian et al., 2022; Nugraha & Talita, 2021; Silva et al., 2023)
Dependencies and SDKs	Vulnerable and outdated dependencies, version conflicts, and poor SDK documentation introduce defects and vulnerabilities	(Wang et al., 2022; Prana et al., 2021; Pashchenko et al., 2020; Frantz et al., 2016)
Cloud platforms	Resource allocation, log overload, load balancing, and multi-cloud debugging burden developers	(Abid et al., 2020; Lovas et al., 2023; Shahid et al., 2020; Bharany et al., 2022)
Hardware compatibility	Incompatibility causes failures and redevelopment; compatibility benchmarking required	(Englander & Wong, 2021; Sedlmeir et al., 2021)

7. Organizational Factors and Consequences

7.1 Security Practice and Its Deprioritization

A systematic mapping of secure software engineering shows that most development projects do not include security in initial stages; security is deprioritized as post-development work, and coding-stage errors left unknowingly produce denial-of-service exposure, data leaks, remote code execution, and cross-site scripting that place organizations at risk, with loss of customer trust as the leading organizational impact (Khan et al., 2021). Practitioner surveys in Agile contexts confirm that developers have minimal security knowledge and practice, that integrating security into Agile processes is immature, and that coding standards and security design assessment are key to application security (Rindell et al., 2021). Misuse of software has produced financial losses, data losses, communication losses, and even critical infrastructure control issues (Khan et al., 2021).

7.2 Regulatory and Compliance Pressure

Security certification under standards such as ISO/IEC 15408 builds user confidence but imposes complex, time-consuming evaluation processes whose duration grows with technological sophistication, delaying time to market; budget and market share influence whether organizations certify at all, and failure to adopt security-by-design increases product vulnerability (Sun et al., 2022). Regulatory fines are material: organizations violating the General Data Protection Regulation (GDPR) have faced penalties reaching four percent of annual revenue for illegal data usage, missing data protection officers, and absent technical protection procedures (Wolff & Atallah, 2021). A systematic review of software compliance found that roughly 36 percent of compliance discussions originate in the software industry, spanning intellectual property, reliability, GDPR, HIPAA, and data localization requirements (Mubarkoot & Altmann, 2021). Quality benchmarking against ISO standards, including ISO/IEC 5055 for code quality measurement and ISO 81001-1 for healthcare software safety, provides standardized trust signals but requires substantial assessment effort (Siavvas et al., 2021).

7.3 Requirements and Business Knowledge

Lack of business knowledge and improper requirements analysis is one of the key contributors to software errors: unclear problem understanding creates communication gaps with developers and testers, resulting in products built and released with defects (Mishra, 2021). Stakeholder identification and involvement is itself challenging, and delayed or improper requirements from product owners adversely affect development, timelines, cost, and organizational value (Lewellen, 2020). Structured scope definition frameworks have been proposed to address the persistent difficulty of bounding what a project must deliver (Hassan & Asghar, 2021).

7.4 Resourcing, Cost, and Process Maturity

Finding appropriately skilled human resources for each technology is among the hardest parts of software development, and organizations incur high hiring and training expenditure, with budgets frequently exceeding plans (Shaposhnykov et al., 2021). Project failures cost money, time, and reputation, and delays cause customer loss, making accurate planning and correct staffing essential (Hassan & Asghar, 2021). Total quality management practices have been proposed as a buffer between global software development challenges and project success (Hashmi et al., 2021). AI-assisted development is positioned as a way to reduce human resource costs, with large code datasets such as CodeNet supporting learning-based coding tools, although these systems remain at an early experimental stage (Puri et al., 2021). Security-focused organizational consequences also interact with product quality: in IoT device industries, security errors invite ransomware, device manipulation, traffic redirection, denial-of-service and man-in-the-middle attacks, and missing cryptography enables interception, all of which damage product quality, reputation, and revenue (Jurcut et al., 2020).

8. External Environmental Factors Affecting Developers

8.1 Market Pressure and Human Values

The competitive urge to acquire markets with new digital technology accelerates delivery schedules, and this fast-evolving environment causes poor project management, improper SDLC implementation, and weak security consideration, leading to software errors and project failures (Baham & Hirschheim, 2022). Case study research on human values in software engineering documents facial recognition failing to identify people of color, delivery systems misrepresenting delivery times, emissions software manipulation, and surge pricing during a hurricane, and finds that absent regulations and guidelines, value-based design loses priority to time-to-market pressure (Hussain et al., 2022).

8.2 Privacy Requirements and Communication Gaps

Interview studies with software developers and privacy experts reveal a structural communication gap: privacy requirements are written in legal language, unidirectional communication hinders cross-team relationships, privacy experts often lack technical knowledge, and business analysts hand over raw requirements lacking implementation detail that developers then misinterpret (Horstmann et al., 2024). Complementary evidence from Brazilian developers shows that most fail to implement privacy and security due to lack of knowledge of privacy-based design, that privacy and security are commonly confused, that nonfunctional

requirements receive lower priority than functional ones, and that seven external factors influence developers' decisions on nonfunctional requirements (Peixoto et al., 2023).

8.3 Distributed Teams, Coordination, and Knowledge Attrition

In scaled Agile development of physical products, synchronization and coordination are the top reported challenges; dependencies between Agile and non-Agile teams, cross-team knowledge sharing across locations, and resource allocation all affect the development process, and a positive error culture is needed so that frequent feedback does not demotivate developers (Michalides et al., 2023). Knowledge-based resources are organizational assets, and the departure or unavailability of key knowledgeable engineers creates direct challenges for Agile teams (Ouriques et al., 2023). Developers also lack education on specific concerns such as accessibility, leading to development deficiencies when industry priorities overlook such features (Patel et al., 2020).

8.4 User Diversity and Human-Centered Design

End users vary from individuals to organizations and third-party applications, and failure to meet user expectations leads to product failure, financial loss, and lost trust; users may be educated or uneducated and may have vision, hearing, or motor challenges, requiring human-centric design across heterogeneous platforms (Sharma & Singh, 2021). Human-centered design principles call for clear iconography, readable text, and confirmation prompts before destructive actions in financial and medical applications so that users do not make mistakes (Kadir & Broberg, 2021; Melles et al., 2020). Studies of Chinese young-old smartphone users document age-specific difficulties with handwriting input, icon design, color schemes, and AI features, underscoring the breadth of user contexts that software must accommodate (Xiao et al., 2022). Understanding user behavior requires collecting and analyzing interaction data, which is itself challenging for developers (Suonsyrja et al., 2018).

9. Software Testing: Approaches and Limitations

9.1 Conventional Testing Methods

Software testing detects and fixes issues before release, protecting organizations from financial, reputational, and security expenses (Ahmad et al., 2019); finding issues in the testing stage costs far less than fixing them in production, where brand value, customer trust, and market investment are also at stake (Fraser & Rojas, 2019). Established methodologies include unit testing, which validates small functional units against varied inputs (Khorikov, 2020); black-box testing, which validates external behavior without regard to internals, including automated black-box generation for RESTful APIs (Viglianisi et al., 2020); white-box testing, which examines internal functions and logic and can be combined with black-box approaches (Martin-Lopez et al., 2021); and regression testing, whose test suite maintenance is a recognized challenge at industrial scale (Ali et al., 2019). Error-free software is the aim in domains such as healthcare, banking, and supply chains where failures cost human lives or critical value (Aniche, 2022). Testing remains resource-intensive: selecting suitable test cases and deciding which tests to execute is a meticulous responsibility, and evaluating testing techniques systematically remains an active research area (Mayeda & Andrews, 2021). Semantic-web-enabled testing (Dadkhah et al., 2020) and A/B testing (Quin et al., 2024) extend the toolbox, while system-level test of complex hardware-software products faces open challenges (Appello et al., 2021), as does testing of low-code platforms (Khorram et al., 2020) and incremental testing in software product lines (Beyazit et al., 2023). Reliability prediction techniques, including particle swarm optimization, support quality estimation across the life cycle (Habtemariam et al., 2022).

