A Taxonomy of Contextual Factors in Continuous Integration Processes

Shujun Huang
Delft University of Technology
Delft, The Netherlands
s.h.huang@tudelft.nl

Sebastian Proksch
Delft University of Technology
Delft, The Netherlands
s.proksch@tudelft.nl

***Abstract*—Numerous studies have shown that *Continuous Integration* (CI) significantly improves software development productivity. Research has already shown in other fields of software engineering that findings do not always generalize and are often limited to a specific context. So far, research on CI has not differentiated between varying contexts of the studied projects, which includes, for example, varying domains, personnel, technical environments, or cultures. We need to extend the theory of CI by considering the relevant context that will impact how projects approach CI. Although existing studies implicitly touch on context, they often lack a consistent terminology or rely on experience rather than a standardized approach. In this paper, we bridge this gap by developing a taxonomy of relevant contextual factors within the domain of CI. Using grounded theory, we analyze peer-reviewed studies and develop a comprehensive taxonomy of contextual factors of CI that we validate through a practitioner survey. The resulting taxonomy contains multiple levels of details, the main dimensions being Product, Team, Process, Quality, and Scale. The taxonomy offers a structured framework to address the gap in CI research regarding contextual theory. Researchers can use it to describe the scope of findings and to reason about the generalizability of theories. Developers can select and reuse practices more effectively by comparing to other, similar projects.**

***Index Terms*—Continuous integration, context, generalizability, Grounded theory, framework**

I. Introduction

CONTINUOUS Integration (CI) has been widely adopted by organizations of all sizes to boost productivity [38, 58, 86]. Numerous studies have demonstrated that CI can effectively improve processes [119, 142] and enhance organizational performance [86, 91]. In research, we have observed that enhancements to CI practices have been examined across diverse domains, personnel, technical environments, and culture, with solutions specifically adapted to each unique context. These case studies highlight the significance of contextual factors in CI. For example, studies have explored the different challenges faced by large companies [30] and small enterprises [57] when implementing CI. Empirical studies in various factors have shown that project background, tools used and their popularity [38, 58], team size [35, 57, 64], and organizational culture [86] also influence CI practices and efficiency. Additionally, we observed large companies like Amazon [6], and Airbnb [79] have dedicated teams to manage CI/CD pipelines and develop custom DevOps platforms, accommodating diverse development teams and software sources. These organizations typically build fully automated pipelines to support multiple environments and balance code and configuration needs. While smaller organizations and some open-source projects often use more cost-effective, cloud-based CI/CD solutions like Jenkins, GitHub, or GitLab CI/CD [38, 52]. In the rapidly changing global landscape, a unified CI practice approach is becoming less applicable, the practices need to adapt to different national and cultural environments [5, 29, 40], the results of case studies are often not generalizable beyond their specific context [78].

The project context has long been an important topic of discussion in SE [36], even though the meaning of the term *context* varies in related work. In our research, we define it as situational variables that directly influence behavior or affect the relations between variables [36]. Research has shown that contextual factors heavily influence the applicability and scalability of software engineering solutions [20] and practitioners frequently adjust development methods based on context [7, 8, 60, 75, 87, 117]. Besides, context-aware frameworks can help practitioners with making informed decisions [75].

In SE theory-building, many studies suggest that *mid-range theories*, which clearly define their scope and relevant phenomena, are more practical and useful than *universal theories*, with the context being a crucial factor in determining the generalizability of research results [55, 112, 127, 128]. These findings underscore the importance of context in both empirical and theoretical research, supporting our discussion of its relevance within the CI domain. There are no *universal* best practices for CI, as research findings cannot be applied without considering specific contexts [36]. Therefore, once a comprehensive framework for describing context is established, it can be used to define the boundaries of applicability for existing CI practices, offering more suitable recommendations to organizations [75]. An effective way to achieve this is through theory-building [75], which can abstract core elements and provide a structured framework for comprehension.

As a discipline that bridges physical machinery and human empirical work, the importance of theory in SE has been examined from various perspectives [70, 111]. Constructing theories, especially middle-range theories, plays a vital role in advancing the maturity of the discipline [17, 22]. However, the CI domain lacks theoretical research and a discussion of the role of project context. While some existing studies implicitly address context, these discussions are often unsystematic, have inconsistent terminology, or rely on personal experience rather than a standardized approach.

In this paper, we will address this gap by providing a framework that comprehensively catalogs and explains the contextual factors within the CI domain. These factors refer to variables that may have practical impact on the efficiency, stability, and maintainability of CI pipelines. Our research follows the assumption that all CI practices contribute to the same goal of establishing a well-functioning CI pipeline [42]. As such, for CI pipleines, our framework covers both creating new pipelines and maintaining existing ones.

In most cases, prior studies described these variables either explicitly or implicitly through background information, experimental design, or research assumptions. In order to develop a systematic and practical result, we formulated the following research questions. We first used existing literature as the foundational knowledge base, and then validated and refined it through insights from domain experts and practitioners:

RQ_1: Did prior CI research consider project context?
RQ_2: Which contextual factors get discussed in prior studies?
RQ_3: Do practitioners agree with the novel taxonomy of contextual factors?

To achieve this goal, we employed grounded theory (GT) as our methodology, which has already proven highly effective in many fields, including software engineering [28, 59, 113]. GT can generate mid-range theories from data [15, 50], which aligns with the objectives of this paper. We follow an established methodology for systematic literature review [4, 60, 78] and investigate representative studies on CI from prestigious venues [51, 113]. The subsequent data analysis was then conducted by two researchers and one validator in the open coding process, a third researcher served as mediator and also supported the finalization of the taxonomy. We validated the resulting taxonomy through a practitioner survey. Large-scale human experiments are costly and potentially influenced by personal experiences, but we decided to include practitioner feedback to prevent accidental omissions of contextual factors.

Our literature review revealed that many previous studies expressed interest in contextual factors. More than half of the studied papers (63%) have included at least one research question that studied the effects of specific contextual factors. Aspects like project size and technical environments are most commonly examined contextual factors. The survey that we conducted among practitioners demonstrated the convergence of our taxonomy as we could not identify further missing concepts. Among the 34 practitioners surveyed, we found strong support of our taxonomy and less than 10% of them disagreed with some aspects. Their open feedback can help to shape future work in the area. We find that our results emphasize the importance of considering a broader range of contextual factors when deciding about CI implementation and resource allocation. Looking ahead, we hope that by effectively describing and structuring *context* information, this framework will help identify appropriate *best practices* across different types of contexts and paradigm combinations, enabling practitioners and researchers to find the most suitable practices based on specific contextual dimensions.

Overall, this paper presents the following main contributions:

- A summary of the current state of CI research on context.
- A comprehensive taxonomy of contextual factors in CI, which offers a unified approach to understanding and defining relevant context.
- Actionable insights and recommendations for future CI research to improve generalizability of findings and to clarify the applicability of practices.

All raw data of the paper, the analysis scripts, and extended results have been published in an online appendix [62].

II. Related Work

This section discusses the previous work in the key research areas of CI, as well as practical approaches.

Context in Software Engineering. Context has been explored in different disciplines. In linguistics, it is defined as the inferred meaning of a sentence or paragraph based on its relationship with internal cues within the text [25]. Johns distinguishes an *omnibus context* (macro context) and a *discrete context*. The former includes broad factors such as *who, what, when, where, and why*, while discrete context refers to specific contextual variables [69]. Most empirical software engineering research adopts the discrete context perspective [36], which aligns with our research objectives.

In software engineering, discussions around context have primarily emerged because researchers have recognized that software development is a dynamic lifecycle involving complex human activities [26]. As a result, developers have sought methods and frameworks to standardize and organize the software development process, such as CMMI [115] and Scrum [102], which benefited developers in various ways [41]. However, as technology evolves, some widely promoted methods like Agile have been suggested to perform better in specific contexts, such as in small software development teams or when applied by more experienced developers [16, 84].

Researchers have increasingly recognized that one of the main reasons for the complexity of software development lies in the complexity of its contextual environment. The goals of software development need to align with project requirements, and both resources and contextual factors must be evaluated. No single methodology can guarantee the same success across, for example, organizations of all sizes or teams with varying skill levels. Some scholars have already started building relevant frameworks for understanding context [26, 75]. Researchers have recognized the importance of discussing context in guiding both practical development and theoretical research in software engineering, as its universality is often overestimated [20]. Case studies, for instance, are conducted within the specific context of a particular company, meaning they are influenced by unique process standards, application domains, staffing practices, software tools, and other environment-specific factors. Context can also affect

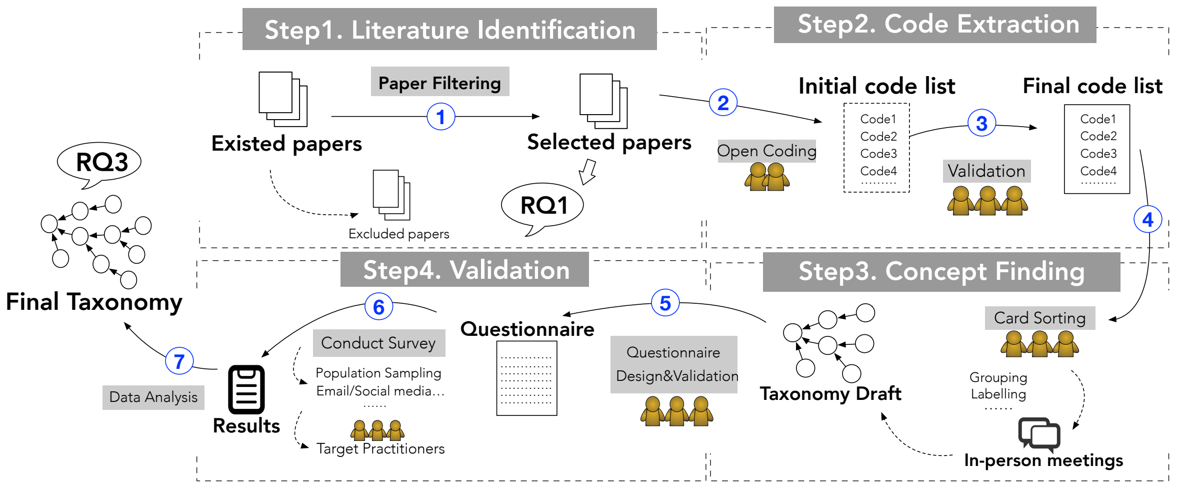

Fig. 1. An overview of the paper that illustrates the methodology and the connection between the different research questions.

metrics such as project maintainability [140], organizational management [69], and more. Findings from case studies are often not generalizable beyond their specific context, making it difficult to extract meaningful evidence, and classifying contextual factors can help to promote these finding [78].

