The Journal of Supercomputing (2024) 80:9579–9608
https://doi.org/10.1007/s11227-023-05818-8

Revolutionizing software developmental processes by utilizing continuous software approaches

Habib Ullah Khan[1] · Waseem Afsar[2] · Shah Nazir[2] · Asra Noor[2,3,7] · Mahwish Kundi[4] · Mashael Maashi[5] · Haya Mesfer Alshahrani[6]

Accepted: 13 November 2023 / Published online: 12 December 2023

Abstract
The development of smart and innovative software applications in various disciplines has inspired our lives by providing various cutting-edge technologies spanning from online to smart and efficient systems. The proliferation of innovative internet-enabled tools has transformed the nation into a globalized world where individuals can participate on various platforms, collaborate in activities, communicate on issues, and exchange information safely and consistently. Coordination and cooperation are essential in software development. It gathers all software developers in one space, encouraging them to discuss goals and work rationally to accomplish the project goal. In recent years, continuous software development and deployment have become increasingly common in software engineering. Continuous software engineering (CSE) is a method that involves a variety of strategies to increase the regularity of novel and modified software versions. CSE enables a continuous learning and improvement process through rapid software update iteration by combining continuous integration and delivery. Continuous integration is a method that has arisen in order to remove gaps between development and deployment. Software engineers must handle uncertainty and alter stakeholders' requirements, which is possible through continuous software developmental strategies that manage the overall software cycle and produce high-quality software applications. The proposed study is a systematic review related to continuous software development and deployment and focuses on achieving four aims: (1) To explore the impacts of continuous development on software, (2) to pinpoint various tools used to carry out this process, (3) to highlight the challenges faced in adopting continuous approaches for development and (4) to analyze the phases of continuous software engineering.

Keywords Software engineering · Continuous software · Automated systems · Software upgradation · Software deployment · Reusable software

Extended author information available on the last page of the article

1 Introduction

The existing software-intensive sector is shifting toward an adaptable and ever-changing business framework based on the creation of value, addressing the demands of a severely competitive and swiftly evolving industrial environment. Developing successful software applications necessitates end-users participation and comprehension of client objectives and behavioral trends. Customer involvement empowers organizations to enhance their products' efficacy by comprehending their customers' needs and requirements [1, 2]. Software engineers often encounter the challenge of managing ambiguity and ever-changing requirements. Agile methodologies provide a solution to combat the challenge of managing ambiguity and ever-changing requirements encountered by software engineers.

Furthermore, the Integration of continuous delivery has enabled the establishment of frequent feedback loops, which have contributed to improving the efficiency of the process. To effectively engage in continuous software engineering (CSE), it is imperative to maintain high code quality through reviews, deploy software frequently, and prioritize user input. Despite these advancements, no software process metamodeling approach currently fully addresses the continuous nature of software engineering [3]. In order to enhance and improve functionality, the perpetual practice of software engineering heavily relies on the acquisition of explicit feedback from users. Incremental feature release can be facilitated more efficiently by consistently monitoring usage, which provides a valuable data source. This is particularly crucial in perpetual software engineering, where explicit feedback from users is vital for improving functionality. Nonetheless, the challenge of effectively correlating observed usage statistics with the changes brought about by a given feature increment, and with that, to a specific feature, proves to be a significant obstacle [4, 5].

Businesses frequently need to improve and adjust their software development procedures to meet security requirements. The complexity of the evaluation protocols presents a challenging environment for professionals to anticipate and assess the amount of effort necessary for compliance assessments [6]. In the field of software quality engineering and security engineering, risks are typically evaluated manually. However, this method is subjective, non-deterministic, error-prone, and time-consuming. As a result, many risks are not explicitly evaluated, which prevents the potential benefits of risk assessment from being fully realized. Nonetheless, in current data-intensive contexts, such as the open Internet environment, continuous project management, or the Internet of Things, data is continuously generated in online, system, or development environments. This data can be leveraged to automatically assess and mitigate software and security threats [7, 8]. For over a decade, the industry has predominantly utilized ongoing procedures emphasizing automated software development processes. Although this method is commonly used, software development is fundamentally a process focused on people, highly collaborative, and requires creativity. Although automated processes have been widely used in the software development cycle, it is important to note that software development is a highly collaborative process that requires creativity and human intervention [9–11]. Software quality presents new problems in programmer development on a regular basis, including characteristics of both software development and system usage that have a substantial influence on software systems' performance [12]. Agile software development approaches have formed the foundation for most software projects in today's world in their many forms. Practically every organization uses the technique. Despite their ubiquity, software implementation failure rates have increased [13]. To be competitive, systems that rely heavily on software must constantly develop their methods. We illustrate the transition process from traditional methodologies to continuous software deployment using a conceptual model called the "Stairway to Heaven" [14]. Larger software development companies require efficient methods for assessing product quality and identifying areas for improvement [15].

The days of developing projects with just a compiler are long gone. Software engineering has become significantly more complicated and diverse in recent years, including several stacks, languages, and frameworks [16]. Continuously releasing product or service increments, such as new features and upgrades, to consumers is becoming more common in developing software-intensive goods and services. Instead of relying on up-front business studies, product and service developers must constantly learn what customers desire through direct user feedback and monitoring of usage behavior [10, 17]. Customer feedback is crucial for software developers as it provides valuable insights into product effectiveness and user satisfaction [18]. This iterative approach helps identify and fix issues, enhances functionality, and improves user experience. This user-centered approach ensures software development aligns with the target market, fostering user loyalty.

2 Methodology

A comprehensive literature study is carried out to analyze the contributions in the domain of Continuous software. A systematic literature review adopts a thorough and rigorous strategy for identifying and evaluating papers relevant to a certain matter of importance [19]. When doing a thorough analysis of the available literature, a variety of criteria that have been put forward by researchers [20, 21] should be followed. The present research implements the aforementioned concepts. The initial stage of the systematic review involves the identification of the criteria necessary to conduct the research. After this requirements assessment, the formulation of research questions has been assumed. The keywords for these research questions were developed with the research questions, so that the search procedure for identifying relevant articles would be more productive. Using logical "AND" and "OR" operators, these terms are then linked to generate queries to solve each research problem.

3 Research protocol

For carrying out this systematic analysis, a research protocol is established, which is responsible for selecting appropriate studies relevant to the topic of interest. Figure 1 shows the graphical representation of this review protocol. The detailed procedure of protocol is given in subheadings.

3.1 Research domain selection

The domain selection process holds significant importance when embarking upon implementing a systematic review. The purpose of the research domain is

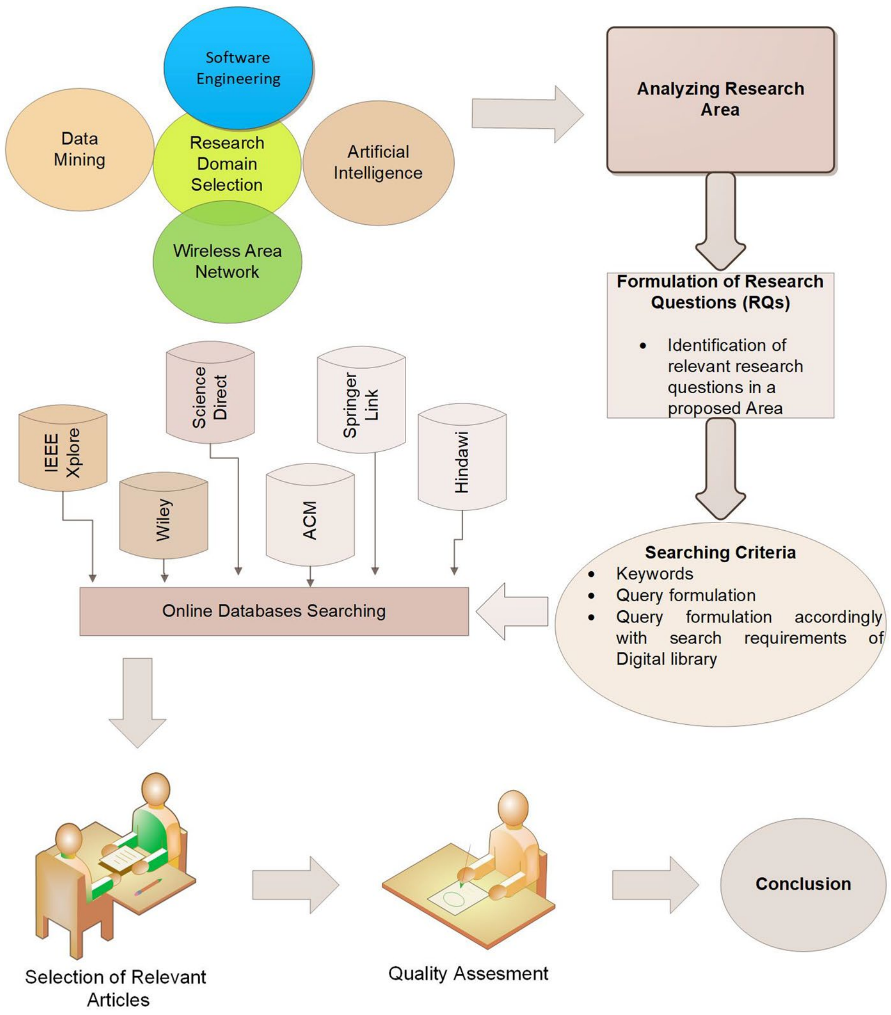
Software Engineering
Data Mining
Research Domain Selection
Artificial Intelligence
Wireless Area Network
Analyzing Research Area
Formulation of Research Questions (RQs)
• Identification of relevant research questions in a proposed Area
Science Direct
Springer Link
IEEE Xplore
Hindawi
Wiley
ACM
Searching Criteria
• Keywords
• Query formulation
• Query formulation accordingly with search requirements of Digital library
Online Databases Searching
Conclusion
Selection of Relevant Articles
Quality Assesment