9.2 Script-Based Automation

Automation replaces repetitive manual test execution. Selenium identifies web elements through XPath, identifiers, and class attributes and supports test authoring in multiple languages, but it cannot achieve full automation or identify bugs the way humans do (Izzat & Saleem, 2023; Anand & Prasad, 2022). Automation integrates into continuous integration and continuous delivery pipelines, including dynamic security testing (Rangnau et al., 2020), and comparative studies document the strengths and costs of tools such as Selenium, Appium, and Postman for web, mobile, and API automation (Ateşoğulları & Mishra, 2020). Robotic process automation systems interact with software like humans and can perform testing through bot agents, but they lack theoretical foundations, which limits research adoption (Syed et al., 2020). Unit testing frameworks embedded in modern development stacks catch bugs early in build pipelines, but time constraints, framework design constraints, and multi-team development impede test case writing (Pargaonkar, 2023; Appello et al., 2021). Fundamental analyses of graphical user interface (GUI) test automation conclude that many of its challenges will remain, because scripts are brittle against interface change and demand skilled maintenance (Nass et al., 2021). Overall, writing and maintaining automation for every functionality remains tedious and time-consuming (Samad et al., 2021), and testing benefits from complementary signals such as opinion mining of user feedback during beta programs (Beniwal et al., 2020). Domain-specific practice adds further difficulty; fuzzing of banking applications illustrates the rigor demanded where security is paramount (Schneider et al., 2020), and pandemic-era load surges exposed how hard it is to test the full space of user actions in large banking systems under compressed release timelines (Kwan et al., 2020).

9.3 AI, Machine Learning, and Vision-Based Testing

Machine learning provides supervised, unsupervised, semi-supervised, and reinforcement paradigms that let systems analyze data intelligently without exhaustive programming (Sarker, 2021), with deep learning extending this to automatic feature extraction over high-dimensional data through architectures such as convolutional networks, recurrent networks, and generative adversarial networks (LeCun et al., 2015; Janiesch et al., 2021; Jacob & Darney, 2021). Applied to software engineering, natural language driven code generation and in-IDE code assistants show promise, but generated code quality does not yet match human-written code, and qualitative enthusiasm outpaces experimental results (Xu et al., 2022). AI-assisted code generation also introduces new risks, including insecure code patterns and data leakage incidents associated with AI tool usage (Negri-Ribalta et al., 2024).

Vision-based approaches aim to test software the way humans do, by perceiving the interface. Computer vision has matured for general object detection (Guo et al., 2022; Zhang et al., 2021) and supports quality inspection tasks such as food grading (Hemamalini et al., 2022) and edge deployment (Semitela, 2021), but GUI object detection is a distinct problem: standard detectors such as YOLO underperform on interface imagery because GUIs pack many small elements into limited screen area, and hybrid old-fashioned plus deep learning pipelines currently perform best (Chen et al., 2020). Early results are nonetheless encouraging. A deep learning system integrated with Selenium generated 98.8 percent of test cases correctly for an e-commerce application under test and generalized to a new application (Khaliq et al., 2023). A DenseNet201-based model evaluated mobile user interfaces for defects with approximately 93 percent accuracy (Soui & Haddad, 2023). Machines that test software like humans have been prototyped in industrial research (Dwarakanath et al., 2018), machine learning has been applied to GUI test automation directly (Walia, 2022), and vision-based generation of HTML from sketches shows the reach of the paradigm, although backend and data-layer generation remains far harder (Khandekar et al., 2022). Table 4 summarizes the approaches, their strengths, and their documented limitations.

Table 4. *Testing Approaches, Strengths, and Documented Limitations*

Approach	Strengths	Limitations
Manual testing (unit, black-box, white-box, regression)	Human judgment, exploratory coverage, business context awareness (Khorikov, 2020; Viglianisi et al., 2020; Martin-Lopez et al., 2021)	Labor-intensive, slow, suite maintenance burden at scale, test selection difficulty (Ali et al., 2019; Mayeda & Andrews, 2021)
Script-based automation (Selenium, Appium, RPA, CI/CD)	Repeatable, fast regression, pipeline integration, cross-platform execution (Izzat & Saleem, 2023; Ateşoğulları & Mishra, 2020; Rangnau et al., 2020)	Cannot reach full automation, brittle to UI change, high authoring and maintenance cost, weak theory for RPA (Anand & Prasad, 2022; Nass et al., 2021; Syed et al., 2020; Samad et al., 2021)
AI, ML, CV, and DL based testing	Human-like GUI perception, automatic test generation, high reported accuracy in early studies (Khaliq et al., 2023; Soui & Haddad, 2023; Dwarakanath et al., 2018)	GUI detection immature, generalization unproven at scale, generated code quality gaps, new security risks (Chen et al., 2020; Xu et al., 2022; Negri-Ribalta et al., 2024)

10. Effects of Software Errors on Products and Stakeholders

10.1 Safety-Critical Failures

The most visible consequences of software errors are safety-critical. The Boeing 737 MAX crashes traced to the Maneuvering Characteristics Augmentation System (MCAS), where a hardware problem was addressed through software that adjusted the aircraft nose based on angle-of-attack sensor data, and where basic software reportedly failed minimum quality assessments (Travis, 2019; Johnston & Harris, 2019). In autonomous driving, a widely reported crash occurred when a vision system misinterpreted a white truck, an edge case missing from the test suite, underscoring that verification and validation of autonomous vehicles must cover an extremely wide scenario space (Rajabli et al., 2021). Autonomous vehicle crashes measurably damage public perception of the technology (Penmetsa et al., 2021), and miscalibration plus missing edge cases remain recurrent causes (Penmetsa et al., 2021; Rajabli et al., 2021).

10.2 Consequences by Stakeholder

Beyond catastrophic cases, the literature documents a systematic distribution of harm. Users experience bugs, security and data theft issues, frustration, and productivity loss (Zwilling et al., 2020; Tahaei et al., 2021). Organizations in public and private sectors suffer operational disruption, revenue loss, and reputational damage that propagates to customers and business partners (Khando et al., 2021), and software defects negatively affect reliability growth and dependent systems (Duarte et al., 2018). Developers absorb altered timelines, on-the-spot decision pressure, and increased workload and stress, with psychological impact from rework

on the same issues (Tahaei et al., 2021; Rauf et al., 2021); human error management research asks whether interventions should target people, processes, or the environment (Mahaju et al., 2023). Table 5 summarizes these stakeholder consequences.