Evolution in CI. As an efficient practice, the importance and prevalence of CI have grown significantly in SE since its introduction [58]. Over time, CI has witnessed remarkable technological advancements, with more open-source organizations and large-scale companies adapting it into their daily development processes [32, 104, 107, 143]. CI has not only attracted considerable attention in industry but has also become a comprehensive research topic in academia. Numerous studies confirm CI's role in enhancing software quality, productivity [119], refining the build process [12, 58], and increasing development speed [86]. This explains the growing preference for CI among developers and researchers.

In-depth research reveals that implementing CI is not always easy and can have significant costs, unsuccessful builds [66] or frequently failing tests [10] commonly happened. During development, developers face trade-offs involving assurance, security, and flexible configuration [57], as well as potential barriers from stakeholders, such as complex architecture, time constraints, organizational distribution [81, 89, 118], and other technical problems [35, 121, 125], including anti-patterns emerging during the CI implementation process [46].

The entire community is making significant efforts to enhance CI efficiency, like reduce failed builds and tests [39, 44, 98], or refining strategies for regression testing [14, 80] and performance testing [65]. Many studies have also contributed valuable insights into the use of CI tools [52, 97] and how they evolve with industry practices [43, 136]. This indicated the dynamic nature of CI development. As projects progress and mature, CI strategies need corresponding adjustments.

Contextual problems. Similar to challenges faced in the broader field of software engineering, many scholars have recognized that there is no one-size-fits-all solution or universal set of practices for CI. The actual solutions need to be dynamically adapted based on the context and the specific stage of the project. Although there is currently no unified contextual framework, many studies have observed the influence of various contextual factors. For instance, we have identified many factors affecting CI research, including corporate backgrounds, project themes, personnel configurations, release strategies, and user markets [38, 103, 107, 125]. Additionally, technical characteristics of a project can also affect CI performance [12, 92]. Reports show that developers must continually adjust based on the specific characteristics of their project's development stage [86]. While some studies do not explicitly focus on contextual issues, they implicitly select projects based on specific language features or technical backgrounds [23, 83, 139]. Some contextual factors have been discussed implicitly. For example, implementing CI may be more challenging in smaller organizations due to a lack of experience and cost-related constraints [57]. Further research examines how conclusions are influenced by factors such as popularity [58] and team size [35, 57, 64]. There are also studies that explore CI practices in non-traditional domains, which highlight the importance of contextual factors related to project domain types [85, 116, 123]. However, the current discussions or capturing of contextual factors often rely on historical research or ad-hoc perspectives [113]. There is still a lack of a comprehensive reference framework to systematically address contextual factors.

Grounded Theory. Theory construction plays an essential role in advancing a discipline, and this is especially important for a field like software engineering, which intersects with human organizational activities and technological development [55, 127]. Grounded theory is a qualitative research method where researchers build theories based on data [50]. Unlike traditional deductive research methods, grounded theory generates mid-range substantive theories by systematically analyzing data, without predefined research hypotheses. It has proven to be an extremely useful research method in various fields [4, 61, 113]. Using grounded theory, we aim to develop a systematic framework for understanding context based on existing CI research. Numerous studies suggest that mid-range theories are more practical and valuable compared to universal theories [55]. A well-constructed contextual framework will help define the scope of theories and their applicability.

In this section, we have identified some of the related work in software engineering and the CI field, alongside the methods we plan to reference. We observed that there is currently no systematic framework in the CI domain, which may weaken our ability to identify key constraints and characteristics of software development environments [26]. Our research aims to build a systematic contextual framework based on historical research using grounded theory, which can help inform future research directions and address current research gaps.

III. Methodology

Our study design is inspired by methodology of previous work [3, 26, 88]. We conduct a mixed-method study that is based on grounded theory. We observed from previous theoretical and framework-building literature that they typically used historical documents as data sources, and applied grounded theory to systematically collect, process, and analyze the data. Some studies also involved practitioners for theory validation. Since the core of this research is to transform the data into a coherent classification of contextual factors and generate core concepts, we used historical literature from the CI domain as a comprehensive and systematic data source. Therefore, we employed a systematic literature review (SLR) to collect relevant studies and incorporated the analysis techniques from grounded theory: coding, constant comparison, and memoing. Additionally, given the high cost of manual surveys, we conducted relevant validation after the framework was developed.

Figure 1 provides an overview of the mix-method approach, which consisted of four steps, *literature identification*, *code extraction*, *concept finding*, and a *practitioner survey*. The specific details are discussed in following sections.

A. Literature Identification

Our systematic literature review follows the methods from Kitchenham et al. [76, 77]. This helps us with effectively identifying relevant research and ensures the reliability of our secondary research findings. In this step, our objective is to identify high-quality research related to CI topics, however, the amount of previous works that mention the concept of *continuous integration* is overwhelming. We needed to employ strategies to limit the results to a feasible subset that still allows us to saturate our identification of context factors. We decided to restrict 1) our search keywords and 2) the scope of journals/conferences to ensure the quality and relevance of the selected literature. To avoid compromising the specificity of our literature selection that we reach through the restriction on certain venues, we also decided against applying a snowballing technique. We are not claiming to have performed an *exhaustive* review over all available literature, but we saw a saturation of topics in our survey, so we are confident that our selection is *representative* for the complete population of related papers. The remainder of this section will introduce our concrete methodology for identifying related literature.

Search String. To find studies related to contextual factors of CI, we searched for articles that contain the search terms *"continuous integration"* or *"CI"*. We included *"CI"* as a query word because it is a common substitute for *"continuous integration"* in the software field and is frequently used in titles. We limited the search scope to titles only because the term *"CI"* resulted in many false hits that mentioned the term in abstracts or introductions, often while referring to the use of CI tools or pipelines rather than CI-related studies. We initially also included *"DevOps"* as a keyword due to its shared core concepts with CI. However, we observed that only a small number of search hits were related to CI, which attributed to the nuanced relationship between CI and DevOps. While DevOps emphasizes agile development practices and mindset, CI is often seen as one of the technical strategies within DevOps. As a result, we excluded *DevOps* as a keyword.

Search Methodology. We used the search string *"continuous integration"* or *"CI"* to find relevant literature in all major digital libraries, including the ACM Digital Library, IEEE Xplore, Science Direct, Springer Link, and Google Scholar. The results have been limited to conference proceedings and journal papers published between October 2000 and October 2023. As elaborated before, the breadth of the survey topic made an exhaustive review infeasible, so we only included high-quality publications in the key venues of the software engineering field. Table I provides an overview over the included venues and the number of included publications for each venue.

Inclusion/Exclusion Criteria. Given the breadth of the search terms, the search results included many unrelated works or disqualified material. To ensure thematic proximity, we found it most effective to establish a set of exclusion criteria rather than refining the search terms. We only accepted papers in main track and excluded papers from co-located events, like workshops or doctoral symposia that might have been misclassified. We also excluded paper that lacked substantial content, such as *New Ideas* paper, talks, and posters.

Level of Context Consideration. To gain a more comprehensive understanding of the scope of focus during the open coding phase and assess the overall attention given to contextual factors in the literature, as well as to obtain an overview of the current state of the art, we classified the remaining papers based on their varying degrees of focus on project contextual factors. We analyzed each paper by abstract and introduction to grasp the main idea of the research. Then we applied the following *classification criteria* to differentiate between three types of articles, considering the thematic content and the

TABLE I
PAPER SOURCES FOR THE SLR

Name	Description	#
Conferences		
ICSE	Int. Conf. on Software Engineering	10
MSR	Int. Conf. on Mining Software Repositories	18
ASE	Int. Conf. on Automated Software Engineering	13
FSE	Int. Conf. on the Foundations of Software Engineering	8
ICSME	Int. Conf. on Software Maintenance and Evolution	9
SANER	Int. Conf. on. Software Analysis, Evolution and Reengineering	9
Journals		
TSE	IEEE Transactions on Software Engineering	10
EMSE	Empirical Software Engineering	9
JSS	Journal of Systems and Software	7
TOSEM	ACM Transactions on Software Engineering and Methodology	3
Total		96

strength of their connection to the project context: *Strong*, *Weak*, and *None*, as we read through the papers, we systematically filter based on the following criteria:

Strong: Any article that addresses at least one research question regarding how contextual factors influence the setup or improvement of CI pipelines. For example, articles may directly discuss the challenges, obstacles, or endeavors to explore CI in various project contexts.

Weak: If the paper doesn't fit the previous criteria, we will check whether the article's theme focuses on researching any project contextual aspects. In such cases, we consider this type to have a *weak* connection. For example, discussions about the evolution and changes in CI tools, or research on how factors such as team size affect the efficiency of CI, fall into this category.

None: If neither the article's theme nor any of its research questions address the impact or study of project context, we categorize it as having no relation. For instance, the article might only discuss how test case selection can reduce runtime cost of CI pipelines, which we consider unrelated.

Our classification aims to provide a clearer understanding of the current state of project context research. The first category refers to articles that explicitly discuss the implementation of CI in various contextual settings. The second category includes factor-based research that focuses on specific aspects. Although we categorize the final group of articles as *"weak"* due to their thematic focus, it is important to note that all CI-related research inherently involves contextual factors. Methodological details and potential factors arising from the experimental design in these papers still serve as valuable input for building our taxonomy framework.

Results. Our search in the digital libraries has revealed 127 paper candidates. Through applying our exclusion criteria, we filtered the candidates down to 96 relevant papers. Among these papers, we found a publication peak between 2017 and

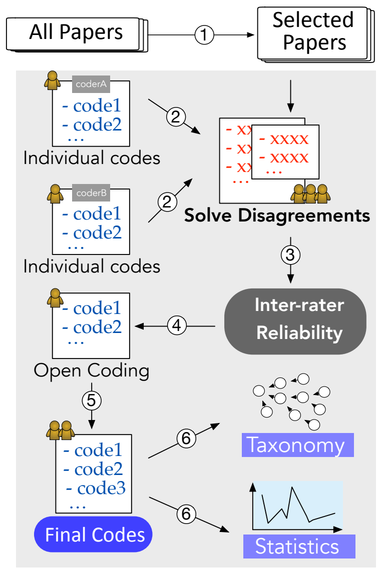

Fig. 2. Open Coding Steps

2022, which was likely caused by the introduction of the TRAVISTORRENT dataset. The dataset has been used as the mining challenge in the MSR conference in 2017 [11], which sparked related research.