Fig. 1 Selection of appropriate studies from libraries

established upon a thorough analysis of previous studies that have been scrutinized to facilitate the research activity. Diverse domains are thoroughly assessed, and subsequently, one that is deemed appropriate is chosen for further evaluation. Researchers are responsible for selecting a domain of their preference from an extensive array of domains, such as software engineering, networking, and architecting, to conduct their assessment.

3.2 Analyzing research area

The first stage in conducting research study is choosing a specific topic. Selecting a research topic must be one's personal or professional interests. According to experienced researchers, "a topic in which you are just genuinely interested in the start is likely to become a topic in which you have no interest and with which you will fail to create your best work [22]." Ideally, your study topic should be related to your professional and personal path and can contribute to achieving your professional goals. An in-depth assessment is conducted to find an appropriate research area to carry out research in that specific area to save effort and time. This step is mandatory to identify what researchers have done so far in that specific area.

3.3 Formulating research question

Studies from the previous ten years were reviewed to discover the problems related to continuous software development. Then research questions were formulated to address those domains and enable authors to accomplish the study's key objectives [23]. Table 1 presents the proposed research questions related to the continuous software engineering domain.

3.4 Searching criteria

Following creating the research questions, the authors observed appropriate keywords for locating relevant resources for the proposed study. The following criteria are adopted for generating the search string. Search terms are retrieved from (i) main terms of formulated research questions, (ii) synonyms of main terms are pointed out, (iii) books, chapters, and publications were reviewed for making appropriate keywords, (iv) synonyms were linked by Boolean OR, (v) other terms (main terms) are concatenated with Boolean AND.

A query has been designed since a single term is inefficient for getting the best results. The keywords are merged using the logical operators "AND" and "OR" to get all relevant papers. The search query is as follows:

("Continuous development" OR "continuous integration" OR "Continuous deployment") AND ("software engineering" OR "Software development) AND ("tools" OR "Methods" OR "Approaches") AND ("Impacts" OR "applications" OR "advantages") AND ("Challenges" OR "Problems") AND ("Phases" OR "stages").

Searching with a single keyword may skip a relevant research article. To avoid this search query, formulation is necessary to get all relevant studies.

Table 1 Formulated research questions

RQS	Description
What are the impacts of continuous integration in software development practices?	Continuous software development has enhanced the developmental process of software. The impact of continuous integration on software practices will be analyzed and described in detail
What are the different tools used for enhancing the software developmental process by utilizing continuous integration?	Software engineers use various tools to accomplish various software-related tasks. These tools will be analyzed in various studies
Discuss various challenges faced in adopting continuous integration for software development	Software development is an intricate process involving the execution of design, development, and testing in multiple stages and by collaborative teams. The process may encounter numerous challenges that merit attention
Which phases of software engineering are completed via continuous integration?	In software engineering, the next step is establishing strong connections between software engineering activities. These objectives and connection are used to accelerate and increase the efficiency of the continuous integration process in software engineering

Certain libraries that do not accept our formulated query syntax are manually searched with formulated keywords.

3.5 Online databases searching

To ensure consistency, a comprehensive assessment of the proposed study was conducted on six electronic repositories with the aim of collecting information from various scholarly contributions pertaining to the domain of continuous software engineering. The electronic repositories comprised of Science Direct, Hindawi, ACM, Wiley, Springer, and IEEE Xplore. The title, keywords, and abstracts of conference papers, journal articles, and book chapters were evaluated in order to execute the proposed inquiry. For engineering and computing research, selecting digital libraries like Science Direct, Hindawi, ACM, Wiley, Springer, and IEEE Xplore is a strategic option driven by a variety of crucial considerations. Specifically geared toward computing and engineering subjects, these libraries are well known for their wide-ranging range of scientific and technical literature. Because these platforms organize a wide variety of conference papers, journal articles, and book chapters pertinent to their field of study, researchers favor them. Additionally, the articles in these libraries are frequently subject to strict criteria of trustworthiness and quality, guaranteeing that scholars can rely on the accuracy of the data they access. Table 2 represents the number of articles in each library relevant to continuous software engineering. These studies are retrieved using search strings or manually by using keywords.

Table 2 Articles in each library

Libraries	Total papers
IEEE Xplorer	54
Wiley	8
ACM	185
Hindawi	8
Springer Link	265
Science Direct	103

3.6 Selection of relevant article

For relevance, the entire texts of the selected articles were reviewed. Papers that do not meet specific criteria were excluded:

1. Research not written in the English language
2. Papers that did not include the phrases “Continuous” and “software engineering” or similar phrases in the title, introduction, or complete text and yet did not address the problems related to continuous software development.
3. Articles published before 2012.
4. Articles that highlight the continuous development but are not relevant to computer or software domain.

A total of 130 articles were retrieved after abstract and title-based selection. These articles are then thoroughly examined and screened to include the best and appropriate research studies and eliminate those that do not address the research question. All articles are reviewed library-wise and filtered out to answer any of the research questions. All libraries proceeded through the same screening procedure.

3.7 Quality of assessment

The assessment quality of selected studies was determined through a process of scoring or evaluation to determine their capacity to address predefined research inquiries. Corresponding concerns are addressed in Table 4. Each inquiry has two possible answers, namely 1 and 0. The overall score for each study is the sum of the answers to the respective questions. To establish the suggested study’s validity, appropriate publications with a quality score of two or three were analyzed. These publications are capable of addressing at least two research inquiries. Based on the assessment quality, 66 articles were identified as being able to address at least two specified questions. Table 3 represents the scoring allotted to each article based on the answering research questions.

The graphical figure of the quality of the assessment is presented. Appropriate studies according to the research aims are separated from other studies (Fig. 2).