Table 5. *Consequences of Software Errors by Stakeholder Group*

Stakeholder	Documented consequences	Studies
Users	Functional failures, security and data theft exposure, frustration, productivity loss, physical risk in safety-critical and health systems	(Zwilling et al., 2020; Tahaei et al., 2021; Manninger et al., 2021; Rajabli et al., 2021)
Organizations	Financial loss, reputational damage, operational disruption, regulatory fines, lost customer trust, technical debt growth	(Khando et al., 2021; Khan et al., 2021; Wolff & Atallah, 2021; Krasner, 2021)
Developers	Timeline disruption, rework, increased workload and stress, forced rapid decisions, psychological strain	(Tahaei et al., 2021; Rauf et al., 2021; Mahaju et al., 2023)
Dependent systems	Propagated failures in systems that consume defective software outputs	(Duarte et al., 2018; Stanley-Marbell et al., 2021)

11. Research Gaps and Future Directions

The reviewed literature extensively documents what software errors cost, where they surface, and how testing intercepts a portion of them. It is comparatively silent on why errors are introduced, viewed from the position of the people who introduce them. Synthesizing Sections 4 through 10, five gaps emerge.

11.1 Gap 1: Root Causes from the Developer's Perspective Across TOE Dimensions

Most studies analyze errors from artifact-centric or user-centric viewpoints; very few study the perspectives of developers themselves (Peixoto et al., 2023), and researchers have addressed human problem solving and error introduction without studying the everyday challenges developers face during development (Lopez et al., 2021). Critical challenges faced by developers in terms of technology, security implications for organizations, and environmental effects have not been fully explored with respect to software errors (Appello et al., 2021; Khorram et al., 2020). Because the developer's actions of writing code, analyzing code, and choosing tools and requirements are the origin points of software defects (Rauf et al., 2021; Tahaei et al., 2021), a qualitative phenomenological study of developers who have lived the software error phenomenon, structured by the TOE constructs, is the natural methodological response (Creswell & Creswell, 2018; Meyers, 2023).

11.2 Gap 2: Integrated Multidimensional Analysis Rather Than Single-Factor Studies

Existing evidence isolates individual factors: documentation (Aghajani et al., 2020), dependencies (Prana et al., 2021), DevOps adoption (Khan et al., 2022), or privacy communication (Horstmann et al., 2024). No integrated account explains how technological, organizational, and environmental pressures interact to produce errors in a given development setting, even though software development involves TOE constructs interacting internally (Hassan & Khan, 2021). Multi-concern assurance work signals movement in this direction at the design level (Jaskolka et al., 2022), but empirical integration across the three contexts remains absent.

11.3 Gap 3: Human-Like Automated Testing Remains Immature

Humans detect on-screen errors, abnormal behavior, and functional deviations with minimal instruction and provide rich feedback (Beniwal et al., 2020), but replicating this remains open. GUI-specific object detection is in the beginning phase of research (Chen et al., 2020), script automation cannot reach complete coverage and will retain fundamental challenges (Nass et al., 2021; Anand & Prasad, 2022), and although early deep learning results report high accuracy (Khaliq et al., 2023; Soui & Haddad, 2023), generalization across application types, platforms, and dynamic interfaces is unestablished. Research is needed on vision-based systems that understand GUI images, infer functionality, and generate functional test cases the way humans do (Walia, 2022; Dwarakanath et al., 2018).

11.4 Gap 4: Security as a First-Class SDLC Concern

Security continues to be deprioritized to post-development stages (Khan et al., 2021), mature models combining security with Agile development are lacking (Rindell et al., 2021), and nonfunctional requirements receive lower priority than functional ones (Peixoto et al., 2023). Research is needed on organizational mechanisms that make security-by-design economically rational under time-to-market pressure, connecting certification cost evidence (Sun et al., 2022) with developer-level practice.

11.5 Gap 5: Error Effects Quantified at the Product Functionality Level

Cost aggregates exist at the national level (Krasner, 2021) and severity classifiers exist at the defect level (Liu et al., 2023; Ni et al., 2022), but the middle layer, that is, how increased error rates translate into measurable degradation of specific product functionalities and downstream organizational security consequences, is not systematically modeled. Ontological groundwork exists (Duarte et al., 2018) and reliability models capture fault introduction statistically (Wang, 2021; Habtemariam et al., 2022), but linking these to functionality-level outcomes across the TOE contexts remains open. Table 6 summarizes the gaps and corresponding research directions.

Table 6. *Identified Research Gaps and Proposed Research Directions*

No.	Research gap	Proposed direction
1	Root causes of software errors unexplored from developers' lived perspective across TOE dimensions (Lopez et al., 2021; Peixoto et al., 2023)	Qualitative phenomenological studies of experienced developers structured by TOE constructs (Creswell & Creswell, 2018; Meyers, 2023)
2	Single-factor studies; no integrated technological, organizational, environmental analysis (Hassan & Khan, 2021; Jaskolka et al., 2022)	Multidimensional empirical designs capturing factor interactions within one development setting
3	Human-like automated testing immature; GUI perception in early research phase (Chen et al., 2020; Nass et al., 2021)	Vision and deep learning systems that recognize GUI functionality and generate functional test cases like humans (Khaliq et al., 2023; Walia, 2022)
4	Security deprioritized in SDLC; no mature secure-Agile integration (Khan et al., 2021; Rindell et al., 2021)	Organizational mechanisms making security-by-design viable under market pressure (Sun et al., 2022)
5	No systematic model linking error rates to functionality degradation and security consequences (Duarte et al., 2018; Krasner, 2021)	Functionality-level impact models connecting defect data, reliability models, and organizational outcomes (Wang, 2021; Habtemariam et al., 2022)

12. Conclusion

This review synthesized the literature on software errors and their effects on software product functionality using the TOE framework as an organizing lens. The evidence shows that errors arise from an interacting system of technological conditions, including tool complexity, documentation defects, architectural misalignment, vulnerable dependencies, and cloud operational burden; organizational conditions, including deprioritized security, compliance cost, weak requirements understanding, and resourcing constraints; and environmental conditions, including market pressure, distributed coordination, regulatory gaps, and diverse user populations. Testing intercepts only part of this flow: manual methods are labor-bound, script automation is brittle and incomplete, and AI-based human-like testing, despite promising early accuracy results, is at the beginning of its research trajectory. The consequences of the errors that escape are borne by users, organizations, developers, and dependent systems, and in safety-critical domains by human lives.

Five research gaps were identified, the most fundamental being the absence of research into the root causes of software errors from the lived perspective of software developers across the technological, organizational, and environmental contexts simultaneously. Addressing this gap through qualitative phenomenological inquiry, complemented by integrated multidimensional designs, human-like vision-based testing research, security-by-design mechanisms, and functionality-level impact modeling, may inform organizational leaders, tool builders, and researchers about the deficiencies that allow errors to persist, and may ultimately improve software quality, reduce error rates, and produce a more resilient software development process.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

ORCID iD: Dr. Rahul Azmeera, https://orcid.org/0000-0003-4454-1426

References

[1] Abid, A., Manzoor, M. F., Farooq, M., Farooq, U., & Hussain, M. (2020). Challenges and issues of resource allocation techniques in cloud computing. Transactions on Internet and Information Systems, 14(7). https://doi.org/10.3837/tiis.2020.07.005

[2] Aghajani, E., Nagy, C., Linares-Vásquez, M., Moreno, L., Bavota, G., Lanza, M., & Shepherd, D. C. (2020). Software documentation: The practitioners' perspective. Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering (ICSE '20), 590– 601. https://doi.org/10.1145/3377811.3380405

[3] Ahmad, J., ul Hassan, A., Naqvi, T., & Mubeen, T. (2019). A review on software testing and its methodology. i-Manager's Journal on Software Engineering, 13(3), 32–38. https://doi.org/10.26634/jse.13.3.15515