B. Code Extraction

After obtaining the selected papers, we used *Open Coding* to extract the contextual factors that gained attention in the previous studies. We excluded two literature review papers to avoid duplicate codes and analyzed 94 papers. Open coding is an inductive approach defined in Grounded Theory [73], which allowed us to identify and define relevant concepts from the data. Three coders, including one author, participated in the open coding process. One coder has relevant expertise in software engineering, and has experience using open-coding, another coder is an expert in the software field. After completing the coding, we validated the resulting codes and their association with the taxonomy. The overall procedure is depicted in Figure 2.

Establishing Coding Guidelines. The coders independently read the identified papers to extract relevant codes. Initially, we started with conducting several rounds of open coding with 4-5 randomly selected papers to establish our coding guidelines. Discussing the results among the coders helped us to establish a shared vocabulary and to clarify the review procedure. A key insight was that the differences in paper lengths and specialized content complexity led to varying code interpretations, so we prioritized sections for reading based on the level of project context relevance determined in the previous step. For papers *Strongly* related to the project context study, we thoroughly examined all content or results in the research question. For others with *Weak* or *None* link to project context, we focused mainly on the methodology section, as it typically outlined authors' attention to open-source project datasets for CI research and sampling plans. After several discussion meetings, we converged towards the following coding guidelines.

1) Codes must relate to the theme of the RQ$_2$. We try to find *"what contextual factors have been studied in the historical literature."* Our emphasis on content will focus on answering key inquiries such as *"What factors are examined in the CI projects?"* and *"What criteria are utilized for selecting CI projects in the research?"*
2) Use minimal-level contents. Due to the concise nature of paper language, for summarized terms, we code the minimal-level descriptions unless the paper lacks details. For example, instead of using a broad code like *"size"*, we specify codes such as *"team size"*, *"organization size"* and *"number of files"*.
3) Ensure codes are observable or measurable. Extracting factors that can be directly observed, measured, or described using specific tools. Non-quantifiable codes are acceptable if they can be described in concrete terms.

When we found relevant statements, we marked them in the original text, matched them with the corresponding code from our vocabulary, and documented the occurrence.

Disagreement Resolution. After each round, the coders compared the extracted codes to arrive at a shared terminology. We discussed the disagreements and found that often non-uniform terminology caused deviations between coders. To address this, we unified the names of recurring codes and established a standardized vocabulary of codes. Emerging codes were added after a discussion.

A recurring challenge was distinguishing whether a factor was *influencing CI* practices or an *outcome of CI*. For example, changes in delivery frequency could be a deliberate process change of a project or the natural result of a proper adoption of CI. We have discussed such corner cases when they emerged and have refined our review guidelines accordingly. We followed a conservative strategy and included all potentially relevant elements, regardless of their possible interpretation in different scenarios. Furthermore, we discussed the broad scope definition of project context, as initially defined. While some papers present their own definitions for *project context* or *process factors*, we found that these categorizations are always closely tied to the specific goals and backgrounds of the paper, so we chose to disregard their categories and include codes as long as they meet our criteria.

Results Reliability. We captured the extracted codes per article in simple text files, which allowed us to validate a consistent usage of codes across the various articles and easily quantify the reliability of our coding process through calculating the *inter-rater reliability* (IRR) [27]. We found that the sparsity of the code would inflate the IRR when computed on all possible codes, as such, we assessed the agreement only on the union of codes extracted by any of the coders, which is a much stricter comparison. In the first two rounds the resulting IRR was too low (0.31 and 0.27), which indicates only a *fair* agreement for the paper selection. Through the discussions after every round, we have matured our vocabulary of codes and our review guidelines and the agreement improved in subsequent rounds. After 4 rounds and a review of 19 papers, we achieved an IRR of 0.63 (*substantial* agreement). This made us confident that the coding task had been clarified enough to be repeatable and we moved on to Step ④, where one coder finished the individual coding for the remaining papers. In Step ⑤, we merged the generated codes to form the final codes. While the first paper reviews have added many new codes to our vocabulary, the last 5 papers have only created 5 new codes. None of these codes has been used more than one time in total throughout the whole set of all 94 coded papers, so we concluded that we have reached saturation and did not push for adding more papers to our sample. Finally, in Step ⑥, we conducted the analysis based on the final codes.

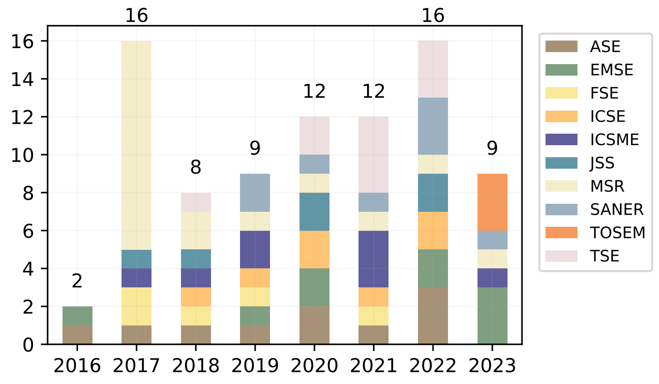

Fig. 3. Overview over included papers from 2016-2023 and venue

Results. When completing open coding for all the papers, we reached the point at which we could no longer find new codes. Being able to reach a saturation point despite our initial search filters and the lack of a snow-balling showed us that we still included a sufficient number of papers. Ultimately, we were able to identify a total of 343 unique codes across all 94 papers. These codes build the foundation for the subsequent analyses and for our research questions that we will present in the following sections.

C. Concept Finding

We analyzed the identified codes to build a taxonomy that could hierarchically represent the previously identified codes. We aimed to establish balanced, high-level categories that comprehensively capture the different aspects of context, providing an abstract framework that could apply to real-world projects. This process was divided into two phases: *axial coding* and *card sorting*.

Axial Coding: Based on grounded theory, axial coding can be used to link codes using a mix of inductive and deductive reasoning [72]. We have organized our extracted codes along with common topics to reflect the different dimensions of project context and their relevance level in CI practices. Codes were grouped into distinct clusters without overlap to maintain consistency.

Card Sorting: As a second step, we utilized card sorting [106] for organizing and structuring our codes and topics into a multi-level hierarchy.

By integrating both methods, we could logically combine closely related codes, and then identify conceptually similar higher-level concepts to form branches. This process involved two authors who participated in face-to-face or online meetings, where they utilized an online tool for card sorting, all codes were displayed and discussed during these meetings.

The concept identification was an iterative process that was spread out over several meetings. Classifications that could not be determined temporarily, were set aside for discussion in subsequent meetings. It took four extended discussions to reach a consensus and construct the basic taxonomy. We established a taxonomy comprising five main categories and 14 subcategories. This taxonomy will be presented in Section V as the answer to RQ_2.

Independent Validation. To validate the reliability of our coding results, we invited an independent software engineering researcher to apply the extracted concepts that emerged in our taxonomy construction process. To keep the efforts reasonable, we provided 20 randomly selected papers and also the set of all second-level categories that we have included in our taxonomy. We asked the expert for each of the papers to identify all categories that apply. We compare the outcome of this independent labeling with the labels to which we have mapped the paper in our own process. The IRR between our coding results and the expert's evaluation reached 0.62, which shows *substantial* agreement and makes us confident in the validity of the overall coding procedure.

D. Practitioner Survey

As a second approach to independently validate our taxonomy and gather additional insights, we surveyed practitioners. We developed a questionnaire based on the taxonomy and distributed it to industry professionals. Our validation process proceeded as follows:

Questionnaire Development. The questionnaire has been developed in two stages. First, we have drafted a questionnaire based on a preliminary taxonomy. Two experts from academia and industry were invited to evaluate the questionnaire. Second, we engaged in iterative discussions with an additional author to incorporate and refine the feedback received. This iterative process includes three rounds, leading to the finalization of our questionnaire. The questionnaire has three sections: demographics, open thoughts about CI, and questions that allow us to validate the taxonomy.

To ensure consistent language and clarity about terminology, we provide a brief introduction to CI and related practice concepts at the beginning of the survey. We also introduce the the first two levels of our taxonomy in the survey, as illustrated in Figure 6. We provide detailed descriptions for all taxonomy items to ensure that participants understand their meaning, but avoid listing the full taxonomy to avoid an information overload for our participants.

Overall, we aimed for a survey completion time of about five minutes to ensure participant engagement. The full questionnaire can be found in Table II.

Survey Distribution. The questionnaire has been implemented using QUALTRICS[1]. We have distributed the questionnaire on public platforms (such as Reddit, Quora, etc.) and industry forums, but we also forwarded it to personal contacts in academia and industry. Our target audience included individuals who have professional experience in the context of continuous integration, preferably in development departments of enterprises, regardless of industry type. We specified these required qualifications in our invitations to find our target audience on public platforms, but we also used these qualifications to identify participants in our personal connections with relevant experience. The anonymous responses make it impossible for us to distinguish between responses, but we estimate that not more than 10% of the responses came from personal contacts and the remaining ones from public sources.

Cleaning the Results. We have received a total of 106 responses to our survey. To maximize the reliability of our survey data, we have filtered 33 incomplete questionnaire submissions from our results and another 13 responses that have been completed in less than two minutes. Additionally, we filtered out 10 responses from participants who reported that they had never used CI tools and 2 responses in which participants rated their own programming proficiency at the lowest level, as this raises doubts about their qualification for participation. We decided to also exclude 14 responses that selected the same option in all multiple-choice questions and did not fill in any open-ended answers, as we assume that these participants did not fill the survey with the expected rigor. After applying these filters, we were left with 34 high-quality responses that we deemed suitable for further analysis to answer our research questions.

IV. DID PRIOR CI RESEARCH CONSIDER PROJECT CONTEXT? (RQ_1)

This research question explores to which extent previous work has considered contextual factors and how these factors are captured. This can guide future research on the impact of specific factors. Figure 4 provides an overview over the identified papers per venue and how the level of context-awareness differs across major venues. Note that *Strong* and *Weak* only indicate the article's focus on contextual factors and present no judgment of the quality of the presented work.