Table 3 Quality assessment of selected articles

Q1	Q2	Q3	Q4	Score	References
1	1	0	0	2	[24]
1	1	0	0	2	[25]
1	1	1	1	4	[26]
1	1	1	0	3	[27]
1	0	1	1	3	[28]
0	1	1	1	3	[29]
1	1	0	1	3	[30]
1	1	0	0	2	[31]
1	0	0	1	2	[32]
1	0	0	1	2	[33]
0	1	1	0	2	[34]
1	0	0	1	2	[35]
1	1	1	0	3	[9]
1	0	0	1	2	[36]
1	1	0	1	3	[37]
1	1	0	1	3	[38]
1	1	0	0	2	[39]
1	1	1	0	3	[40]
1	1	1	1	4	[41]
1	0	1	1	3	[42]
1	0	0	1	2	[43]
1	0	1	1	3	[44]
1	1	0	0	2	[45]
1	0	0	1	2	[46]
1	0	0	1	2	[47]
1	0	0	1	2	[48]
0	1	0	1	2	[49]
1	0	1	1	3	[50]
0	1	0	1	2	[51]
1	0	1	1	3	[29]
1	1	1	0	3	[52]
1	0	0	1	2	[53]
1	0	1	1	3	[54]
0	1	1	0	2	[55]
0	1	1	1	3	[56]
1	1	0	1	3	[57]
1	1	0	0	2	[58]
1	0	0	1	2	[59]
1	1	0	1	3	[60]
1	0	1	1	3	[61]
1	0	0	1	2	[10]
1	1	1	0	3	[9]
1	1	0	1	3	[38]

Table 3 (continued)

Q1	Q2	Q3	Q4	Score	References
0	0	1	1	2	[62]
1	0	1	0	2	[63]
1	0	1	1	3	[64]
0	0	1	1	2	[65]
1	0	1	1	3	[66]
0	0	1	1	2	[67]
1	1	0	1	3	[68]
1	1	0	1	2	[69]
1	0	0	1	2	[70]
0	1	1	0	2	[71]
0	0	1	1	2	[72]
0	0	1	1	2	[73]
1	1	1	0	3	[74]
1	0	0	1	2	[75]
1	0	0	1	2	[4]
1	0	0	1	2	[14]
0	1	0	1	2	[39]
1	0	0	1	2	[76]
0	1	1	0	2	[77]
1	1	0	0	2	[78]
1	1	0	1	3	[79]
1	0	0	1	2	[80]
1	0	0	1	2	[81]

4 Results and discussion

The proposed research was carried out in order to address the specified research questions. The subsections reveal the intended research's results and discussions.

RQ1. What are the impacts of continuous integration in software development practices?

Continuous integration (CI) significantly improves software development by promoting regular code merges into a shared repository, enabling early error identification and resolution. It speeds up the development process, improves code quality through automated testing and review, and fosters collaboration between developers through simple tasks and transparent reviews. CI also provides fast feedback, builds confidence, simplifies deployment, and unifies security controls, leading to improved efficiency, collaboration, and reliability, ultimately enabling faster delivery of high-quality software. According to researchers and industry practitioners, performance testing must be included in agile development processes in a timely and efficient manner. On the other hand, existing approaches are fragmented and unintegrated, failing to account for the diverse capabilities of users producing polyglot distributed software and their requirement to automate

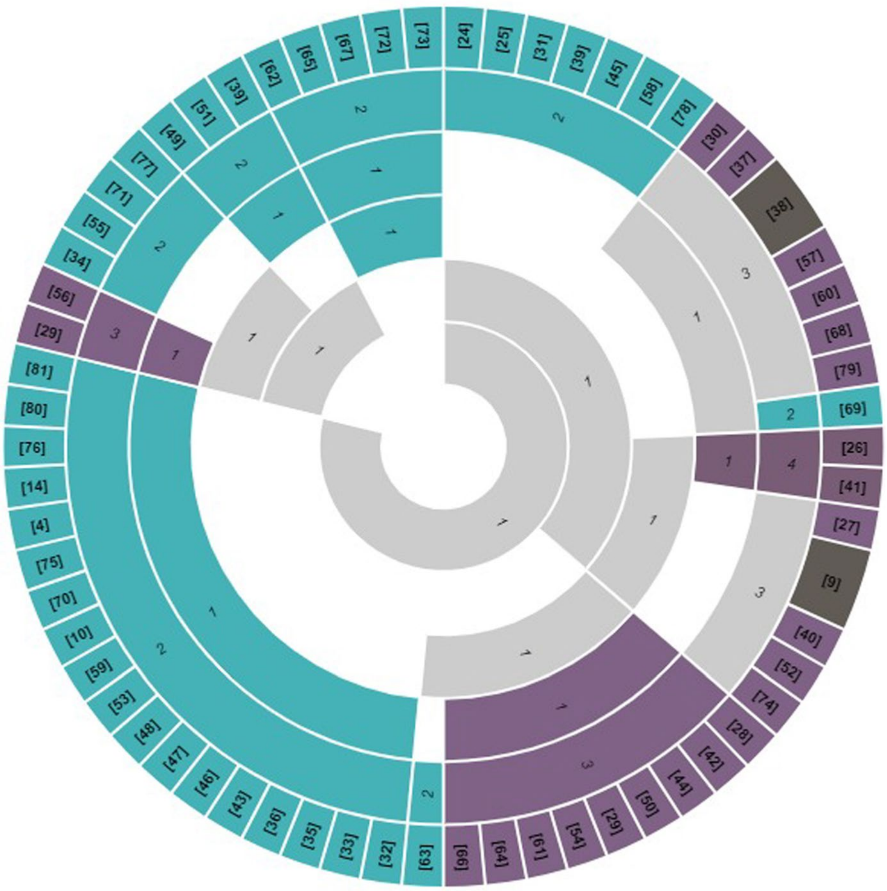

Fig. 2 The quality of the assessment figure

performance practices as they are integrated throughout the lifecycle without slowing it down [52]. Table 4 represents the impact of continuous software practice.

RQ2. What are the different tools used for enhancing the software developmental process by utilizing continuous integration?

Continuous integration (CI) tools are crucial for enhancing software development processes by automating code integration, testing, and deployment tasks. Tools like Jenkins, CircleCI, GitLab, TeamCity, Bamboo, CruiseControl, and Buddy offer CI/CD capabilities, scalability, and compatibility with various technologies. These tools can enhance productivity and efficiency, potentially allowing for personalized implementation of tools and processes for software engineers. These tools can include e-learning platforms, online documentation, development environments, and collaboration platforms. Customization of individual development processes, techniques, and tools, considering their unique characteristics, has been shown to

Table 4 Impacts of continuous software development

Impacts	Description	Citations
Rapid development	Due to highly competitive market prospects, software development enterprises face intense pressure to rapidly expand product-intensive systems and launch valuable software in short timeframes	[24]
Software productivity	In terms of periodic effort and productivity, non-continuous software development has more obstacles. Furthermore, teams using GitHub Tracker scored higher than other teams, indicating that GitHub Tracker has a favorable influence on software development	[25]
Planning	Demonstrating several events at once, yet each event would have had an impact at the time it occurred in continuous software release planning	[26]
Software development	Efficient software development processes are vital for any firm, as they involve many personnel in diverse positions who must communicate and collaborate seamlessly within and beyond the organization. This is especially important for major development endeavors requiring seamless communication and collaboration	[10, 27, 82]
High-quality software development	Advanced agile approaches like continuous integration and test-driven development now cover a wide variety of software development tasks, making it easier to create high-quality software quickly. On the other hand, the reuse of third-party software components is not one of them	[28]
Project coordination	The quality engineer, management, and developers work together to ensure a project meets desired quality standards and criteria, often tailored to the company's needs. Once a consensus is reached, the process becomes transparent. The quality engineer assesses automated analyses, provides developers with reworking assignments, and initiates refactoring tasks by translating measured outputs. They also record quality requirements fulfillment or non-fulfillment in regular reports	[30]
Visual interaction	Springboards are essential tools for developers to visualize specific areas or periods of interest in a software development project. They offer unique perspectives, such as commit springboards, which display branches and code commits and provide visual interaction for developers to perform daily tasks	[23, 31]
User-centered design (UCD)	Implementing user-centered design (UCD) in agile development has been a widely debated topic since its inception. The seamless collaboration of stakeholders, the intertwining of UCD principles with agile development methodologies, and the emphasis placed on artifact-mediated cooperation among diverse professionals are key elements of this approach	[32]

Table 4 (continued)