[4] Alami, A., & Krancher, O. (2022). How Scrum adds value to achieving software quality? Empirical Software Engineering, 27(7). https://doi.org/10.1007/s10664-022-10208-4

[5] Alexandrova, A., & Rapanotti, L. (2019). Requirements analysis gamification in legacy system replacement projects. Requirements Engineering, 25(2), 131–151. https://doi.org/10.1007/s00766-019-00311-2

[6] Ali, N. B., Engström, E., Taromirad, M., Mousavi, M. R., Minhas, N. M., Helgesson, D., & Varshosaz, M. (2019). On the search for industry-relevant regression testing research. Empirical Software Engineering, 24(4), 2020–2055. https://doi.org/10.1007/s10664-018-9670-1

[7] Anand, S. G., & Prasad, L. (2022). Impact of Selenium in automation testing? Challenges and opportunities. NeuroQuantology, 20(22), 2789–2801. https://www.proquest.com/docview/2900743716?pq-origsite=gscholar&fromopenview=true&sourcetype=Scholarly%20Journals

[8] Andrade, S. A., Nunes, F. L. S., & Delamaro, M. E. (2019). Towards the systematic testing of virtual reality programs. 2019 21st Symposium on Virtual and Augmented Reality (SVR). https://doi.org/10.1109/svr.2019.00044

[9] Aniche, M. (2022). Effective software testing: A developer's guide. Simon and Schuster. https://livebook.manning.com/book/effective-software-testing

[10] Appello, D., Chen, H. H., Sauer, M., Polian, I., Bernardi, P., & Reorda, M. S. (2021). System level test: State of the art and challenges. 2021 IEEE 27th International Symposium on On-Line Testing and Robust System Design (IOLTS). https://doi.org/10.1109/iolts52814.2021.9486708

[11] Arrey, D. A. (2019). Exploring the integration of security into software development life cycle (SDLC) methodology (Order No. 13428588). ProQuest Dissertations & Theses Global. https://www.proquest.com/dissertations-theses/exploring-integration-security-into-software/docview/2191192923/se-2

[12] Ateşoğulları, D., & Mishra, A. (2020). Automation testing tools: A comparative view. International Journal on Information Technologies & Security, 12(4), 63–76. https://hdl.handle.net/11250/3095653

[13] Baham, C., & Hirschheim, R. (2022). Issues, challenges, and a proposed theoretical core of agile software development research. Information Systems Journal, 32(1), 103-129. https://doi.org/10.1111/isj.12336

[14] Baker, J. (2011). The technology–organization–environment framework. Information Systems Theory, 231–245. https://doi.org/10.1007/978-1-4419-6108-2_12

[15] Beniwal, R., Jain, M., & Gupta, Y. (2020). Opinion mining to aid user acceptance testing for open beta versions. Proceedings of International Conference on Artificial Intelligence and Applications, 291–301. https://doi.org/10.1007/978-981-15-4992-2_28

[16] Beyazit, M., Tuglular, T., & Kaya, D. Ö. (2023). Incremental testing in Software Product Lines—An event based approach. IEEE Access, 11, 2384–2395. https://doi.org/10.1109/access.2023.3234186

[17] Bharany, S., Badotra, S., Sharma, S., Rani, S., Alazab, M., Jhaveri, R. H., & Gadekallu, T. R. (2022). Energy efficient fault tolerance techniques in green cloud computing: A systematic survey and taxonomy. Sustainable Energy Technologies and Assessments, 53, Article 102613. https://doi.org/10.1016/j.seta.2022.102613

[18] Biable, S. E., Garcia, N. M., Midekso, D., & Pombo, N. (2022). Ethical issues in software requirements engineering. Software, 1(1), 31–52. https://doi.org/10.3390/software1010003

[19] Boisvert, R. F., Cools, R., & Einarsson, B. (2005). 2. Assessment of accuracy and reliability. In Accuracy and Reliability in Scientific Computing (pp. 13–32). https://doi.org/10.1137/1.9780898718157.ch2

[20] Bojanova, I., & Galhardo, C. E. C. (2023). Bug, fault, error, or weakness: Demystifying software security vulnerabilities. IT Professional, 25(1), 7–12. https://doi.org/10.1109/mitp.2023.3238631

[21] Bosch, J., & Olsson, H. H. (2021). Digital for real: A multicase study on the digital transformation of companies in the embedded systems domain. Journal of Software: Evolution and Process, 33(5). https://doi.org/10.1002/smr.2333

[22] Boulos, M., Wheeler, S., Tavares, C., & Jones, R. (2011). How smartphones are changing the face of mobile and participatory healthcare: An overview, with example from eCAALYX. BioMedical Engineering OnLine, 10(1), Article 24. https://doi.org/10.1186/1475-925x-10-24

[23] Chang, C., Kontovas, C., Yu, Q., & Yang, Z. (2021). Risk assessment of the operations of maritime autonomous surface ships. Reliability Engineering & Systems Safety, 207, Article 107324. https://doi.org/10.1016/j.ress.2020.107324

[24] Chen, M., Yang, J., Zhu, X., Wang, X., Liu, M., & Song, J. (2017). Smart Home 2.0: Innovative smart home system powered by botanical IoT and emotion detection. Mobile Networks and Applications, 22(6), 1159–1169. https://doi.org/10.1007/s11036-017-0866-1

[25] Chen, J., Xie, M., Xing, Z., Chen, C., Xu, X., Zhu, L., & Li, G. (2020). Object detection for graphical user interface: Old fashioned or deep learning or a combination? Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. https://doi.org/10.1145/3368089.3409691

[26] Chen, Y., & Huang, C. (2022, March). Java implementation of business management system and information analysis platform for economic innovation projects. In 2022 6th International Conference on Computing Methodologies and Communication (ICCMC) (pp. 1497-1500). IEEE. https://doi.org/10.1109/ICCMC53470.2022.9753768

[27] Creswell, J. W., & Creswell, D. J. (2018). Research design: Qualitative, quantitative, and mixed methods approaches (5th ed.). SAGE Publications, Inc.

[28] Cusumano, M. A., Yoffie, D. B., & Gawer, A. (2020). The future of platforms. MIT Sloan Management Review, 46-54. https://sloanreview.mit.edu/article/the-future-of-platforms/

[29] Dadkhah, M., Araban, S., & Paydar, S. (2020). A systematic literature review on semantic web enabled software testing. Journal of Systems and Software, 162, Article 110485. https://doi.org/10.1016/j.jss.2019.110485

[30] Dantu, R., Dissanayake, I., & Nerur, S. (2020). Exploratory analysis of Internet of Things (IoT) in healthcare: A topic modelling & co-citation approaches. Information Systems Management, 38(1), 62–78. https://doi.org/10.1080/10580530.2020.1746982

[31] Duarte, B. B., De Almeida Falbo, R., Guizzardi, G., & Souza, V. E. S. (2018). Towards an ontology of software defects, errors, and failures. ResearchGate. https://www.researchgate.net/publication/325988886_Towards_an_Ontology_of_Software_Defects_Errors_and_Failures

[32] Dwarakanath, A., Dubash, N., & Podder, S. (2018). Machines that test software like humans. arXiv (Cornell University). https://doi.org/10.48550/arxiv.1809.09455

[33] Englander, I., & Wong, W. (2021). The architecture of computer hardware, systems software, and networking: An information technology approach (6th ed.). John Wiley & Sons. https://bit.ly/3X5nkjn