Overall, we have collected 96 papers, out of which 72 (75%) demonstrate at least a *weak* concern for contextual factors in their research design. They either focus their themes on contextual factors and how they affect CI implementation or delve into specific aspects of the project context. As the papers are distributed across all included venues of our survey, we can conclude that research on CI is not a niche topic, but that it is widely relevant in the area of *Software Engineering*.

[1]https://www.qualtrics.com

TABLE II
SURVEY QUESTIONS (O: OPEN ANSWER, MC: MULTIPLE CHOICE, C: EXCLUSIVE CHECKBOXES)

Section	ID	Question	Type	Answer
Demographic	Q1.1	What's your current/most recent job title in the software engineering field?	O	Open answer
	Q1.2	What is the size of your development team/department?	C	1 - [2,5] - [6,10] - [11,20] - [21,50] - [51,99] - 100+
	Q1.3	What is the highest degree of education you have obtained?	C	High school, Training, Associate, Bachelor, Master, PhD, N/A
	Q1.4	Which main programming language do you primarily use in your projects?	MC	Java, Python, C++, JavaScript, Ruby, PHP, Others
	Q1.5	On a scale, How would you rate your programming skills?	C	Scale of 1 to 5
CI Thoughts	Q2.1	How often do you apply CI practices(setup/maintenance) in your projects?	C	Never (I have never applied CI in my work) Rarely (I use CI approximately once a year or less) Occasionally (I apply CI a few times a year) Frequently (I use CI on a monthly basis) Always (CI is a constant part of my daily or weekly workflow)
	Q2.2	Do you believe there's a one-size-fits-all approach to CI implementation?	C	Yes, No
	Q2.3	Can you briefly explain the reasons?	O	Open answer
Taxonomy Validation	Q3.1	How much do you agree that these aspects impact CI implementation?	C	strongly agree, agree, neutral, disagree, strongly disagree
	Q3.2	Do you think any of the mentioned aspects should be changed/removed?	O	Open answer
	Q3.3	Do you think any additional aspects should be added?	O	Open answer

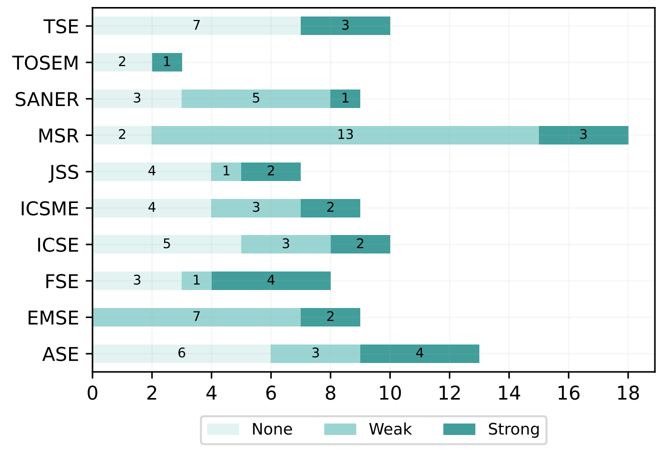

Fig. 4. Conference Distribution by Context-Type

A look into the details revealed the highest number of papers in the conference on *Mining Software Repositories* (MSR). We investigated this phenomenon and attribute the observed effect to an *MSR Mining Challenge* that focused on a TRAVISCI dataset, which created a strong bias in our numbers. We also see a high number of relevant papers in the *Empirical Software Engineering* (EMSE) journal. We believe that the empirical nature of the research that is presented at both venues is a second contributing factor. When extensive software data is used to extract actionable insights for enhancing software engineering practices, it seems natural to also consider specific information about the projects and their processes in the investigation to explain differences.

We have categorized the 96 surveyed papers according to the extent to which that have considered context, as outlined in Section III-A. Our results show that 24 (25%) exhibit a *Strong* link to project context, with theme or research question aligned with studying CI performance/setup affected by contextual factors. Another 36 (39%) papers have at least a *Weak* connection and build on research or explore some specific aspects of project context. We interpret this as a sign that most researchers are intuitively aware of the potential impact that the context might have on their experimental results, however, more than a third (36, 36.4%) do not consider the project at all (*None*). In the following, we will investigate the three different context-awareness categories more closely and describe patterns and observations that we found.

A. Context Relation: None

This category includes 36 identified papers. An assignment into this category does not imply a complete absence of context-related considerations, but rather indicates the influence and utility of contextual factors is not the primary focus of the paper. Many papers we found have focused on topics like test-case selection or build-failure reduction, but they often fail to investigate the project characteristics that affect their research directions or findings. Most of the time, project context is only indirectly mentioned in the methodology descriptions, for example, a discussion of the suitability of builds or commits for the respective studies, or a selection of projects based on certain popular project characteristics such as programming language, project size, watchers, stars, or builds. As project context is not the main topic of these articles, we try to find implicit references to project context in the methodology sections and we found two types: *Data Accessibility* and *Relevance to Research Scope*.

Data Accessibility. These articles typically aim to efficiently select projects with available data, or focus on projects with higher visibility to ensure reliable research outcomes. Factors such as project language, activity levels, and popularity are commonly considered for selection. For instance, some studies filter projects based on lines of code and build information [139], while others use popular projects like stars [24, 121] or analyze recently active projects [80, 83]. Commits frequency and project size are also widely used indicators for project selection [54, 80, 82]. Moreover, certain studies select projects from specific databases such as TRAVISTORRENT [45, 141] or GHTORRENT [121, 129], while others rely on previous research findings or dataset [21].

Relevance to Research Scope. Many papers require the selection of relevant projects as study objects or for validation. This desire can guide the selection of projects that are thematically suitable. For instance, to show a reduction or successful prediction of build failures, it is required to find a sufficient number of OSS projects that actively use a build server [122]. Another example is a study on the impact that pull requests have on the delivery time, which besides the popularity of projects has also filtered out projects with less than 100 linked PRs to find suitable study objects [12]. Finally, research that focuses on test-case analysis needs a considerable quantity of workable test cases and will naturally select projects that contain active tests [23], without explicitly mentioning this as a characteristic of the project context.

In summary, we recognize that the methodological details presented in these papers are still valuable contributions for building the taxonomy framework. The focus on project selection, particularly on areas like build and test case selection, tends to prioritize test suite representation and build types over contextual factors. Indeed this type of approach is the most efficient and cost-effective choice for experimental design.

B. Context Relation: Weak

This category comprises 36 selected papers, where topics are *weakly* tied to the project's context. Articles in this category typically do not consider project context as a whole, but only explore one aspect of it, such as a particular tool, language, or any other specific dimension of the project context.

Various Contextual Aspects. The project context consists of very different attributes that describe diverse project characteristics and development processes. In our survey, we observed that most articles focus on very specific aspects for their research. In this type of research, it is common for articles to include one or more research questions investigating the impact of specific contextual factors. For example, studies may ask questions such as *"What is the influence of code characteristics?"* [14] or *"Do the sizes of teams and projects have any correlation with CI build failure?"* [64]. However, the primary focus of these articles lies often on other aspects, like improving regression testing strategies or reducing build failures. We acknowledge that these topics relate to contextual research to some extent and, based on our inclusion and classification criteria, we categorize such articles under the *Weak* context relation category. Among these contextual aspects, many articles concentrate on technical and activity-related factors, such as code characteristics [14], the utilization of static code analysis tools in CI pipelines [135], the impact of integration environments [10], changes or types of development activities [96], or the connection between project features and build failure types to predict the first build [66]. Additionally, some articles explore the potential impact of project complexity and build strategies [64].

We also notice a growing interest in non-technical factors. For instance, some articles investigate the interplay between non-functional requirements and build outcomes [93], the sentiment of commit messages [105], or human-related aspects such as personnel growth and CI [89], or changes within the Travis CI team [89].

Many studies do not even directly discuss the role of project context in CI, but they still consider the project context through their methodology. For instance, some studies provide extensive statistical analyses to reflect on project characteristics [1, 13, 129], like explore the correlation between process factors and pull request merges [132]. Certain studies also limit their scope to specific programming languages due to tool limitations, like the abstract syntax tool's capability to parse only Java code [101]. But not all the statistics presented directly impact the research outcomes as dependent variables.

Broad CI Improvements Study. Another subset of research focuses extensively on various improvement studies within CI automation. Identifying this type of article is challenging because they typically do not explicitly discuss individual contextual factors. Instead, they address improvements and optimizations in CI from a broader and more macro-level perspective. For example, prior work implicitly discusses aspects of tool selection and evolution through the analysis of trends such as the *"most frequent combinations of CI tools"* [52]. Similarly, discussions around challenges in CI usage and bad practices, such as the investigation of *"the relationship between CI anti-patterns and developer behavior"* [136], inherently touch upon external and internal environmental factors.

Although these studies do not directly focus on the overall project context, the findings still provide valuable insights into understanding the influence of contextual factors on CI practices. Therefore, we categorize such articles under the *weak* context relation group. This type of research also includes other studies like the analysis of configuration files within CI, such as restructuring operations in CI/CD pipelines [133], and the development of tools aimed at identifying and solving potential issues within CI configurations [138]. Additionally, discussions concerning information exchange between continuous integration servers contribute to the detection of software development issues [33], as do considerations regarding the activity profiles of stakeholders [19]. Another type of research focuses on optimizing specific stages within automation. For instance, some study the file changes (such as file name, number of files, file types, etc.), terms in commit messages, and contributor experience to skip the broken builds, to save cost for large-scale companies [68] [99]. Solutions have also been proposed for automatically fixing errors in build scripts, preventing software build failures [56]. Researchers also explore tools for traceability in successful large-scale continuous integration and delivery processes [109].

Although these studies cover diverse motivations, they collectively contribute to the understanding of the different aspects of automation of CI projects, and focus on different aspects in context.

In summary, we found a substantial body of literature that can be assigned to this category. Many papers have investigated CI efficiency and examined at least one part of diverse project contexts or the progressive adoption of CI across various domains. While the topics of these articles may vary widely, ranging from enhancing CI practices to exploring new applications of CI, their methodologies and topics emphasize the important role that project context has on the outcomes of CI.

C. Context Relation: Strong

We have identified 24 papers that study the impact of project context on the setup or execution of CI in at least one research question. For instance, corresponding papers may model the setup of CI pipelines or improvements in various contexts, or they may explore how to implement CI in non-traditional fields outside of software engineering. Because these articles share the same motivation as our exploration of contextual factors, we consider them *strongly* relevant to the study of project context. We found that these papers that fall into two categories: *Studying Context* and *Exploring New Fields*.