Impacts	Description	Citations
Product quality	The product quality has significantly improved, with over 90% of open bugs reduced. Continuous delivery (CD) is used to promptly update the code base upon code commitment, allowing affected teams to address production issues quickly. This reduces waiting periods for bug fixes and priority one production incidents, which were previously extended. The CD pipeline also helps reduce human settings and error-prone methods, resulting in faster and more accurate product releases	[4, 33, 52, 75]
Features selection	The strategic integration of modern software tools, primarily open source, within NearForm's framework fosters an ongoing process of software evolution and delivery. Competitive advantage can be gained by swiftly introducing innovative features, but their efficacy depends on superior and trustworthy deployments	[5, 35]
Automation	The industrial sector has long used automation in software development, but it is crucial to remember that software development is a human-driven, collaborative, and creative activity. While studies have explored its impact on continuous processes, quality, and development velocity, there is a lack of research on developer behavior	[9]
Improved customer satisfaction	The organization has migrated its software programs to CD due to improved trust and collaboration between the software development teams and the department's internal customers. The transition has led to a breakdown in trust, but the benefits of CD have decided to migrate the software programs	[36]
Embedded system products	Despite the prevalent utilization of continuous deployment in online, cloud-based, and web-based services like Bing and Facebook, there is a shortage of empirical data regarding the perceived advantages of implementing and executing continuous deployment in software-intensive embedded system solutions	[37]
Development software rapidly	An organization's capacity to build, release, and learn from software in rapid parallel cycles is known as continuous software engineering. Planning, analysis, design, and programming have all suffered from negative disconnects in software development	[38]
Frequently continuous integration test	Big bang integration tests should be conducted more frequently to avoid costly hardware expenses and long running periods. To ensure cost-effectiveness, each modification should have its own test suite, designed to be concise and easy to understand, preventing unjustifiable costs in the automotive industry	[39, 63, 80]
Automate deployment	This method decreases the number of mistakes caused by manual stages while also speeding up the deployment process. Automated deployment is now known as continuous deployment, and projects that automate their deployments release twice as frequently as those that do not. This faster deployment pace allows for more frequent input from customers	[40, 66, 70]

Table 4 (continued)

Impacts	Description	Citations
Continuous practices	Continuous processes are increasingly important, leading to a surge in scholarly work on methodologies, tools, practices, and challenges. Research shows a trend toward continuous integration, rapid release, continuous delivery environment, and deployment. However, their complexity makes defining them challenging, and their conceptualizations depend on an organization's interpretation and implementation	[9, 41]
Defect management	Continuous processes have gained importance, leading to extensive literature on techniques, tools, practices, and challenges. Five studies support this trend: continuous integration (CI), rapid release, continuous deployment/continuous delivery (CDE/CD), and related practices. Differentiating between these practices is challenging, and their meanings depend on an organization's understanding and usage	[42]
Toward the integration of usage	Continuous software engineering (CSE) helps stakeholders capture and understand use and decision knowledge, contributing to software evolution and product quality. CSE's feedback loops enable user engagement, providing valuable insights into software increment acceptability. This feedback guides developers toward their goals and validates requirements. User-centered design methodologies improve software system usability by utilizing implicit or explicit feedback, thereby enhancing the overall software system's usability	[44]
Continuous testing	Continuous testing involves automated cycles for continuous integration and quality feedback. Acceptance testing determines the software candidate's suitability for publication. Monitoring collects live system data for optimization. DevOps adoption drives business objectives into KPIs, emphasizing the importance of regular monitoring for process success	[45]
Software product quality	Process enhancement methodologies improve software performance by continuously modifying and enhancing processes, focusing on product quality, efficiency, and change reduction, using various methodologies and frameworks	[46, 50]
Time-saving	Software development is ongoing. Continuous integration deployment and development reduces organizational and time consumption	[24]
Effectiveness	To enhance performance in the software sector, it is critical to apply software process improvement (SPI) effectively. Software applications and the product business are experiencing disasters, impeding the software's success and eventually leading to its collapse	[47]
Flexibility	Continuous integration produces adaptable software that can be changed and modified to meet changes in requirements	[14, 29, 48, 79]

Table 4 (continued)

Impacts	Description	Citations
Fast development	An organization faced with shortened development cycles must simultaneously take on the responsibilities of development, deployment, and enhancement in software engineering, where progress is uninterrupted	[83]
Continuous software consistency	CSE ensures transparency and consistency in software development cycle artifacts, including requirements, specifications, design, implementation, and updates. It identifies variations, invariants, and dependencies to estimate potential impacts	[76]
Continuous software improvement	Continuous software improvement (CSI) is a systematic process that involves iteration over processes like Integration, testing, delivery, deployment, usage, and run-time monitoring to ensure high-quality software releases	[52]
CAFE	The present study proposes a system with the capability of Context Awareness Feedback, which comprises a well-defined structure for obtaining feedback from the users and a sophisticated mechanism for assimilating the feedback into the development initiatives of the team	[53]
Load test	Organizations must evaluate application performance for user happiness and commercial success, but load testing has large maintenance and execution overhead, making it challenging to implement in reality	[54]
Shorten development cycle	Companies have reduced development cycles and boosted customer collaboration by implementing agile principles. However, this is insufficient. Continuous planning, construction, operation, deployment, and assessment actions are required to manufacture goods that adequately fulfil consumers' demands, make well-informed decisions, and find commercial prospects	[57]
Continuous reliability testing	The utilization of empirical data is employed to enable the execution of testing based on operational profiles, which is designed to form an integral component of acceptability testing before each new production release	[58, 81]
Continuous human-centered design	Usability and user experience specialists favor human-centered design (HCD). It focuses on the tasks that users must do, usability, and user experience. These factors do not generally play an essential role for software developers. Developers frequently concentrate solely on the technical features of an application	[59]
Continuous integration	Continuous integration (CI) is a methodology that involves the continuous integration and testing of code across multiple levels, including block, module, and system. Two essential quality assurance attributes that have been identified for this approach are its effectiveness and reliability	[63, 64, 84]

Table 4 (continued)

Impacts	Description	Citations
Reduce bottlenecks effect	The influence on downstream activities and the acceleration of software releases showed bottlenecks in control systems that indicated the need for additional procedures to manage and make possible continuous and quick software releases, which has become known as DevOps	[61]
Serviceability	Various services are provided by continuous software engineering to its clients in such a business	[74]
Improvements in customer collaboration	New ways of collaborating and working together were also a hot topic in the customer side talks. The lead users were relieved that providing excellent change requirements was no longer solely on their shoulders. In addition, the constant contact and debate about the requirements and application area is very welcomed	[68]
Maintenance	Continuous software is maintained regularly to achieve organizational goal	[78]

increase efficiency, according to a previous study [83]. The evolution of sophisticated tools and libraries has integrated into intricate toolchains, enabling multiple development stages and smooth migration. Advanced frameworks have been created to support distributed and component-based development, addressing the increasing complexity of software development and reducing time and cost. These frameworks have been designed to streamline the process [85].

Continuous software development is increasingly prevalent in software engineering, leading to the development of new toolchains to manage complexities. Containers are lightweight applications, offering advantages over virtual machines (VMs) in system resource utilization. Continuous integration (CI) is a process initiated automatically, including code creation, compilation, unit testing, and validation. Its high execution frequency ensures a quick feedback loop for engineers [74, 77, 86]. Table 5 lists various tools used for continuous software development. Table 5 shows the various tools used for carrying out continuous software development.

RQ3. Discuss various challenges faced in adopting continuous integration for software development

Continuous integration (CI) in software development offers numerous benefits but also presents challenges. It requires adapting workflows, which may lead to resistance from the development team. CI can expose defects or bugs, requiring rigorous testing and hindering early progress. Setting up a CI environment is a technical challenge, with automated tests requiring significant time and resources. Coordination between teams can be complex due to time zones and communication barriers. Scaling CI for large projects can be challenging, and automated processes can introduce security vulnerabilities. Despite these challenges, CI offers significant benefits for software development, particularly in Lean, agile, and DevOps methodologies [48]. These challenges pertain to the construction and configuration of builds, system architecture, seamless Integration, comprehensive testing, timely release, and complex human and organizational resource management issues [29]. Table 6 represents various challenges faced by adopting continuous software practice to software development.