[34] Forelle, M. (2022). The material consequences of "chipification": The case of software- embedded cars. Big Data & Society, 9(1), Article 205395172210954. https://doi.org/10.1177/20539517221095429

[35] Frantz, R. Z., Corchuelo, R., & Roos-Frantz, F. (2016). On the design of a maintainable software development kit to implement integration solutions. Journal of Systems and Software, 111, 89–104. https://doi.org/10.1016/j.jss.2015.08.044

[36] Fraser, G., & Rojas, J. M. (2019). Software testing. In Handbook of Software Engineering (pp. 123–192). https://doi.org/10.1007/978-3-030-00262-6_4

[37] Gheorghe, D. (2022). Development of artificial intelligence in the automotive field. Informatica Economica, 26(2), 29–36. https://doi.org/10.24818/issn14531305/26.2.2022.03

[38] Gillath, O., Ai, T., Branicky, M. S., Keshmiri, S., Davison, R. B., & Spaulding, R. (2021). Attachment and trust in artificial intelligence. Computers in Human Behavior, 115, Article 106607. https://doi.org/10.1016/j.chb.2020.106607

[39] Guo, M. H., Xu, T. X., Liu, J. J., Liu, Z. N., Jiang, P. T., Mu, T. J., Zhang, S. H., Martin, R. R., Cheng, M. M., & Hu, S. M. (2022). Attention mechanisms in computer vision: A survey. Computational Visual Media, 8(3), 331–368. https://doi.org/10.1007/s41095-022-0271-y

[40] Habtemariam, G. M., Mohapatra, S. K., & Seid, H. (2022). Prediction of software reliability using particle swarm optimization. Communications in Computer and Information Science, 148–156. https://doi.org/10.1007/978-3-031-23233-6_11

[41] Hadwer, A. A., Tavana, M., Gillis, D., & Rezania, D. (2021). A systematic review of organizational factors impacting cloud-based technology adoption using the Technology- Organization-Environment framework. Internet of Things, 15, Article 100407. https://doi.org/10.1016/j.iot.2021.100407

[42] Haque, M. U., Iwaya, L. H., & Babar, M. A. (2020). Challenges in Docker development. Proceedings of the 14th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (ESEM '20). Association for Computing Machinery, New York, NY, USA, Article 7, 1–11. https://doi.org/10.1145/3382494.3410693

[43] Hashmi, S. D., Shahzad, K., & Izhar, M. (2021). Proposing total quality management as a buffer between global software development challenges and project success. The TQM Journal, ahead-of-print. https://doi.org/10.1108/TQM-08-2020-0192

[44] Hassan, C. A. U., & Khan, M. S. (2021). An effective nature-inspired approach for the estimation of software development cost. 2021 16th International Conference on Emerging Technologies (ICET), 1–6. https://doi.org/10.1109/icet54505.2021.9689832

[45] Hassan, I. U., & Asghar, S. (2021). A framework of software project scope definition elements: An ISM-DEMATEL approach. IEEE Access, 9, 26839–26870. https://doi.org/10.1109/access.2021.3057099

[46] Hein, A., Böhm, M., & Krcmar, H. (2018). Tight and loose coupling in evolving platform ecosystems: The cases of Airbnb and Uber. Business Information Systems, 295–306. https://doi.org/10.1007/978-3-319-93931-5_21

[47] Hemamalini, V., Rajarajeswari, S., Nachiyappan, S., Sambath, M., Devi, T., Singh, B. K., & Raghuvanshi, A. (2022). Food quality inspection and grading using efficient image segmentation and machine learning-based system. Journal of Food Quality, 2022, 1–6. https://doi.org/10.1155/2022/5262294

[48] Horani, O. M., Al-Adwan, A. S., Yaseen, H., Hmoud, H., Al-Rahmi, W. M., & Alkhalifah, A. (2023). The critical determinants impacting artificial intelligence adoption at the organizational level. Information Development, Article 02666669231166889. https://doi.org/10.1177/02666669231166889

[49] Horstmann, S. A., Domiks, S., Gutfleisch, M., Tran, M., Acar, Y., Moonsamy, V., & Naiakshina, A. (2024). "Those things are written by lawyers, and programmers are reading that." Mapping the communication gap between software developers and privacy experts. Proceedings on Privacy Enhancing Technologies, 2024(1), 151–170. https://doi.org/10.56553/popets-2024-0010

[50] Hussain, W., Perera, H., Whittle, J., Nurwidyantoro, A., Hoda, R., Shams, R. A., & Oliver, G. (2022). Human values in software engineering: Contrasting case studies of practice. IEEE Transactions on Software Engineering, 48(5), 1818–1833. https://doi.org/10.1109/tse.2020.3038802

[51] Izzat, S., & Saleem, N. N. (2023). Software testing techniques and tools: A review. Mağallaẗ Al- tarbiyaẗ Wa-al-ʻilm, 32(2), 31–40. https://doi.org/10.33899/edusj.2023.137480.1305

[52] Jacob, I. J., & Darney, P. E. (2021). Design of deep learning algorithm for IoT application by image-based recognition. Journal of ISMAC, 3(03), 276-290. https://doi.org/10.36548/jismac.2021.3.008

[53] Janiesch, C., Zschech, P., & Heinrich, K. (2021). Machine learning and deep learning. Electronic Markets, 31(3), 685–695. https://doi.org/10.1007/s12525-021-00475-2

[54] Jaskolka, J., Hamid, B., & Kokaly, S. (2022). Software design trends supporting multiconcern assurance. IEEE Software, 39(4), 22–26. https://doi.org/10.1109/ms.2022.3164872

[55] Jayakody, J. a. V. M. K., & Wijayanayake, W. (2021). Challenges for adopting DevOps in information technology projects. 2021 International Research Conference on Smart Computing and Systems Engineering (SCSE), Colombo, Sri Lanka, 203–210. https://doi.org/10.1109/scse53661.2021.9568348

[56] Johnston, P., & Harris, R. (2019). The Boeing 737 MAX saga: Lessons for software organizations. Software Quality Professional, 21(3), 4-12. https://www.proquest.com/scholarly-journals/boeing-737-max-saga-lessons-software/docview/2246851715/se-2

[57] Jurcut, A., Niculcea, T., Ranaweera, P., & Le-Khac, N. (2020). Security considerations for Internet of Things: A survey. SN Computer Science/SN Computer Science, 1(4). https://doi.org/10.1007/s42979-020-00201-3

[58] Kadir, B. A., & Broberg, O. (2021). Human-centered design of work systems in the transition to Industry 4.0. Applied Ergonomics, 92, Article 103334. https://doi.org/10.1016/j.apergo.2020.103334

[59] Khaliq, Z. (2022, January 14). Artificial intelligence in software testing: Impact, problems, challenges, and prospects. arXiv.org. https://arxiv.org/abs/2201.05371

[60] Khaliq, Z., Khan, D., & Farooq, S. U. (2023). Using deep learning for Selenium web UI functional tests: A case-study with e-commerce applications. Engineering Applications of Artificial Intelligence, 117, Article 105446. https://doi.org/10.1016/j.engappai.2022.105446

[61] Khan, R. A., Khan, S. U., Khan, H. U., & Ilyas, M. (2021). Systematic mapping study on security approaches in secure software engineering. IEEE Access, 9, 19139–19160. https://doi.org/10.1109/access.2021.3052311

[62] Khan, M. S., Khan, A. W., Khan, F., Khan, M. A., & Whangbo, T. (2022). Critical challenges to adopt DevOps culture in software organizations: A systematic review. IEEE Access, 10, 14339–14349. https://doi.org/10.1109/access.2022.3145970