Studying Context. The first type of article directly examines the importance of project context characteristics, such as how various aspects of a project affect the implementation and improvement of CI. Some articles delve into the characteristics of project context and specifically address how project types shape CI implementation. For instance, some study categorizes four different types of OSS repositories in the CI context based on popularity, size, and testing, arguing that different repository types may be suitable for different members and have cultural implications [47]. Others investigate the differences in CI implementation across various cases [46, 107]. They may also discuss the type of CI users [34] or analyze which types of projects are more likely to embrace CI [58]. In these studies, they paid attention to project context, often focusing on enhancing the efficiency of CI automation steps. Questions like *"What project characteristics are associated with build performance?"* [49] demonstrate this focus.

On the other hand, another type of article discusses the challenges encountered during CI implementation by exploring the obstacles and requirements faced in different contextual settings. These studies may examine the reasons behind CI abandonment in certain open-source projects or the barriers encountered [126, 134]. Other survey-based research studies did interviews with software practitioners to explore how factors like organizational size, development approach, and CI preferences lead to changes in CI practices and evolutions [110, 125]. They discuss the evolution of CI in projects, analyzing both advantageous and disadvantageous factors within the context and involving more human-related insights.

Overall, these articles that explore the influence of contextual factors on CI implementation share the same motivation as our research, which aims to investigate how contextual factors shape CI adoption.

Exploring New Fields. Another type of study explores the application of CI in *new* domains where CI is not traditionally prevalent, such as non-software fields. Their exploration highlights how new contexts shape the method of CI implementation. For instance, researchers explored the challenges of adopting CI in new architecture software [31, 116], automotive industry [123], or Android apps, [85]. Even though the existing study mostly focuses on describing the current state, without extensively comparing the impact of the different features in environments. They still underscore the adaptability of CI concepts across various domains, and also emphasize the importance of utilizing CI practices from traditional fields to aid CI implementation in unfamiliar contexts. Besides these, although few papers discuss the setup of CI environments for beginners, several articles on student education focus on creating CI environments in academic settings that mimic industrial practices [18, 114]. These studies aim to bridge the gap between academic and professional experiences, enhancing students' understanding of agile development and testing practices. While such research mainly focuses on students' performance and ways to enhance their learning experiences, observing how they establish simulated industrial environments is valuable. Still, in these contexts, where students are newcomers to CI development, authors extensively consider the toolset or environment used in practice, including language and framework selection, test project choices, and organizing effective group collaboration for development.

D. Conclusion

The answer to RQ_1 is a reflection of the current state of contextual considerations within CI. In the majority of existing CI research, over half of the articles indicate that context plays a secondary role, typically influencing project selection based on factors such as popularity, size, or programming language for convenience. Still, one-third of the reviewed articles explicitly incorporate the impact of project context within their research scope, treating CI project factors as either direct or indirect elements of their studies. In the included literature, events related to open-source projects, such as MSR, place more emphasis on the importance of project context.

Figure 5 presents how the distribution of the different code categories has evolved over the years. The plot shows that the role of context in research on CI is slow, but steadily increasing. The distributions also show that researchers have slightly shifted their attention more towards organizational and process-related aspects. Future research should explore the effect of context on experimental variables, which can potentially lead to confounding factors that impact the internal validity of the research.

V. Which contextual factors get discussed in prior studies? (RQ_2)

The results of our investigation of related work provide a comprehensive view of the structure and content of a

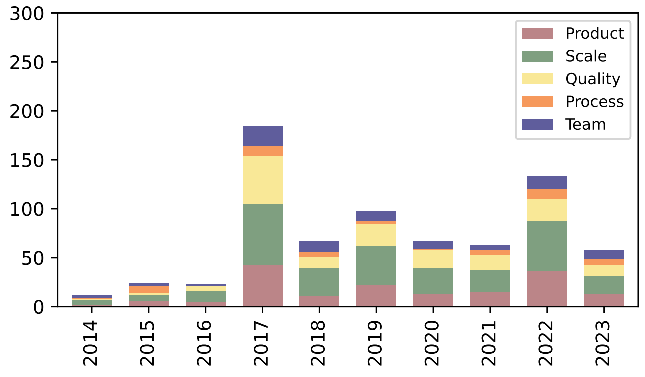

Fig. 5. Codes Distribution by Year

contextual framework. The following discussion of the new taxonomy will illustrate how each branch has been identified and discussed in previous work, which will also aid future research in exploring the impact of specific factors on CI practices. After axial coding and the categorization through our open coding rules, our coding vocabulary consists of a total of 343 unique codes, for which we could find a total count of 781 code instances across all reviewed papers. Figure 6 shows the resulting hierarchy, which focuses on the most relevant entries. It contains all elements in the first and second level. We also include the three most relevant leaves of the third level, but filter less frequent categories or concrete codes, to keep the taxonomy readable. The full, unfiltered taxonomy that includes all codes can be found in our online appendix [62]. The annotated numbers indicate the sum of all aggregated codes. As one paper can include multiple codes in one category, the count can be higher than the number of papers reviewed.

The numbers in the taxonomy represent the number of occurrences of the corresponding codes in the surveyed papers, illustrate the level of attention given to specific factors across different studies, providing a clear view of the more frequently discussed contextual factor, offer reference examples for various types and scopes of research.

In the following, we will focus on explaining the specific branches of the taxonomy. Future research should pay closer attention to identifying the specific impacts of relevant factors. We will introduce the contents of the taxonomy in detail and dedicate a subsection to each of the five top categories, in which we will explain the meaning of each dimension and introduce an exemplary selection of papers that fall into the category. We then discuss the results and provide an answer to RQ_2 in the conclusion.

A. Scale

Software scale is highly considered in observation, primarily because the measurability of a software project's size is a commonly used indicator of its developmental stage and data size. Our classification of scale comprises three parts: *size*, *popularity*, and *agility*, each offering different dimensions to the project's developmental status. While size reflects the project's code-level magnitude, popularity gauges its influence within the OSS community, and agility shows its level of activity. During discussions, we debated whether to categorize *"size"* separately but decided to abandon it. We realized that defining *"size"* separately was too narrow and we believe that *"scale"* included various aspects of a software's development stages. As such, we included other aspects alongside traditional size metrics.

Size. As the most direct measure of software, size is a context that has received more attention in CI research, due to its measurability and potential as substantial data samples, many studies filter out small and *"toy"* projects when conducting analyses. Additionally, size is considered a contextual factor influencing CI setup and implementation difficulty. Some studies have mentioned that different project sizes imply varying capabilities, which impact the effectiveness of CI pipeline construction and the allocation of resources [47, 57, 64]. Within the category of size, we have summarized three subcategories.

The first subcategory is the overall size of the project, encompassing aspects like the project repository size, which includes parameters such as KB size, lines of code, and the number of commits [48, 80, 124, 129]. Scholars often have diverse criteria for determining the size of a project. Commonly, research papers utilize lines of code, and some researchers also consider the project's commit counts [125, 142] or the number of files [46], as a criterion for size. In discussions involving card sorting, we also argued that age alone cannot entirely represent the scale of a project. Although we acknowledge that team size can also serve as evidence of project size and complexity, for the sake of maintaining consistency in the *"team"* category, we have chosen to align this element with the team-related aspects.

The second subcategory involves the complexity of the project, covering code-level complexity such as cyclomatic complexity, the number of methods, and the number of classes [1, 14]. It extends to the complexity of the project's structure and dependencies, such as the number of components/dependencies [33, 123] or third-party components [126]. The final subcategory of complexity related to different project phases includes considerations such as static code analysis [34], automated testing [57], deployment phases [108], compatibility of configuration files [48, 133], and integration levels with platforms like GitHub [52]. It also considers whether the project involves physical hardware, such as simulators [118, 134] or other external tools [122]. We introduced complexity as a supplementary measure to project size because the dimension of code lines alone is too singular to comprehensively simulate the project's size. These different dimensions indicate the maturity level of the current project, reflecting the costs associated with establishing or improving CI on this foundation.

Popularity. Popularity itself is also a popular contextual factor in CI research, with many articles utilizing it as one of the criteria for filtering samples to ensure a representative and sufficiently large dataset for the experiment. In OSS, popularity is primarily measured by the number of forks, stargazers,

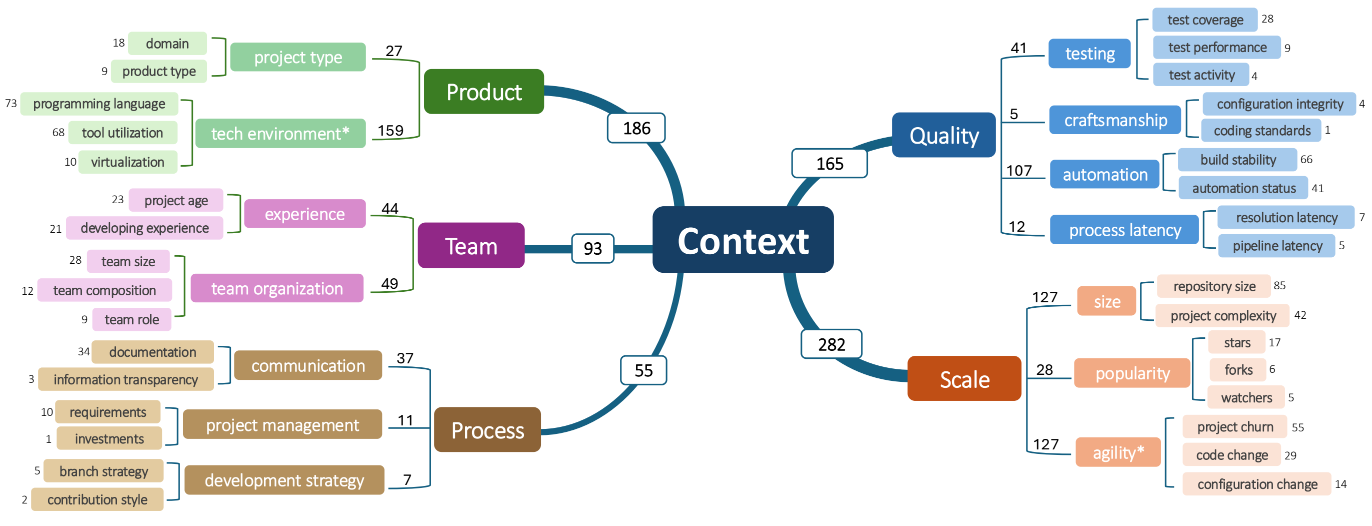

Fig. 6. Taxonomy (Numbers represent prevalence, categories with * in the name are not fully displayed)

and watchers [24] [83] [9] [137]. The metrics commonly used to measure popularity in OSS primarily include these three, but we believe this dimension can be further expanded, despite some practical limitations imposed by platforms. For instance, associating downloads with OSS projects is also an important factor in demonstrating their influence. We categorize popularity under the scale category because we believe that the level of attention an open-source project receives also reflects its visibility and exposure within the community, representing the project's magnitude of influence through another dimension.