RQ4. Which phases of software engineering are completed via continuous integration? Software system evolves via many life cycles, and throughout each iteration, there are certain periods of commencement, development, and climax for defining requirements. As a result, this framework distinguishes between the requirements that must be created at the beginning of an iteration, those that must be developed throughout an iteration, and those that must be refactored at the end of each iteration [48]. Even the development and progression procedures of software systems and communication infrastructures necessitate persistent observation, oversight, and refinement [76]. The basic principle of continuous software engineering is a well-designed and well-established engineering process that carefully outlines all developmental initiatives, assignments, and outcomes. Formalized models that are methodically altered from one action to the next contain the outcomes of every activity [76]. Continuous software engineering and DevOps go beyond agile software development and not just focus on incremental or scheduled releases. This approach is being advocated as a means to increase productivity, reduce the risk of failure, enhance visibility, feedback, and quality,

Table 5 Tools used for continuous development

Tools	Description	Citations
Java-script and Node.js	Utilizing a uniform programming language across the entire system makes it possible to expedite the code development process to an exceptional degree	[24]
Travis CI toolset	These tools enable the firm to perform successfully under a service and materials contract, in which clients are first lured by the speedy delivery of a prototype in ten days, and then frequent revisions of new functional software are assessed every five days after that	[9, 24, 40, 43, 55]
GitHub Tracker	GitHub Tracker is an application that uses the major capabilities of GitHub to improve continuous software development	[25, 71]
Jenkins	Jenkins serves as a prevalent platform for continuous integration. To streamline the process of continuous integration, open-source toolsets like Jenkins are readily accessible, thus rendering this methodology within reach for prospective users	[9, 16, 26, 38, 40, 41, 43, 55, 60, 71, 79]
Software product line management	The key characteristics of software product line management systems such as graphical user interface, prototype, online, plug-in, free, open-source, user guide, sample solutions, and application framework are necessary to be practiced by developers while developing continuous software	[27, 87]
Source code manager (SCM)	Source code management (SCM) is a tool for organizing developer cooperation during the development phase of a project. SCM addresses how expert developers may utilize, share, and change the primary source of the project's codes across several locations and workstations	[27]
Systems tackle	Snowflake Servers are utilized because it is challenging for customers to retain and reuse their delivery process specifications in these systems. In the second generation of delivery systems, all delivery process models and associated data are kept in a single, version-controlled file using the infrastructure as a code concept	[29]
ConQAT (Continuous quality assessment toolkit)	The Continuous Quality Assessment Toolkit (ConQAT) is a highly customizable software quality analysis engine based on advanced pipe and filter design research, offering superior metrics and tools like ConQAT, Team Scale, and Sonar	[30]

Table 5 (continued)

Tools	Description	Citations
DecDoc tool	The use of the DecDoc methodology by Lee is aimed at visualizing decision-related knowledge by representing individual design decisions in a hierarchy of decision elements, which includes arguments or alternatives. It is important to mention that this study is centered on design requirements rather than the conventional software artifact decisions that occur throughout continuous software engineering	[31]
Continuous Integration Visualization Technique (CIViT)	CIViT is a visualization method for end-to-end testing procedures, addressing businesses' lack of understanding of testing operations and frequency. It facilitates discussions, identifies testing activities, evaluates measurements, and optimizes testing operations for continuous integration	[34, 88]
Continuous Integration	Continuous integration (CI) is a software engineering practice that necessitates the perpetual integration and testing of code at various levels, such as block, module, and system. Additionally, software quality characteristics that have been specified include performance and stability	[37, 77]
Trace-Based test selection	Trace-Based Test Selection is a novel approach for the selection of tests in distributed embedded systems at the system level. It eliminates the necessity of possessing knowledge of the source code; instead, it necessitates an understanding of the functional architecture concerning the distribution across the control units of the system and the signals exchanged between them	[39]
Bamboo and Hudson	The utilization of Bamboo and Hudson facilitates software deployment to a staging or production environment. In a research publication, Jenkins functioned as a continuous delivery/deployment server. These tools were situated in the locations mentioned earlier. TeamCity, as a pipeline continuous integration server, was highlighted in one study alongside other CI servers	[41]
DevOpRET	DevOpRET is a DevOps technique for reliability testing, acceptance testing, and operational-profile-based testing. It continuously improves its estimation by analyzing usage and failure data. Controlled experiments show that DevOpRET provides reliable and efficient reliability estimations	[45, 58]
JIRA	A functional need is documented in a JIRA backlog as a user story with acceptance criteria and an estimated effort	[49, 68]

Table 5 (continued)

Tools	Description	Citations
NetBeans	NetBeans is an IDE that allows applications to be developed from a set of modular software components called modules	[83]
BizDev	BizDevOps, also called DevOps, represents a software development methodology that places great emphasis on fostering collaboration among developers, operations personnel, and business teams	[51, 57]
BenchFlow	BenchFlow allows continuous development lifecycle operations to include performance testing and analytic procedures	[52]
Eclipse	Eclipse serves as an essential tool in the realm of continuous software advancements	[56]
Docker	Both component-based architectures and micro-service architectures utilize Docker, yet practitioners often view containers as a simple resource-saving technique compared to virtual machines	[69, 74, 78]

Table 6 Challenges in adopting continuous integration

Challenges	Description	Citations
Risk of overrun	The release plan may not be completed on time owing to unanticipated events, which might have financial consequences. In such cases, this risk may arise. Such a risk can be categorized as either a high-priority risk, where the release plan has already passed the publishing date owing to changes, or a risk indicator, where there is a possibility for delays based on previous occurrences	[26]
Continuous practices	Continuous practices like upgradation and maintenance pose various challenges for programmers and testers of continuous software engineering	[27, 63]
Reuse	Experts must thoroughly evaluate potential reuse components, considering both functional and non-functional requirements to improve the reliability and efficiency of the selection process, requiring thorough attention to detail	[28, 73]
Organizational challenges	The projects face organizational challenges due to divisions within the firm with diverging objectives, leading to team tension. To resolve root access to servers involved extensive evaluation. The leadership team initiated a restructuring initiative to foster collaboration	[33]
Project’s software architecture	The software delivery system is subjected to additional functional requirements due to the architectural decision. The software delivery system may need the integration of new components and systems	[29]
Delivery process	The delivery process may change; for example, a new test stage may need to be added. Finally, new non-functional needs or rules may develop from an organizational standpoint, affecting not just the delivery process but also the delivery system itself, and the project may be required to comply with new laws and regulations. Overall, the evolution challenge necessitates adaptable and maintainable delivery methods	[29, 62, 65, 77]
Continuous processes	Industrial practitioners use fundamental methodologies for continuous advancement, including software system creation, quality assurance integration, automated testing, and operational protocol modification beyond software organization, facilitating computer science engineering implementation	[32, 67]
Continuous integration	Implementing continuous integration in large software development firms presents challenges due to various factors, such as organizational, social, and technological aspects. One of the technological challenges faced in implementing continuous integration in large software development firms is prioritizing test cases that can be executed quickly and generate the greatest number of failures on time	[18, 34, 64]
Transition period	Kubernetes automation led to process modifications, impacting developer behavior and team conduct, causing a challenging transition period. This issue may not be widespread across other institutions	[9]

Table 6 (continued)

Challenges	Description	Citations
Proper testing	As noted, adequate testing procedures and low-test quality Partially following this strategy also risks test efficiency, flaky tests, insufficient test coverage, and long-running tests	[40]
Lack of investment	Due to various factors, continuous practice adoption in customer and software development organizations is linked to significant costs and investments	[11, 41]
Defect management	Software engineering tasks can lead to quality issues or bugs, such as mistakes, malfunctions, and failures. These can result in financial loss, reputational damage, and decreased client satisfaction, especially harmful defects	[42, 89]
Threats to validity	The practitioners' diverse perspectives on CSE may lead to minimal conformity, but observations and interviews with coworkers were conducted to address this heterogeneity.	[44, 55, 63, 71, 74]
Performance test	A declarative domain-specific language (DSL) for software performance testing should be developed to address the lack of a clear statement of performance assessment aims and context in current methodologies	[50, 54]
Lifecycle integration	Define acceptable methods for specifying which performance analysis queries to include in which activities of the continuous software improvement lifecycle, while minimizing the impact on the lifecycle's "continuity."	[52]
Pipelines development	Software engineering involves DevOps, microservice architectures, continuous integration, and delivery. Self-contained services are developed and delivered through automated pipelines, reducing load testing time and resources	[54]
Detecting architectural problems	The solution offers massive system insight without developers needing prior knowledge or prerequisites. Tested on open-source projects, it has identified flaws that could have been identified earlier	[56]
Time-based iterations	Release planning challenges time-based iterations, requiring flexible, responsive practices prioritizing deliverables over schedules. This approach initiates initial CP projects, promoting continuous flow and responsiveness to business requirements	[61]
Software configuration management process	The application development life-cycle may encounter obstacles due to implementing methodologies such as continuous delivery and continuous deployment. Several of these challenges are related to the software configuration management procedure and effectively communicating software modifications to significant stakeholders such as operational teams	[72]

Table 6 (continued)

Challenges	Description	Citations
Testability	Testability is defined simply as the degree of ease or difficulty in testing a system or software artifact. Such artifacts may encompass software modules, unified modeling language models, requirements documents, and applications	[9]
Technical challenges	The main barrier to advancement is the inadequate assistance provided by tools, which are largely technical solutions and have matured. Practical solutions are close but not viable alternatives. Restrictions on daily system cycles and deployment-induced downtime hinder progress	[66]