[63] Khandekar, S., Korade, S., Kulkarni, R., Pathak, T., & Kamble, S. (2022). Automatic HTML code generation using image processing. In ICT Analysis and Applications (pp. 709-717). Springer, Singapore. https://doi.org/10.1007/978-981-16-5655-2_68

[64] Khando, K., Gao, S., Islam, S. M., & Salman, A. (2021). Enhancing employees' information security awareness in private and public organizations: A systematic literature review. Computers & Security, 106, Article 102267. https://doi.org/10.1016/j.cose.2021.102267

[65] Khorikov, V. (2020). Unit testing principles, practices, and patterns. Simon and Schuster. https://livebook.manning.com/book/unit-testing/about

[66] Khorram, F., Mottu, J., & Sunyé, G. (2020). Challenges & opportunities in low-code testing. Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (MODELS '20), 1–10. https://doi.org/10.1145/3417990.3420204

[67] Knight, J. C. (2002). Software challenges in aviation systems. In International Conference on Computer Safety, Reliability, and Security (pp. 106-112). Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-45732-1_12

[68] Koumaras, H., Makropoulos, G., Batistatos, M., Kolometsos, S., Gogos, A., Xilouris, G., Sarlas, A., & Kourtis, M. A. (2021). 5G-enabled UAVs with command and control software component at the edge for supporting energy efficient opportunistic networks. Energies, 14(5), Article 1480. https://doi.org/10.3390/en14051480

[69] Krasner, H. (2021). The cost of poor software quality in the US: A 2020 report. Proc. Consortium Inf. Softw. Quality (CISQTM). https://www.it-cisq.org/cisq-files/pdf/CPSQ-2020-report.pdf

[70] Kucuk, B., & Tuzun, E. (2021). Characterizing duplicate bugs: An empirical analysis. 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), 661–668. https://doi.org/10.1109/saner50967.2021.00084

[71] Kumari, M., Misra, A., Misra, S., Fernandez Sanz, L., Damasevicius, R., & Singh, V. B. (2019). Quantitative quality evaluation of software products by considering summary and comments entropy of a reported bug. Entropy, 21(1), Article 91. MDPI AG. https://doi.org/10.3390/e21010091

[72] Kwan, A., Lin, C., Pursiainen, V., & Tai, M. (2020). Stress testing banks' digital capabilities: Evidence from the COVID-19 pandemic. SSRN Electronic Journal. https://doi.org/10.2139/ssrn.3694288

[73] Laplante, P. (2024). Use a pencil: On writing software documentation well and the role of autodocumentation. Computer, 57(5), 102–105. https://doi.org/10.1109/mc.2024.3374008

[74] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436–444. https://doi.org/10.1038/nature14539

[75] Lewellen, S. (2020). Identifying key stakeholders as part of requirements elicitation in software ecosystems. Proceedings of the 24th ACM International Systems and Software Product Line Conference - Volume B, October 2020 (pp. 88–95). https://doi.org/10.1145/3382026.3431249

[76] Liu, J., Liang, C., Feng, J., Xiao, A., Hui, Z., Wu, Q., & Yu, T. (2023). A multi-feature fusion- based automatic detection method for high-severity defects. Electronics, 12(14), Article 3075. https://doi.org/10.3390/electronics12143075

[77] Lopez, T., Sharp, H., Petre, M., & Nuseibeh, B. (2021). Bumps in the code: Error handling during software development. IEEE Software, 38(3), 26–34. https://doi.org/10.1109/ms.2020.3024981

[78] Lovas, R., Rigó, E., Unyi, D., & Gyires-Tóth, B. (2023). Experiences with deep learning enhanced steering mechanisms for debugging of fundamental cloud services. IEEE Access, 11, 26403–26418. https://doi.org/10.1109/access.2023.3243201

[79] Madison, A., Arestides, A., Harold, S., Gurchiek, T., Chang, K., Ries, A., & Tossell, C. (2021, April). The design and integration of a comprehensive measurement system to assess trust in automated driving. In 2021 Systems and Information Engineering Design Symposium (SIEDS) (pp. 1-6). IEEE. https://doi.org/10.1109/SIEDS52267.2021.9483758

[80] Mahaju, S., Carver, J. C., & Bradshaw, G. L. (2023). Human error management in requirements engineering: Should we fix the people, the processes, or the environment? arXiv (Cornell University). https://doi.org/10.48550/arxiv.2304.02702

[81] Manninger, M., Zweiker, D., Svennberg, E., Chatzikyriakou, S., Pavlovic, N., Zaman, J. A., & Duncker, D. (2021). Current perspectives on wearable rhythm recordings for clinical decision-making: The wEHRAbles 2 survey. EP Europace, 23(7), 1106–1113. https://doi.org/10.1093/europace/euab064

[82] Marques, R., Costa, G., Silva, M., & Gonçalves, P. (2017). A survey of failures in the software development process. In Proceedings of the 25th European Conference on Information Systems (ECIS), Guimarães, Portugal, June 5-10, 2017 (pp. 2445–2459). ISBN 978-989- 20-7655-3 Research Papers. https://aisel.aisnet.org/ecis2017_rp/155

[83] Martin-Lopez, A., Arcuri, A., Segura, S., & Ruiz-Cortés, A. (2021). Black-box and white-box test case generation for RESTful APIs: Enemies or allies? In 2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE) (pp. 231–241). IEEE. https://doi.org/10.1109/ISSRE52982.2021.00034

[84] Mayeda, M., & Andrews, A. (2021). Evaluating software testing techniques: A systematic mapping study. In Advances in Computers (pp. 41–114). https://doi.org/10.1016/bs.adcom.2021.01.002

[85] Melles, M., Albayrak, A., & Goossens, R. (2020). Innovating health care: Key characteristics of human-centered design. International Journal for Quality in Health Care, 33(Supplement_1), 37–44. https://doi.org/10.1093/intqhc/mzaa127

[86] Meyers, B. S. (2023). Human error assessment in software engineering (Order No. 30811280). Available from ProQuest Dissertations & Theses Global. (2899532737). https://www.proquest.com/dissertations-theses/human-error-assessment-software-engineering/docview/2899532737/se-2

[87] Michalides, M., Bursac, N., Nicklas, S. J., Weiss, S., & Paetzold, K. (2023). Analyzing current challenges on scaled agile development of physical products. Procedia CIRP, 119, 1188– 1197. https://doi.org/10.1016/j.procir.2023.02.188

[88] Mishra, D. (2021). A study of business knowledge requirements for software projects. Journal of Global Operations and Strategic Sourcing, 14(2), 291–311. https://doi.org/10.1108/jgoss-07-2020-0041

[89] Mosolygo, B., Vandor, N., Antal, G., & Hegedus, P. (2021). On the rise and fall of simple stupid bugs: A life-cycle analysis of SSTUBs. 2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR), Madrid, Spain, 2021, 495–499. https://doi.org/10.1109/msr52588.2021.00061

[90] Mubarkoot, M., & Altmann, J. (2021). Software compliance in different industries: A systematic literature review. https://ceur-ws.org/Vol-2966/paper3.pdf

[91] Nakrani, N. M., & Joshi, M. M. (2021). A human-like decision intelligence for obstacle avoidance in autonomous vehicle parking. Applied Intelligence, 52(4), 3728–3747. https://doi.org/10.1007/s10489-021-02653-3