Agility. The dynamic nature of OSS projects is also a focal point of CI studies, as depicted in Figure 6. It reflects project evolution, aiding scholars in identifying improvement points from historical data. Additionally, they also provide researchers with frequently updated data, making them valuable for research purposes.

Within this category, agility included two primary dimensions: the first is related to the project itself, encompassing the churn occurring across various areas within a specified period; the second dimension focuses on developers' contributions to different project aspects within a defined timeframe.

In project churn, we primarily observed dynamics across various aspects. Firstly, there are overall development dynamics in the codebase, focusing on the quantity and frequency of changes, such as pull requests [10, 12, 92], commits [48, 125, 129], builds [139], source codelines, and source files [64, 66, 135], among others. Secondly, updates in configuration files, including changes to CI configuration files over a certain period [46, 58], or build configuration/scripts [12, 64]. Thirdly, there are quality developments mainly involving changes in test line numbers and test suite cases [80]. The final category is deployment development, which includes the number of project versions, release frequencies, and deployment frequencies [57, 124]. This classification identifies the most code, not only because dynamic changes in projects are fascinating for finding patterns but also due to the diverse needs of various project study topics, leading to the development of different types of data.

In developers' contributions, we identified three types. Firstly, there is the average performance of developers, mainly reflected in their average commits [48]. Secondly, some projects focus on the performance of core developers, such as their contribution to core development tasks [47]. Lastly, there is the contribution of project personnel to maintenance work, reflected in configuration changes [49, 135] or refactoring frequency [120]. This focus reflects the project's emphasis on contributions from different types of developers, understanding the role of different developer types in fostering agility could lead to new project management and contributor engagement strategies.

Overall, a project's agility reflects the speed of updates and iterations over time, offering insights into its adaptability and responsiveness. Agility in this context can be seen as a reflection of the project's overall health and dynamism, where higher agility can correlate with a project's ability to evolve and maintain relevance in a rapidly changing technological landscape, but also indicates the potential lower code quality. Research could explore how agility correlates with project success metrics, such as response to trends, security issues, or user feedback.

B. *Product*

The term *product* in CI typically refers to the software application or system under development. It serves as a fundamental concept in open-source software development, with its technical nature and domain being the main aspects. Just as a seed determines the growth of a tree, the product background influences the overarching branches and fruits of the entire development process.

Within this classification, we summarized two main categories. The first is *project type*, referring to the domain and application type of the software. The second category is *tech environment*, including the entire technological ecosystem within OSS. This involves the selection and usage of tools, and these choices are mainly influenced by the *project type* at the core of development.

Project type. The project domain tends to be a significant influencing factor, especially when considering non-technical contexts. While it may not always be treated as a variable in the analysis of results, some research papers acknowledge the domain of OSS when understanding project backgrounds. However, in many studies, the product background of the project is not explicitly taken into consideration.

The project type mainly comprises two parts. Firstly, the product type refers to the final delivered software type, indicating the kind of software being produced, such as Android apps [85] or embedded systems [123], and so on. These considerations often influence the selection of the technological environment. The second aspect is the software domain. CI's application has extended beyond traditional software practices to include various domains. Traditional industries also embrace technological innovations to stay current, as seen in areas like Cyber-Physical Systems [134] and highly configurable software systems [94]. However, as mentioned in some papers, there is currently a lack of sufficient scientific support, especially when dealing with embedded software that involves hardware complexities or situations with too many stakeholders [118]. In some studies, even when the project domain is not considered in the study's direction, researchers often mention or pay attention to the various domains of the projects when analyzing their backgrounds. This underscores the growing interest in understanding project types.

Technology environments. As depicted in Figure 6, technology environments gained the highest interest in the previous study, since technological environments include all technical aspects associated with the project and are heavily influenced by the project type, and this type of data is also easier to extract and observe. Within this category, programming languages, tool usage, and project environments are frequently mentioned.

Many research studies use programming languages as a convenient filter for selecting relevant projects because the attributes of programming languages often impact the selection of technology tools or other techniques. Certain programming languages are often chosen due to their popularity or availability in specific datasets [66, 92, 105], which means popular languages with mature frameworks in testing or build and tool options provide researchers with abundant data, contributing to the heightened focus on these languages [10, 35, 142].

Moreover, tool usage is also important, as many development tools are assisting various steps in the automation pipeline. Tools such as test frameworks [39, 57], build tools [10], [120], bots [37], static analysis tools, communication platforms, and version control systems [110] are all subjects of interest in development.

Additionally, as an integral part of the entire development process, the development and build environment within a project is also a component of the technical ecosystem. This encompasses considerations such as whether the project utilizes cloud infrastructure like AWS [80, 114], containerization usage [133], and the existence of distinct environments for building, testing, and development purposes [34].

C. Quality

As the most significant benefit and process assurance in the CI pipeline, we categorize *"quality"* as a key aspect. It represents the assurance of a project's health level and effectiveness. Within this category, we summarize four aspects: Testing, Craftsmanship, Automation, and Process Latency, to describe the dimensions of quality from various stages of the pipeline.

Testing. Testing plays an important role in project development, with numerous articles addressing its role within CI processes. We have observed a heightened emphasis on testing-related code in two key categories: test coverage and testing performance, they represent key areas of interest and research in software testing.

Test coverage refers to the extent to which a software application's source code is tested by its automated test suite, i.e., the percentage of code lines, branches, statements, or conditions that are executed. A higher proportion of test code lines [23, 71] or the test cases [45] usually indicates the higher test coverage, and the weight on unit, integration, or regression tests [38] also underscores varying degrees of quality assurance within the codebase.

Moreover, testing performance includes metrics such as the number of tests run, their success or failure rates [67, 95], execution time [82, 129], and frequency [142] These metrics provide indicators of code quality and also reflect the developers' sensitivity to project quality.

Craftsmanship. While craftsmanship may not be a commonly referenced term in research due to its subjective nature, it represents the emotional investment and skill level of developers. Despite its challenging measurability, we still believe that certain aspects of code can represent this dimension, which we consider its impact on project quality.

Craftsmanship in code can be categorized into two key aspects. Firstly, the adherence to coding standards reflects the code's conformity to established guidelines [135]. Secondly, the precision of files in configurations and processes, particularly notable in legacy projects where configuration file updates are frequent [56, 100, 138]. Although factors like *Clean Code* [63], conciseness of build logs, or comment clarity were not explicitly discussed, we still believe they demonstrate a specific mindset of the respective developers.

Automation. Figure 6 shows that automation was often mentioned in the surveyed literature, which is reasonable given our focus on CI research. Within this classification, we found it includes several key aspects: the overall automation level of projects and building practices and stability.

The overall automation level of a project encompasses the evolution and degree of change in automation tools, such as the usage transition in CI tools, differing from previous tool discussions that mainly refer to project usage preferences. Specifically, it includes the evolution and extent of usage changes in CI tools [52, 97]. On the other hand, build stability encompasses various factors related to the outcome of builds, such as success or failure rates [44, 66], as well as the employed build strategies. For instance, researchers have shown interest in the timing of builds, whether they occur during the day or night, weekdays [48, 64, 136], etc. This provides insights into the robustness and reliability of the build process within the project.

We categorize these two as automation subcomponents because they encompass the evolution of automation tools and the stability of project builds.

Process latency. In the related work, we explored the overall positive impact of CI on development processes. Several studies have also investigated how CI may contribute to process delays. The delays from CI implementation or feedback do not always imply negative effects, but monitoring these delays and feedback speed provides perspectives about developers' habits and response efficiency.

In this category, we identified two primary types of latencies. The first involves feedback delays in pipeline developer activities, focusing on pull requests, issues, and other developer actions. Current collaborative platforms include features such as automated testing, commenting, and code review triggered by pull request submissions, interactions among stakeholders within these platforms also affect the processing delay of pull requests [12, 13, 131]. The second type of latency concerns the recovery time during project process failures, including the interval between a build failure and the next successful build, or the time taken to resolve tool-generated warnings [135]. The speed and waiting time in addressing these failures reflect the developers' dedication to the project. Analyzing these recovery times offers insights into development process efficiency and team responsiveness to issues encountered during Continuous Integration CI.

D. Team

While technology drives software innovation, software development remains a human-driven social activity, so we have abstracted the *"team"* category, focusing primarily on two aspects. Firstly, *"team organization"* includes personnel structure, roles, physical presence, and organizational hierarchy. Secondly, developer experience, as another part, reflects programming and project expertise, tool familiarity, and the ability to establish a mature CI process.

Within CI literature, the importance of individuals is consistently highlighted. Margan [90] argues that a successful OSS project requires an independent and influential leader, responsible for decision-making, project management, and supervision. While quantifying the alignment of these decisions with market preferences in project strategy may be challenging, OSS projects still offer code-related indicators to evaluate the technical expertise and experience of team members.

Team Organization. In OSS projects, team organization includes team size, team role and team composition. In terms of team size, aside from the total number of developers [110, 124] and the number of stakeholders [19] involved in the project, some studies specifically focus on the size of core developers [12, 47] or the Travis CI team [89]. Team composition refers to the roles of different team members [74]or physical distance [109, 136]. Some research also highlights the importance of having stakeholders or an independent testing team [39] within the overall team structure. In our discussion, we have not found any articles addressing the influence of spoken language, but we think that research in this area would be intriguing.

Experience. In the realm of open-source projects, quantifying developers' experience is challenging. It is difficult to capture all developer actions comprehensively due to limitations in accessing their contributions to confidential or proprietary projects. However, we can still model developers' experience levels in open-source projects based on factors such as their activity level, project involvement, and tool proficiency. Within the code we have obtained, we categorize developers' experience into tool proficiency [126], programming expertise [96], and project experience [1], mainly calculated by their follower count, past development activities, and the frequency and duration of their contributions. Besides, project age is also considered in context since it is related to understanding project development. While developer experience is a topic that has not been extensively explored in research, further investigation is needed to fully understand its impact.