Table 7 Continuous software phases

Software phases	Description	Citations
Continuous delivery	Continuous delivery is a collection of techniques to provide value to consumers quickly, consistently, and with little manual effort. For a short time to market, you must deliver as quickly as feasible. Customers appreciate prompt delivery. This frequently translates to better flexibility in software development	[16, 29, 36, 43, 56, 65, 68, 70, 72, 75, 79, 81, 48, 64]
Planning	Continuous efforts involve planning, protection, and constant usage. BizDev, similar to DevOps, synchronizes organizational strategy and software engineering. It strengthens the link between the development, planning, integration, and testing phases. Continuous software release planning (SRP) allows adjustments based on daily development events	[26, 40, 67, 57, 61]
Reuse	Test cases produced through a conventional test-driven, agile development methodology serve as queries for test-driven search engines, which define the required functionality. The objective is to identify potential candidates that can be reused promptly and continuously for software component reuse	[28]
Quality improvement	Consistent quality control is essential for project preservation and maintenance. DevOps techniques like canary releasing and A/B testing aim to improve software quality while maintaining a quick development pace. This methodology monitors the system state and aligns with quality objectives	[30, 62]
Testing	Test automation and live site monitoring are important metrics in continuous software development (CSE). Unit test automation should include integration and acceptability tests. However, the cost of automated testing in mobile applications is often overlooked for development speed. Customer acceptance tests are executed manually, resulting in a lack of progress. Performance testing methods mainly rely on scripting languages and frameworks	[32] [38, 54, 73, 50, 54]
Continuous deployment	Continuous deployment provides a sense of closeness to the product development team but may make cross-functional teams struggle to plan due to regular feedback. Continuous software engineering integrates development, deployment, and operating phases, reducing the development and production intervals. This approach allows for continuous extension and modification of a system's functionality, leading to efficient management of long-lasting systems	[37, 60, 66, 70, 80, 10, 14, 44, 51, 69]

Table 7 (continued)

Software phases	Description	Citations
Architecting	The process architecture uses four technologies: Java-script and Node.js for rapid code development, distributed microservices architecture for small code lines, continuous deployment with Docker, and quality features like GitHub code commit hooks and Travis continuous integration system for efficient system management. The architecting phase is crucial for successfully implementing procedures, with ability being a key quality standard for continuous delivery and deployment	[41, 35]
Defect management	Defect detection, defect analysis, defect prevention, defect resolution, defect monitoring, and defect process improvement are the six processes in this process. 1) Detection of flaws, 2) analysis of defects, 3) prevention of defects, 4) error correction, 5) monitoring and reporting of defects, 6) improving the defect process	[42, 90, 91]
Requirement analysis	Users and organization requirements are acquired during requirement analysis, which is a key phase in continuous software development	[46]
Decision support systems	Decision support systems (DSSs), such as lightweight blockchain systems, play an important role in system selection	[47]
Requirements specification	A requirement specification is a list of all the requirements established throughout the product’s creation and testing	[49]
Continuous evolution	A well-defined system, adequate infrastructure, procedures, and communication mechanisms are essential to effectively engage users. Supervisory endorsement and regular consideration of user feedback are also essential in the ongoing software development process	[53, 76]
Continuous validation	Continuous validation is the process of automatically integrating freshly created code into the primary database	[56]
Continuous run-time monitoring	For continuous innovation to be deemed viable, constant run-time monitoring is required. It must respond to changing market conditions and guide the company’s strategy	[59]
Designing	Software design comprises a methodical approach to converting user needs into a suitable form and making it easier for developers to create and deploy software	[4]
Requirements engineering (RE)	Requirements engineering is one of the most important processes in software development since it aims to maximize the value of a software release while allowing for a collaborative approach with diverse stakeholder views throughout the product development process	[1]

and provide value more expeditiously by employing continuous delivery and deployment methodologies. Rather than focusing solely on incremental or scheduled releases, continuous software engineering, and DevOps extend beyond agile software development. This approach has been supported to increase productivity, reduce the risk of failure, enhance visibility, feedback, and quality, and provide value more expeditiously by employing continuous delivery and deployment methodologies [68, 90]. Table 7 discusses all the phases covered by the continuous software life cycle.

5 Conclusion

With the rapid increase in technology, the software engineering discipline is also facing undeniable change. In comparison with other scientific sectors, much research is conducted within corporations to create new potential software solutions to enhance the business operation of organizations. The software industry is increasingly confronted with the necessity for rapid and continuous system deployment. Continuous software development strives to enhance application development by digitizing the entire software development process. CSE utilizes iterative developmental approaches to compete with a rapidly changing market. The creation of software-intensive products and services is increasingly characterized by the continuous release of product or service increments, such as new features and upgrades, to users. The four objectives of the provided SLR—the influence of continuous software engineering, the tools used to improve software processes, the difficulties organizations experience in implementing CSE and identifying many phases of software engineering encompassed by continuous conception are all highlighted. The research study will support scholars, innovators, researchers, and institutions working in the field of software engineering to create and employ optimal resources for improving software development procedures.

Acknowledgements We are thankful to all the authors of this manuscript. *These authors contributed equally and are considered as first co-authors.

Author contributions H.U.K and W.A are concern with the methodology, A.N and M.K are concern with Analysis and writing, S.N is concern with validation, M.M and H.MA are concern with data extraction, data validation, presentation and grammar checking.

Funding Princess Nourah bint Abdulrahman University Researchers Supporting Project number (PNURSP2023R237), Princess Nourah bint Abdulrahman University, Riyadh, Saudi Arabia. Research Supporting Project number (RSPD2023R787), King Saud University, Riyadh, Saudi Arabia.

Declarations

Conflict of interest The authors declare that there is no conflict of interest.

Ethics approval and consent to participate Not applicable.

Consent for publication Not applicable.

References

1. Yaman SG, Sauvola T, Riungu-Kalliosaari L, Hokkanen L, Kuvaja P, Oivo M et al Customer involvement in continuous deployment: a systematic literature review. In: International working conference on requirements engineering: foundation for software quality, pp 249–265
2. Hu F, Xi X, Zhang Y (2021) Influencing mechanism of reverse knowledge spillover on investment enterprises' technological progress: an empirical examination of Chinese firms. Technol Forecast Soc Chang 169:120797
3. Krusche S, Bruegge B (2017) CSEPM-a continuous software engineering process metamodel .In: 2017 IEEE/ACM 3rd international workshop on rapid continuous software engineering (RCoSE), pp 2–8
4. Johanssen JO, Kleebaum A, Bruegge B, Paech B Feature crumbs: adapting usage monitoring to continuous software engineering. In: International conference on product-focused software process improvement, pp 263–271
5. Zhou X, Zhang L (2022) SA-FPN: an effective feature pyramid network for crowded human detection. Appl Intell 52:12556–12568
6. Moyón F, Bayr C, Mendez D, Dännart S, Beckers K (2020) A light-weight tool for the self-assessment of security compliance in software development–an industry case. In: International conference on current trends in theory and practice of informatics, pp 403–416
7. Felderer M (2018) Risk-based software quality and security engineering in data-intensive environments. In: International conference on future data and security engineering, pp 12–17
8. Khan HU, Hussain A, Nazir S, Ali F, Khan MZ, Ullah I (2023) A service-efficient proxy mobile IPv6 extension for IoT domain. Information 14:459
9. Elazhary O, Storey M-A, Ernst NA, Paradis E (2021) Adept: a socio-technical theory of continuous integration. In: 2021 IEEE/ACM 43rd international conference on software engineering: new ideas and emerging results (ICSE-NIER), pp 26–30
10. Fagerholm F, Guinea AS, Mäenpää H, Münch J (2017) Building blocks for continuous experimentation. In: Proceedings of the 1st international workshop on rapid continuous software engineering, pp 26–35
11. Li T, Fan Y, Li Y, Tarkoma S, Hui P (2021) Understanding the long-term evolution of mobile app usage. IEEE Trans Mob Comput. https://doi.org/10.1109/TMC.2021.3098664
12. Franch X, Lopez L, Martínez-Fernández S, Oriol M, Rodríguez P, Trendowicz A (2019) Quality-aware rapid software development project: the Q-rapids project. In: International conference on objects, components, models and patterns, pp 378–392
13. Fogarty A, Edgeworth A, Smith O, Dowling M, Yilmaz M, MacMahon ST et al (2020) Agile software development–do we really calculate the costs? A multivocal literature review. In: European Conference on Software Process Improvement, pp 203–219
14. Olsson HH, Bosch J (2014) Climbing the "Stairway to Heaven": evolving from agile development to continuous deployment of software. In: Continuous software engineering, ed: Springer, pp 15–27
15. Antinyan V, Staron M, Meding W (2014) Profiling prerelease software product and organizational performance. In: Continuous software engineering, ed: Springer, pp 167–182
16. Garcia J, Cabot J (2018) Stepwise adoption of continuous delivery in model-driven engineering. In: International workshop on software engineering aspects of continuous development and new paradigms of software production and deployment, pp 19–32
17. Li T, Zhang M, Cao H, Li Y, Tarkoma S, Hui P (2020) what apps did you use?: Understanding the long-term evolution of mobile app usage. In: Proceedings of the web conference 2020, pp 66–76
18. Xi X, Xi B, Miao C, Yu R, Xie J, Xiang R et al (2022) Factors influencing technological innovation efficiency in the Chinese video game industry: Applying the meta-frontier approach. Technol Forecast Soc Change 178:121574
19. Kitchenham B (2004) Procedures for performing systematic reviews. Keele, UK, Keele University 33:1–26
20. Keele S. Guidelines for performing systematic literature reviews in software engineering. Technical report, Ver. 2.3 EBSE Technical Report. EBSE2007
21. Kitchenham B, Charters S (2007) Guidelines for performing systematic literature reviews in software engineering
22. Saunders M (2014) Research Methods for Business Students, 6th edn.