[92] Nass, M., Alégroth, E., & Feldt, R. (2021). Why many challenges with GUI test automation (will) remain. Information and Software Technology, 138, Article 106625. https://doi.org/10.1016/j.infsof.2021.106625

[93] Negri-Ribalta, C., Geraud-Stewart, R., Sergeeva, A., & Lenzini, G. (2024). A systematic literature review on the impact of AI models on the security of code generation. Frontiers in Big Data, 7. https://doi.org/10.3389/fdata.2024.1386720

[94] Ni, X., Zheng, J., Yu, X., Xu, J., & Li, L. (2022). Predicting severity of software vulnerability based on BERT-CNN. 2022 International Conference on Computer Engineering and Artificial Intelligence (ICCEAI). https://doi.org/10.1109/icceai55464.2022.00151

[95] Nugraha, A. R., & Talita, A. S. (2021). Mobile application architecture restructuring with microservice approach. JITeCS (Journal of Information Technology and Computer Science), 5(3). https://doi.org/10.25126/jitecs.202053239

[96] Ouriques, R., Wnuk, K., Gorschek, T., & Svensson, R. B. (2023). The role of knowledge-based resources in Agile Software Development contexts. Journal of Systems and Software, 197, Article 111572. https://doi.org/10.1016/j.jss.2022.111572

[97] Pargaonkar, S. (2023). A study on the benefits and limitations of software testing principles and techniques: Software quality engineering. International Journal of Scientific and Research Publications, 13(8), 149–155. https://doi.org/10.29322/ijsrp.13.08.2023.p14018

[98] Pashchenko, I., Vu, D., & Massacci, F. (2020). A qualitative study of dependency management and its security implications. Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security, 1513–1531. https://doi.org/10.1145/3372297.3417232

[99] Patel, R., Breton, P., Baker, C. M., El-Glaly, Y. N., & Shinohara, K. (2020). Why software is not accessible: Technology professionals' perspectives and challenges. Extended Abstracts of the 2020 CHI Conference on Human Factors in Computing Systems (CHI EA '20), 1–9. https://doi.org/10.1145/3334480.3383103

[100] Peixoto, M., Ferreira, D., Cavalcanti, M., Silva, C., Vilela, J., Araújo, J., & Gorschek, T. (2023). The perspective of Brazilian software developers on data privacy. Journal of Systems and Software, 195, Article 111523. https://doi.org/10.1016/j.jss.2022.111523

[101] Penmetsa, P., Sheinidashtegol, P., Musaev, A., Adanu, E. K., & Hudnall, M. (2021). Effects of the autonomous vehicle crashes on public perception of the technology. IATSS Research, 45(4), 485-492. https://doi.org/10.1016/j.iatssr.2021.04.003

[102] Prana, G. a. A., Sharma, A., Shar, L. K., Foo, D., Santosa, A. E., Sharma, A., & Lo, D. (2021). Out of sight, out of mind? How vulnerable dependencies affect open-source projects. Empirical Software Engineering, 26(4). https://doi.org/10.1007/s10664-021-09959-3

[103] Puri, R., Kung, D. S., Janssen, G., Zhang, W., Domeniconi, G., Zolotov, V., Dolby, J., Chen, J., Choudhury, M., Decker, L., Thost, V., Buratti, L., Pujar, S., Ramji, S., Finkler, U., Malaika, S., & Reiss, F. (2021). CodeNet: A large-scale AI for code dataset for learning a diversity of coding tasks. arXiv. https://doi.org/10.48550/arxiv.2105.12655

[104] Quin, F., Weyns, D., Galster, M., & Silva, C. C. (2024). A/B testing: A systematic literature review. Journal of Systems and Software, 211, Article 112011. https://doi.org/10.1016/j.jss.2024.112011

[105] Rahmati, P., Tafti, A., Westland, J. C., & Hidalgo, C. (2021). When all products are digital: Complexity and intangible value in the ecosystem of digitizing firms. MIS Quarterly, 45(3), 1025–1058. https://doi.org/10.25300/misq/2021/15384

[106] Rajabli, N., Flammini, F., Nardone, R., & Vittorini, V. (2021). Software verification and validation of safe autonomous cars: A systematic literature review. IEEE Access, 9, 4797–4819. https://doi.org/10.1109/access.2020.3048047

[107] Rangnau, T., Buijtenen, R. V., Fransen, F., & Turkmen, F. (2020). Continuous security testing: A case study on integrating dynamic security testing tools in CI/CD pipelines. 2020 IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC), Eindhoven, Netherlands, 145–154. https://doi.org/10.1109/edoc49727.2020.00026

[108] Rauf, I., Petre, M., Tun, T., Lopez, T., Lunn, P., Van Der Linden, D., Towse, J., Sharp, H., Levine, M., Rashid, A., & Nuseibeh, B. (2021). The case for adaptive security interventions. ACM Transactions on Software Engineering and Methodology, 31(1), 1– 52. https://doi.org/10.1145/3471930

[109] Rindell, K., Ruohonen, J., Holvitie, J., Hyrynsalmi, S., & Leppänen, V. (2021). Security in agile software development: A practitioner survey. Information and Software Technology, 131, Article 106488. https://doi.org/10.1016/j.infsof.2020.106488

[110] Samad, A., Nafis, M. T., Rahmani, S., & Sohail, S. S. (2021, April). A cognitive approach in software automation testing. In Proceedings of the International Conference on Innovative Computing & Communication (ICICC). https://doi.org/10.2139/ssrn.3834262

[111] Sarker, I. H. (2021). Machine learning: Algorithms, real-world applications and research directions. SN Computer Science, 2(3). https://doi.org/10.1007/s42979-021-00592-x

[112] Schneider, M. A., Wendland, M. F., Akin, A., & Senturk, S. (2020). Fuzzing of mobile application in the banking domain: A case study. 2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C). https://doi.org/10.1109/qrs-c51114.2020.00087

[113] Sedlmeir, J., Buhl, H. U., Fridgen, G., & Keller, R. (2021). Recent developments in blockchain technology and their impact on energy consumption. arXiv. https://doi.org/10.48550/arXiv.2102.07886

[114] Semitela, A. F. C. (2021). Computer vision on the edge (Master's thesis). University of Coimbra. https://hdl.handle.net/10316/97997

[115] Seo, E., Bae, S., Choi, H., & Choi, D. (2020). Preference and usability of smart-home services and items - A focus on the smart-home living-lab. Journal of Asian Architecture and Building Engineering, 20(6), 650–662. https://doi.org/10.1080/13467581.2020.1812397

[116] Shahid, M. A., Islam, N., Alam, M. M., Su'ud, M. M., & Musa, S. (2020). A comprehensive study of load balancing approaches in the cloud computing environment and a novel fault tolerance approach. IEEE Access, 8, 130500–130526. https://doi.org/10.1109/access.2020.3009184

[117] Shaposhnykov, K., Kochubei, O., Grygor, O., Protsenko, N., Vyshnevska, O., & Dzyubina, A. (2021). Organizational and economic mechanism of development and promotion of IT products in Ukraine. Studies of Applied Economics, 39(6). https://doi.org/10.25115/eea.v39i6.5264

[118] Sharma, R., & Singh, J. N. (2021). Human-centred software development approaches. In 2021 3rd International Conference on Advances in Computing, Communication Control and Networking (ICAC3N), 602-605. https://doi.org/10.1109/icac3n53548.2021.9725715