E. Process

While the term *"process"* can encompass all activities, within the context of CI and this study, we specifically define it as process management. This includes a holistic view of how to manage different components that work together within the CI process. This category includes three key elements: communication, development management, and project management. These aspects go beyond coding tasks, providing insights into the entire ecosystem of interactions that shape workflow efficiency.

Communication. Scholars emphasize the importance of information sharing levels among developers and stakeholders in agile development [118]. Communication frequency, conflict resolution, and potential errors can impact CI efficiency. Despite being underexplored due to data collection challenges in open-source projects, communication is important in software development as a human-driven activity [37].

In open-source projects, communication is mostly conveyed through textual content throughout the development process. This includes documentation in pull/merge requests [119], features/issues [124], commits [105], and comments [2, 101]. Documentation instructions, such as completeness level in README files [37], are crucial for new contributors, representing the degree of information exposure to individuals involved.

Moreover, communication encounters increased challenges during business transformations [118]. The shift of traditional manufacturers towards a more software-centric model is particularly challenging, as these industries, primarily reliant on human resources, struggle to integrate numerous stakeholders into the technological era. Despite efforts towards integration, the software industry's structure remains deeply rooted in technological innovation, highlighting a gap in the conversion between human and technological resources.

Project management. Measuring project management in OSS projects is challenging, yet we include this category due to its growing importance. We identify codes related to human-related management, particularly concerning the project's plans and visions. This includes aspects such as time constraints [18, 114], budgets or cost [94], and different requirements type [93, 130]. These factors indicate that in development, there are certain constraints in terms of budget or time allocated from the higher management level, which can influence the pace of development and the formulation of plans, which may also contribute to variations in project quality and outcomes.

Development. Regarding development management, this includes aspects such as the development branch's preference [9, 53] or the contribution type [64] made by developers. Developers may employ different collaboration methods, often opting for various development or build branches [136]. The choice between utilizing the 'master' branch or alternative branches for development varies [64]. The number of branches within a project also reflects this [49]. Although we have not identified more aspects, these workflow habits and collaboration patterns within the project reflect its preferences and development practices. whether these habits have an impact on code development quality, efficiency, and other factors remains further research.

F. Conclusion

When studying the prevalence of the taxonomy categories in Figure 6, we can see that all top-level elements have a substantial prevalence. The most common dimension is *Scale* with 282 (36%) instances, followed by *Product* (186, 24%) and *Quality* (165, 21%). This is unsurprising. The corresponding codes are technical and can be extracted from repositories, which makes them easier to study. The remaining dimensions, *Team* (93, 12%) and *Process* (55, 7%), are more difficult to measure automatically and require a more qualitative investigation of projects. While both are still represented in a wide range of papers, we find fewer examples.

This picture already changes on the second taxonomy level and the prevalence differs substantially, within and across categories. We can find several codes that have been mentioned in fewer than 20 papers. The most common categories are *tech environment* (159, 20%), *size* (127, 16%), *agility* (127, 16%), and *automation* (107, 13%). All of these can be directly mined from repositories, which adds further support to our hypothesis that the availability of data and convenient access is a major factor when investigating relevant context.

To answer RQ$_2$, our review of related work has highlighted various aspects that have been explored in previous work, which can also assist developers and researchers in identifying more appropriate frameworks. It is evident that certain key factors have been extensively studied, but further investigation is needed to determine whether less popular contextual factors also warrant greater attention.

VI. Do practitioners agree with the novel taxonomy of contextual factors? (RQ$_3$)

We have designed a survey to validate our taxonomy of project context with practitioners and gain additional feedback. We have received a total of 34 responses. In this section, we will present the results. We will first investigate the personal perspectives on CI that our participants have provided in our open-ended questions. We will then investigate to which extent the participants agree with our taxonomy for CI context.

Demographics. Among the selected 34 participants, 22 (64.7%) are employed in companies with development teams exceeding 100 employees. All participants hold at least a bachelor's degree. Most participants (31/91.1%) associate their work with key development fields, including software back-end/front-end development, testing, network management, data engineering, embedded systems engineering, and software architecture, with 27 (79.4%) reporting *frequent use of CI (on a monthly basis)* or *constant integration of CI into their daily or weekly workflow*. These demographic information make us confident that the participants possess the required experience in CI and that they qualify for our survey.

Personal Perspectives. Before priming the participants with our taxonomy, we tested our motivating assumption and asked *"Do you believe there exists one approach for implementing Continuous Integration (CI) that applies to all projects?"* to investigate their general opinion on CI implementation methods. This question had no predetermined *"correct"* answer, instead, we collected open thoughts on this topic from the participants.

Our survey results reveal a divide among participants, a majority believing that no universal CI method exists (No: 21, Yes: 13). Participants who answered *"No"* emphasized the personalized characteristics of projects, such as *programming language*, *technology stack*, *team size*, and *management processes*, some respondents also point out the importance of *different business scenarios* in CI practices, arguing that actual projects often require customized implementations to meet unique demands. These perspectives collectively indicate that a one-size-fits-all solution is impractical in practice, as project diversity and complexity inevitably introduce exceptions.

On the other hand, participants who answered *"Yes"* stressed the core principles of CI, to automate the integration of code changes. These respondents argued that, despite variations in project contexts, like *project environments* and *business requirements*, the fundamental concept of CI remains

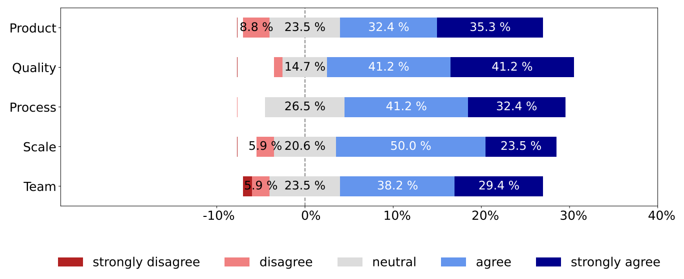

Fig. 7. Agreement Rate on Taxonomy

consistent across different types of projects. This consistency implies that CI implementation should revolve around its core principles and maintain a certain level of universality.

Overall, these two perspectives collectively illustrate the complex nature of CI in real-world projects: while the *core goal of CI* is frequent delivery through automation, the *implementation pathways* are influenced by variations in project contexts. The purpose of this survey question was not to find a single *correct answer*, but rather to gain deeper insights into our participants views on CI methodologies across diverse project settings. This divide highlights the complexity of CI practices in different project environments.

Taxonomy Validation. The results of our questions regarding the agreement to our taxonomy can be found in Figure 7. In the survey, we presented the specific content of each higher-level branch and asked for suggestions on potential modifications. We measure participant agreement with the classification and content of each branch through a Likert scale. Among 34 validated results, the findings indicate a strong level of agreement with our final taxonomy. The results are very similar for all dimensions and every aspect of the taxonomy received more than 60% agreement, with *quality* slightly higher and *team* and *product* slightly lower. On average, a dimension has received with an average agreement rate of 72.9%. The high level of agreement makes us confident that the taxonomy adequately represents expert opinions. Approximately 21.7% of respondents had a neutral reaction, likely, because they could either not relate to the relevance of the dimensions or when they expected a low effect in practice.

A small fraction of 5.2% disagreed or strongly disagreed with certain aspects of the taxonomy. We have reviewed their open feedback through to understand why participants disagreed. While we could not identify gaps in the taxonomy, we found several explanations for disagreement.

Disagreements. Not all participants who disagreed made use of the option to elaborate on their responses. From the explanations that were actually provided, we found a comment that disagreed with the *team* aspect of the taxonomy. The comment said that CI should be easy for team members to adopt and should remain accessible regardless of expertise level. We agree from the user perspective, but we believe that the comment does not fully consider the setup and maintenance of CI, which is more difficult and often done by a more experienced developer, so we decided to reject the criticism.

One person raised concerns about changes or removals: an increase in development manpower can complicate CI implementation, as it leads to parallel projects and issues, which in turn expands the scope of regression testing and making the overall process more challenging to manage. We generally agree with the argument, but since team factors are already addressed in another category, we have decided to maintain the existing hierarchy.

Regarding *scale*, one respondent explains that scale should not inherently impact CI, but rather CI systems should adapt to changes in scale. We find this to be a philosophical *chicken-and-egg* discussion. While we generally agree with the comment for projects that naturally grow, not every project starts small, so we decided to keep the option in the taxonomy.

Requested Additions. In addition, we also asked respondents to suggest any aspects they felt were missing from the taxonomy. The few answers to the open questions suggested additions such as *"Standard and culture"*, *"cost"*, and *"The heterogeneity of software"*. As elaborated before, participants did not see the full taxonomy to avoid information overload. As such, they were not aware that their suggestions had already been part of the taxonomy, but were included on deeper levels. As we were able to easily identify suitable taxonomy categories for all suggestions, we concluded that the taxonomy is complete.

A. Conclusion

The responses from our practitioner survey have showed a high level of agreement. Given the overall small number of disagreements and the ease with which we could counter-argue against the raised issues, we conclude that the taxonomy is valid and does not have major issues or limitations. This makes us confident about the completeness of the taxonomy.

Despite the positive results, it becomes clear though that there is still a gap between academic research and the reality faced by practitioners. The open survey answers gave us diverse perspectives across different fields and industries. Despite the widespread adoption of CI, there seems to be a lack of standardized guidelines tailored to different organizational structures and project requirements. One possible explanation could be the lack of a unified terminology around project context that would enable a better discourse. Many respondents also emphasized the importance of considering non-technical factors like business requirements, management and human factors, or remote work.

Our results show that there is ample need for further exploration of the different use cases in future work and we believe that an exhaustive taxonomy is a crucial ingredient and a first step towards overcoming the gap.

VII. Discussion

Beyond the immediate observations and findings that we have formulated alongside the research questions, our results warrant a deeper discussion of several aspects.

A. Immediate Impact and Actionable Insights

The results of this work, especially the creation of the taxonomy, have the potential to shape future CI research.