23. Lu S, Liu M, Yin L, Yin Z, Liu X, Zheng W (2023) The multi-modal fusion in visual question answering: a review of attention mechanisms. PeerJ Comput Sci 9:e1400
24. Clarke PM, Elger P, O'Connor RV (2016) Technology enabled continuous software development. In: Proceedings of the international workshop on continuous software evolution and delivery, pp 48–48
25. Silvestre L, Vera JM (2019) Improving continuous software development in academic scenarios using GitHubTracker. In: 2019 38th international conference of the chilean computer science society (SCCC), pp 1–8
26. Ameller D, Farré C, Franch X, Valerio D, Cassarino A (2017) Towards continuous software release planning. In: 2017 IEEE 24th international conference on software analysis, evolution and reengineering (SANER), pp 402–406
27. Uzunbayir S, Kurtel K (2018) A review of source code management tools for continuous software development. In: 2018 3rd international conference on computer science and engineering (UBMK), pp 414–419
28. Kessel M, Atkinson C (2018) Integrating reuse into the rapid, continuous software engineering cycle through test-driven search. In: 2018 IEEE/ACM 4th international workshop on rapid continuous software engineering (RCoSE), pp 8–11
29. Steffens A, Lichter H, Döring JS (2018) Designing a next-generation continuous software delivery system: Concepts and architecture. In: 2018 IEEE/ACM 4th international workshop on rapid continuous software engineering (RCoSE), pp 1–7
30. Steidl D, Deissenboeck F, Poehlmann M, Heinke R, Uhink-Mergenthaler B (2014) Continuous software quality control in practice. In: 2014 IEEE international conference on software maintenance and evolution, pp 561–564
31. Johanssen JO, Kleebaum A, Bruegge B, Paech B (2017) Towards the visualization of usage and decision knowledge in continuous software engineering. In: 2017 IEEE working conference on software visualization (VISSOFT), pp 104–108
32. Dittrich Y, Nørbjerg J, Tell P, Bendix L (2018) Researching cooperation and communication in continuous software engineering. In: 2018 IEEE/ACM 11th international workshop on cooperative and human aspects of software engineering (CHASE), pp 87–90
33. Chen L (2015) Continuous delivery: huge benefits, but challenges too. IEEE Softw 32(2):50–54
34. Knauss E, Staron M, Meding W, Söder O, Nilsson A, Castell M (2015) Supporting continuous integration by code-churn based test selection. In: 2015 IEEE/ACM 2nd international workshop on rapid continuous software engineering, pp 19–25
35. O'Connor R, Elger P, Clarke PM (2016) Exploring the impact of situational context—a case study of a software development process for a microservices architecture. In: 2016 IEEE/ACM international conference on software and system processes (ICSSP), pp 6–10
36. Chen L (2015) Towards architecting for continuous delivery. In: 2015 12th Working IEEE/IFIP conference on software architecture, pp 131–134
37. Dakkak A, Mattos DI, Bosch J (2021) Perceived benefits of continuous deployment in software-intensive embedded systems. In: 2021 IEEE 45th annual computers, software, and applications conference (COMPSAC), pp 934–941
38. Klepper S, Krusche S, Peters S, Bruegge B, Alperowitz L (2015) Introducing continuous delivery of mobile apps in a corporate environment: a case study. In: 2015 IEEE/ACM 2nd international workshop on rapid continuous software engineering, pp 5–11
39. Vst S, Wagner S (2016) Trace-based test selection to support continuous integration in the automotive industry. In 2016 IEEE. In: ACM international workshop on continuous software evolution and delivery (CSED), pp 34–40
40. Elazhary O, Werner C, Li ZS, Lowlind D, Ernst NA, Storey M-A (2021) Uncovering the benefits and challenges of continuous integration practices. IEEE Trans Softw Eng 48(7):2570–2583
41. Shahin M, Babar MA, Zhu LJIA (2017) Continuous integration, delivery and deployment: a systematic review on approaches, tools, challenges and practices. IEEE access 5:3909–3943
42. Abd Rahman A, Hasim N (2015) Defect management life cycle process for software quality improvement. In: 2015 3rd international conference on artificial intelligence, modelling and simulation (AIMS), pp 241–244
43. Paule C, Düllmann TF, Van Hoorn A (2019) Vulnerabilities in continuous delivery pipelines? a case study. In: 2019 IEEE international conference on software architecture companion (ICSA-C), pp 102–108

44. Johanssen JO, Kleebaum A, Paech B, Bruegge B (2019) Continuous software engineering and its support by usage and decision knowledge: an interview study with practitioners. J Softw Evol Process 31:e2169
45. Bertolino A, Angelis GD, Guerriero A, Miranda B, Pietrantuono R, Russo S et al (2020) DevOpRET: Continuous reliability testing in DevOps. J Softw Evol Process 35(3):e2298
46. Sun L, Nazir S, Hussain AJSP (2021) Multicriteria decision making to continuous software improvement based on quality management, assurance, and metrics. Sci Program. https://doi.org/10.1155/2021/9953618
47. Xiaolong H, Nazir S, Lunchao Z, Jun DJSP (2021) Library-based overview of multicriteria decision making for continuous software improvement for internet of software industry. Sci Program. https://doi.org/10.1155/2021/5519900
48. Theunissen T, Van Heesch U (2017) Specification in continuous software development. In: Proceedings of the 22nd European conference on pattern languages of programs, pp 1–19
49. Van Heesch U, Theunissen T, Zimmermann O, Zdun U (2017) Software specification and documentation in continuous software development: a focus group report. In: Proceedings of the 22nd European conference on pattern languages of programs, pp 1–13
50. Ferme V, Pautasso C (2018) A declarative approach for performance tests execution in continuous software development environments. In: Proceedings of the 2018 ACM/SPEC international conference on performance engineering, pp 261–272
51. Fitzgerald B, Stol K-J (2014) Continuous software engineering and beyond: trends and challenges. In: Proceedings of the 1st international workshop on rapid continuous software engineering, pp 1–9
52. Ferme V, Pautasso C (2017) Towards holistic continuous software performance assessment. In: Proceedings of the 8th ACM/SPEC on international conference on performance engineering companion, pp 159–164
53. Dzvonyar D, Krusche S, Alkadhi R, Bruegge B (2016) Context-aware user feedback in continuous software evolution. In: 2016 IEEE/ACM international workshop on continuous software evolution and delivery (CSED), pp 12–18
54. Schulz H, Angerstein T, van Hoorn A (2018) Towards automating representative load testing in continuous software engineering. In: Companion of the 2018 ACM/SPEC international conference on performance engineering, pp 123–126
55. Zahedi M, Rajapakse RN, Babar MA (2020) Mining questions asked about continuous software engineering: A case study of stack overflow. In: Proceedings of the evaluation and assessment in software engineering, ed, pp 41–50
56. Goldstein M, Segall I (2015) Automatic and continuous software architecture validation. In: 2015 IEEE/ACM 37th IEEE international conference on software engineering, pp 59–68
57. dos Santos Júnior PS, Perini Barcellos M, Borges Ruy F (2021) Tell me: Am I going to Heaven? A Diagnosis Instrument of Continuous Software Engineering Practices Adoption. In: Evaluation and assessment in software engineering, ed, pp 30–39
58. Pietrantuono R, Bertolino A, De Angelis G, Miranda B, Russo S (2019) Towards continuous software reliability testing in DevOps. In: 2019 IEEE/ACM 14th international workshop on automation of software test (AST), pp 21–27
59. Forbrig P (2016) Continuous software engineering with special emphasis on continuous business-process modeling and human-centered design. In: Proceedings of the 8th international conference on subject-oriented business process management, pp 1–4
60. Leppänen M, Kilamo T, Mikkonen T (2015) Towards post-agile development practices through productized development infrastructure. In: 2015 IEEE/ACM 2nd international workshop on rapid continuous software engineering, pp 34–40
61. De França BBN, Simões RV, Silva V, Travassos GH (2017) Escaping from the time box towards continuous planning: an industrial experience. In: 2017 IEEE/ACM 3rd international workshop on rapid continuous software engineering (RCoSE), pp 43–49
62. Düllmann TF, Paule C, van Hoorn A (2018) Exploiting devops practices for dependable and secure continuous delivery pipelines. In: 2018 IEEE/ACM 4th international workshop on rapid continuous software engineering (RCoSE), pp 27–30
63. Laukkanen E, Mäntylä M (2015) Build waiting time in continuous integration--an initial interdisciplinary literature review. In: 2015 IEEE/ACM 2nd international workshop on rapid continuous software engineering, pp 1–4