[119] Siavvas, M., Kehagias, D., Tzovaras, D., & Gelenbe, E. (2021). A hierarchical model for quantifying software security based on static analysis alerts and software metrics. Software Quality Journal. https://doi.org/10.1007/s11219-021-09555-0

[120] Silva, S., Tuyishime, A., Santilli, T., Pelliccione, P., & Iovino, L. (2023). Quality metrics in software architecture. 2023 IEEE 20th International Conference on Software Architecture (ICSA), L'Aquila, Italy, 2023, 58-69. https://doi.org/10.1109/icsa56044.2023.00014

[121] Simonsson, J., Magnusson, M., & Johanson, A. (2020). Organizing the development of digital product-service platforms. Technology Innovation Management Review, 10(3), 37–48. https://doi.org/10.22215/timreview/1335

[122] Soui, M., & Haddad, Z. (2023). Deep learning-based model using DenseNet201 for mobile user interface evaluation. International Journal of Human-computer Interaction, 1–14. https://doi.org/10.1080/10447318.2023.2175494

[123] Stanley-Marbell, P., Alaghi, A., Carbin, M., Darulova, E., Dolecek, L., Gerstlauer, A., Gillani, G., Jevdjic, D., Moreau, T., Cacciotti, M., Daglis, A., Jerger, N. E., Falsafi, B., Misailovic, S., Sampson, A., & Zufferey, D. (2021). Exploiting errors for efficiency. ACM Computing Surveys, 53(3), 1–39. https://doi.org/10.1145/3394898

[124] Sun, N., Li, C., Chan, H., Le, B. D., Islam, M. Z., Zhang, L. Y., Islam, M. R., & Armstrong, W. (2022). Defining security requirements with the common criteria: Applications, adoptions, and challenges. IEEE Access, 10, 44756–44777. https://doi.org/10.1109/access.2022.3168716

[125] Suonsyrja, S., Sievi-Korte, O., Systa, K., Kilamo, T., & Mikkonen, T. (2018). Objectives and challenges of the utilization of user-interaction data in software development. 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA). https://doi.org/10.1109/seaa.2018.00065

[126] Syed, R., Suriadi, S., Adams, M., Bandara, W., Leemans, S. J., Ouyang, C., ter Hofstede, A. H., van de Weerd, I., Wynn, M. T., & Reijers, H. A. (2020). Robotic process automation: Contemporary themes and challenges. Computers in Industry, 115, Article 103162. https://doi.org/10.1016/j.compind.2019.103162

[127] Tahaei, M., Vaniea, K., Beznosov, K., & Wolters, M. K. (2021). Security notifications in static analysis tools: Developers' attitudes, comprehension, and ability to act on them. Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems (CHI '21). Association for Computing Machinery, New York, NY, USA, Article 691, 1–17. https://doi.org/10.1145/3411764.3445616

[128] Tao, C., Gao, J., & Wang, T. (2019). Testing and quality validation for AI software– Perspectives, issues, and practices. IEEE Access, 7, 120164–120175. https://doi.org/10.1109/access.2019.2937107

[129] Taylor, R. N. (2019). Software architecture and design. In Springer eBooks (pp. 93–122). https://doi.org/10.1007/978-3-030-00262-6_3

[130] Thirumoorthy, K., & Britto, J. J. J. (2022). A clustering approach for software defect prediction using hybrid social mimic optimization algorithm. Computing, 104, 2605–2633. https://doi.org/10.1007/s00607-022-01100-6

[131] Tian, F., Liang, P., & Babar, M. A. (2022). Relationships between software architecture and source code in practice: An exploratory survey and interview. Information and Software Technology, 141, Article 106705. https://doi.org/10.1016/j.infsof.2021.106705

[132] Travis, G. (2019). How the Boeing 737 Max disaster looks to a software developer. IEEE Spectrum. https://spectrum.ieee.org/how-the-boeing-737-max-disaster-looks-to-a- software-developer

[133] Vadan, A.-M., & Miclea, L.-C. (2023). Software testing techniques for improving the quality of smart-home IoT systems. Electronics, 12(6), Article 1337. https://doi.org/10.3390/electronics12061337

[134] Viglianisi, E., Dallago, M., & Ceccato, M. (2020). Resttestgen: Automated black-box testing of RESTful APIs. In 2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST) (pp. 142-152). IEEE. https://doi.org/10.1109/icst46399.2020.00024

[135] Waja, G., Shah, J., & Nanavati, P. (2021). Agile software development. International Journal of Engineering Applied Sciences and Technology, 5(12), 73-78. https://doi.org/10.33564/IJEAST.2021.v05i12.011

[136] Walia, R. (2022). Application of machine learning for GUI test automation. 2022 XXVIII International Conference on Information, Communication and Automation Technologies (ICAT). https://doi.org/10.1109/icat54566.2022.9811187

[137] Wang, J. (2021). Model of open-source software reliability with fault introduction obeying the generalized Pareto distribution. Arabian Journal for Science and Engineering, 46(4), 3981-4000. https://doi.org/10.1007/s13369-021-05382-4

[138] Wang, W., Cheng, J., & Guo, J. L. (2022). How do open source software contributors perceive and address usability?: Valued factors, practices, and challenges. IEEE Software, 39(1), 76–83. https://doi.org/10.1109/ms.2020.3009514

[139] Wieringa, J., Kannan, P. K., Ma, X., Reutterer, T., Risselada, H., & Skiera, B. (2021). Data analytics in a privacy-concerned world. Journal of Business Research, 122, 915-925. https://doi.org/10.1016/j.jbusres.2019.05.005

[140] Wolff, J., & Atallah, N. (2021). Early GDPR penalties: Analysis of implementation and fines through May 2020. Journal of Information Policy, 11, 63–103. https://doi.org/10.5325/jinfopoli.11.2021.0063

[141] Xiao, Y., Ye, Y., & Liu, Y. (2022). Research on the age-appropriate design of mobile phone apps based on the experience of using smartphones for Chinese young-old. In Human Aspects of IT for the Aged Population. Design, Interaction and Technology Acceptance (pp. 248–262). https://doi.org/10.1007/978-3-031-05581-2_19

[142] Xu, F. F., Vasilescu, B., & Neubig, G. (2022). In-IDE code generation from natural language: Promise and challenges. ACM Transactions on Software Engineering and Methodology, 31(2), 1–47. https://doi.org/10.1145/3487569

[143] Yoo, H., & Ahmed, I. (2019). Control logic injection attacks on industrial control systems. In ICT Systems Security and Privacy Protection (pp. 33–48). https://doi.org/10.1007/978-3-030-22312-0_3

[144] Zhang, J., Li, C., Rahaman, M. M., Yao, Y., Ma, P., Zhang, J., Zhao, X., Jiang, T., & Grzegorzek, M. (2021). A comprehensive review of image analysis methods for microorganism counting: From classical image processing to deep learning approaches. Artificial Intelligence Review, 55(4), 2875–2944. https://doi.org/10.1007/s10462-021-10082-4

[145] Zwilling, M., Klien, G., Lesjak, D., Wiechetek, Ł., Cetin, F., & Basim, H. N. (2020). Cyber security awareness, knowledge and behavior: A comparative study. The Journal of Computer Information Systems, 62(1), 82–97. https://doi.org/10.1080/08874417.2020.1712269

[146] Šišejković, D., Merchant, F., Leupers, R., Ascheid, G., & Kegreiss, S. (2019). Control-lock. Proceedings of the 2019 on Great Lakes Symposium on VLSI. https://doi.org/10.1145/3299874.3317983