64. Krusche S, Alperowitz L, Bruegge B, Wagner MO (2014) Rugby: an agile process model based on continuous delivery. In: Proceedings of the 1st international workshop on rapid continuous software engineering, pp 42–50
65. Toh MZ, Sahibuddin S, Mahrin MNR (2019) Adoption issues in DevOps from the perspective of continuous delivery pipeline. In: Proceedings of the 2019 8th international conference on software and computer applications, pp 173–177
66. Virtanen A, Kuusinen K, Leppänen M, Luoto A, Kilamo T, Mikkonen T (2017) On continuous deployment maturity in customer projects. In: Proceedings of the symposium on applied computing, pp 1205–1212
67. Kirikova M (2017) Continuous requirements engineering. In: Proceedings of the 18th international conference on computer systems and technologies, pp 1–10
68. Itkonen J, Udd R, Lassenius C, Lehtonen T (2016) perceived benefits of adopting continuous delivery practices. In: ESEM, pp 42:1–42:6
69. Prens D, Alfonso I, Garcés K, Guerra-Gomez J (2019) Continuous delivery of software on IoT devices. In: 2019 ACM/IEEE 22nd international conference on model driven engineering languages and systems companion (MODELS-C), pp 734–735
70. Mäkinen S, Lehtonen T, Kilamo T, Puonti M, Mikkonen T, Männistö T (2019) Revisiting continuous deployment maturity: a two-year perspective. In: Proceedings of the 34th ACM/SIGAPP symposium on applied computing, pp 1810–1817
71. Huijgens H, Spadini D, Stevens D, Visser N, Van Deursen A (2018) Software analytics in continuous delivery: a case study on success factors. In: Proceedings of the 12th ACM/IEEE international symposium on empirical software engineering and measurement, pp 1–10
72. Cardoso TE, Santos AR, Chanin R, Sales A (2020) Communication of changes in continuous software development. In: International conference on software business, pp 86–101
73. Bosch J (2014) Continuous software engineering: an introduction. In: Continuous software engineering, ed: Springer, pp 3–13
74. Koskinen M, Mikkonen T, Abrahamsson P (2019) Containers in software development: a systematic mapping study. In: International conference on product-focused software process improvement, pp 176–191
75. Siebra C, Lacerda R, Cerqueira I, Quintino JP, Florentin F, Silva F et al (2018) Empowering continuous delivery in software development: the DevOps strategy. In: International conference on software technologies, pp 247–265
76. Sandkuhl K (2017) Aligning software architecture and business strategy with continuous business engineering. In: International conference on advanced information systems engineering, pp 14–26
77. Feilhauer M, Häring J, Buchner J (2016) Continuous delivery for simulation-model development. In: 16. Internationales Stuttgarter Symposium, pp 467–477
78. Theunissen T, van Heesch U, Avgeriou P (2022) A mapping study on documentation in Continuous Software Development. Inf Softw Technol 142:106733
79. Fitzgerald B, Stol K-J (2017) Continuous software engineering: a roadmap and agenda. J Syst Softw 123:176–189
80. Nakagawa EY, Antonino PO, Schnicke F, Kuhn T, Liggesmeyer P (2021) Continuous systems and software engineering for Industry 4.0: a disruptive view. Inf Softw Technol 135:106562
81. Chen L (2017) Continuous delivery: overcoming adoption challenges. J Syst Softw 128:72–86
82. Khan HU, Ali F, Ghadi YY, Nazir S, Ullah I, Mohamed HG (2023) Human–computer interaction and participation in software crowdsourcing. Electronics 12:934
83. Papatheocharous E, Belk M, Nyfjord J, Germanakos P, Samaras G (2014) Personalised continuous software engineering. In: Proceedings of the 1st international workshop on rapid continuous software engineering, pp 57–62
84. Liu C, Wu T, Li Z, Ma T, Huang J (2022) Robust online tensor completion for IoT streaming data recovery. IEEE Trans Neural Netw Learn Syst 10178–10192
85. Wätzoldt S, Neumann S, Benke F, Giese H (2012) Integrated software development for embedded robotic systems. In: International Conference on simulation, modeling, and programming for autonomous robots, pp 335–348
86. Liu X, Shi T, Zhou G, Liu M, Yin Z, Yin L et al (2023) Emotion classification for short texts: an improved multi-label method. Humanit Soc Sci Commun 10:1–9
87. Cheng B, Zhu D, Zhao S, Chen J (2016) Situation-aware IoT service coordination using the event-driven SOA paradigm. IEEE Trans Netw Serv Manage 13:349–361

88. Ni Q, Guo J, Wu W, Wang H, Wu J (2021) Continuous influence-based community partition for social networks. IEEE Trans Netw Sci Eng 9:1187–1197
89. Li B, Zhou X, Ning Z, Guan X, Yiu K-FC (2022) Dynamic event-triggered security control for networked control systems with cyber-attacks: a model predictive control approach. Inf Sci 612:384–398
90. Wu Z, Cao J, Wang Y, Wang Y, Zhang L, Wu J (2018) hPSD: a hybrid PU-learning-based spammer detection model for product reviews. IEEE Trans Cybern 50:1595–1606
91. Ahmad I, Ullah I, Khan WU, Ur Rehman A, Adrees MS, Saleem MQ et al (2021) Efficient algorithms for E-healthcare to solve multiobject fuse detection problem. J Healthc Eng 21:1–16

Authors and Affiliations

Habib Ullah Khan[1] · Waseem Afsar[2] · Shah Nazir[2] · Asra Noor[2,3,7] · Mahwish Kundi[4] · Mashael Maashi[5] · Haya Mesfer Alshahrani[6]

✉ Waseem Afsar
cswaseemafsar@gmail.com

✉ Shah Nazir
shahnazir@uoswabi.edu.pk

✉ Asra Noor
asranoor997@gmail.com

1 Department of Accounting and Information Systems, College of Business and Economics, Qatar University, Doha, Qatar

2 Department of Computer Science, University of Swabi, Swabi, Pakistan

3 Department of Computer Science, Women University, Swabi, Pakistan

4 Abdul Wali Khan University Mardan, Mardan, Pakistan

5 Department of Software Engineering, College of Computer and Information Sciences, King Saud University, P.O. Box 103786, Riyadh 11543, Saudi Arabia

6 Department of Information Systems, College of Computer and Information Sciences, Princess Nourah Bint Abdulrahman University, P.O. Box 84428, Riyadh 11671, Saudi Arabia

7 Department of Computer Science, Women University, Mardan, Pakistan