Improve Future Research. The holistic taxonomy provides a systematic framework for describing project context. Many studies on similar topics tend to adopt context factors to at least some degree within a specific scope. While such choices can be beneficial for the concrete research question, introducing specific contextual factors affects comparability to previous studies. Future work should investigate the effects that subtle changes in project context might have on conclusions of previous studies. The systematic framework that our taxonomy provides will support such an endeavor, it allows researchers to identify gaps, uncover missing contextual factors in their experimental design, and reduce the number of deviations and corner cases that were previously unexplainable.

Comparison to Existing Context Structures. Existing research in software engineering has already explored the role of context. While not all previous works have presented taxonomies, a comparison the the taxonomy in CI research is warranted.

Kirk [75] proposed a six-dimensional context description: *Who* (culture), *Where* (space and time), *What* (product constraints), *When* (product life-cycle stage), *How* (engagement constraints), *Why* (organizational drivers). This framework has a goal similar to our work: examining the impact of context on SE. The concrete implementations differ though as they focus on different levels, best explained through the context terminology of Dybå et al. [36]. Kirk focuses more on the *macro-context* and includes broad environmental and background factors. In contrast, our work emphasizes more the *discrete context*, targeting specific, measurable context variables. We also have a focus on CI and wanted to develop a taxonomy that is actionable and quantifiable. Both our works share certain similarities, like primarily involve developers, teams, organizations, projects, and the surrounding development environment. This aligns with our expectations as CI is a subset of software engineering. Some of our aspects could be mapped to the other work, like *time allocation*, *geographical distribution*, and *investment levels*, which are reflected in *developer cognition* and *organizational culture*. However, some of our aspects are new and directly tied to software quality, such as *test coverage* and *build stability*, which shows its importance in the CI domain.

Avoid Overclaims. For some studies, especially those centered on algorithmic research for builds or testing, project context is often merely serving as a background setting, if considered at all. This raises the valid question of whether a consideration of the project context is even necessary in these types of studies. We acknowledge that context might not be the first focus for novel approaches, but we still think that empirical evaluations should always be transparent about the test setting or any assumptions that must hold. This allows us to put results into perspective and avoids over-claims for generalizability.

Terminology Alignment. We have observed an inconsistent use of terminology when related work describe project context. A few contextual factors such as *project size*, *project age*, or *popularity* are commonly used for research purposes, however, many other factors exist that we found under different names. Furthermore, we have seen a lack of uniformity in the definition and measurement of similar metrics across papers, e.g., *project size* can be quantified in terms of lines of code, number of files/commits, or size on disk. Similarly, *popularity* could refer to stargazers, forks, or dependent projects. Although these varied measurements all capture similar concepts, the differences affect the result consistency and prevent a comparative discussion about the impact of said factors.

Paper Selection and Taxonomy Evolution. We have selected a representative set of high-quality publications from reputable software engineering venues, and applied keyword-based searches to construct the current CI context taxonomy. As the primary objective was not to collect an exhaustive list of papers, but rather to identify representative contextual factors, we are confident that the selected sample is sufficient to achieve our research goal. This has been confirmed by our converging vocabulary and through the practitioner survey, which confirmed the general relevance and validity of the identified contextual factors. Although software development technologies will continue to evolve, new development paradigms will emerge, and the distribution of contextual factors may shift, we believe that the *highest levels* of the taxonomy will remain stable, as they are independent of specific tool and technologies. Nevertheless, this taxonomy is evolving. Future research can build upon the existing hierarchical structure and extend it with new contextual factors driven by technological changes. Similarly, practitioners should adapt the relevant contextual dimensions to their specific technological environments.

B. Reflection and Future Work

Some parts of our study reveal new research opportunities or warrant a reflection on certain aspects.

Unclear Impact. The survey has revealed a wide range of context factors that have not been considered in many studies so far. Furthermore, our taxonomy was suffering from an imbalance, especially in the lower branches of the hierarchy. It remains unclear why some of the context features are infrequently used. Possible explanations are lack of awareness, difficulty in measurement, or lack of actual impact in practice. Future work is needed, for example, studying the practical impact of certain context factors on concrete study results.

Data Availability. Project context consists of a wide range of factors. Many are technical, which are usually easier to acquire, but many factors are human-centered, like the *team composition* or *communication strategies*, which makes them much more difficult to acquire at scale. If we expect future research to pick up context factors, it becomes necessary to manually curate datasets that grant easy access to previously understudied aspects. We postulate that data must be readily available to manifest project context in more future studies.

Definition Boundary. In our attempt to classify codes and construct a taxonomy structure, we defined context as the *input factors* that influence the building and enhancement of a CI pipeline, the corresponding effects that a project can enjoy as the result would be the *output*. However, during our literature survey, we realized that the boundary between input and output is not always clear-cut, many metrics could be considered as both. For example, *release frequency* could be a process metric that is specified by the product manager (e.g., bi-weekly releases), which will influence the coding practices in the project. However, it could also be considered as a *result* that reflects the efficiency of the adopted CI practices. In this paper, we acknowledge that some factors cannot be isolated and we decided to be conservative and include these factors in our taxonomy as an input factor. Future research should revisit this question to refine our definition attempt.

Measuring Success. The plethora of open-source projects that are available in public repositories makes it possible to collect an endless number of example projects. Differences between projects could be explained by different requirements, but it could also be the case that one team lacks experience and is not following best practices. It remains unclear how to compare the success or maturity of projects to differentiate both cases. When project context is the *input* to a black box that represents CI, we still need to establish means to model success or maturity of software projects as the *output* of CI.

Studying Project Evolution. One important factor that is notably absent in the list of the investigated context factors is time. The assumption is that many projects start small and project maintainers learn over time. Projects evolve from the bare minimum that is required to get started to mature projects, which automate the major pain points of the integration process. This is of course not true for all projects, some projects remain in these initial stages, while others are created by experienced developers and they start with advanced setups and processes in place. The average project will likely take wrong turns in the development, but will slowly but steadily improve over time. Future work needs to understand these evolution paths, as new spare-time projects will need different support than a company-backed project that is developed with full-time developers. We need to investigate the impact of evolution both from the perspective of individual projects, but also from the field as a whole, as new technologies might replace former best practices.

Practitioner's Dilemma. The survey revealed an interesting observation: our open question about the possibility of a one-size-fits-all approach to CI split our participants almost evenly in half. We investigated the responses and found that the opposing majority pointed at different project contexts as proof that this idea is infeasible. Most interestingly, even the proponents acknowledged the inherent complexities that are introduced by different project contexts. However, they argued that CI is so common and important that a universal method *should* exist to address the needs of all projects.

This observation indicates a dilemma. Even though the survey included participants from very different domains, most individual developers do not engage simultaneously in projects with radically different contexts and, as such, only have limited experience with applying CI in multiple contexts. For example, a data scientist has likely little experience in the development of Android apps or web applications. Practitioners are *specialists* with in-depth knowledge and a thorough understanding of CI, but usually only in their very specific context. This can make it challenging to adopt a neutral third-person perspective or consider alternative viewpoints. On the other hand, researchers in the field of CI present *generalists*. They lack the same depth of experience, but they are able to see the broader picture that practitioners might be lacking.

Individually, both groups might not be able to solve the challenge of project context, solutions will always be too specific or too shallow. Future work needs to investigate how to join both worlds and leverage experiences to overcome the current gaps in our understanding of CI adoption.

C. Threats to validity

We have identified several factors that threaten the internal or external validity of our presented results. In the following, we will assess the threat and present out mitigation strategies.

Data Selection. To construct the dataset for our study, we primarily searched for prior work that is related to *Continuous Integration*. As such, the resulting articles inherently contain more discussions about CI, process automation, and quality, which could create a bias and overrepresentation of these topics in our findings. To mitigate this, We investigated further search terms like *DevOps* to widen the results, but ultimately decided to drop them, as we could not find additional material that would add to our results. We believe that our data collection strategy was reasonable for our goal to identify contextual factors that affect CI processes.

A second threat is to limit the data selection to the most prestigious venues in software engineering and not to apply *snowballing* to identify further works from other venues. Our goal was not to curate an exhaustive paper catalog, but to extract a representative list of codes. As the list of codes has converged and the practitioner responses to our taxonomy were positive, we strongly believe that we have reached our goal.

Biased Information Source. This paper tried to study the contextual factors that might affect the setup and maintenance of CI projects. To achieve this goal, however, only scientific publications have been studied, which might limit the insights that this study can reveal. We think that the risk of this is low, as many included papers were empirical studies on open-source software, which reflect practical experience. In addition, our validation of the taxonomy among practitioners revealed general agreement. Despite these arguments, we believe it would be worthwhile to study the different taxonomy dimensions in open-source projects and investigate a potential difference in their prevalence.

Manual Classification. There are two main threats to validity in our manual classification process. First, the potential bias introduced by researchers' personal interpretations during open coding. The outcome could be influenced by individual understanding and preconceptions. To mitigate this, we conducted multiple discussions to determine the standard and procedure for final code inclusion. Disagreements were addressed by involving a third volunteer in the discussions. Second, open coding on scholarly articles is challenging, and there is always the risk of misunderstanding or omission of codes, when reading highly specialized articles. We cannot avoid inaccuracies in the coding process, but we think that this threat to the overall results is low, as the setup of our methodology leads to a saturated list of codes and a stable hierarchy. This makes us confident that our results are representative and that a repetition of our survey by other researchers would come to a comparable result.

Practitioner Survey. The results of our practitioner survey are based on only 34 responses. While this may seem like a small number on a first glance, we would like to highlight that these are only the high-quality responses that remained after we filtered the 106 original responses with strict quality criteria. When considering the demographics of the participants, the time they spent on the survey, and the extent, to which they have provided open answers, we are confident that the responses are representative. Our study would have been strengthened by asking participants to assess whether each factor should be considered a contextual factor. However, we were concerned about the complexity and duration of the survey and that further extensions would have reduced the quality of feedback or increased the drop-out rates. As a compromise, we therefore allowed participants to suggest missing or incorrectly classified factors in open-ended questions, but we could only find minor comments on our results in the responses. We avoided a potential experimenter bias, caused by the self-selection of participants, by being vague in the explanation of our role in the research and by receiving the majority of responses from a public audience on platforms like Reddit.

VIII. Summary

Continuous Integration has been proven to play a crucial role in enhancing the practical productivity of software development. Understanding the context of the project enables developers and researchers to comprehend the specific circumstances and limitations of projects. This understanding facilitates the selection of the most appropriate strategies to improve CI pipelines and benefit other projects that share similar contexts. Despite the importance of project context, current research on CI often overlooks or simplifies the impact of contextual factors. In some studies, context is treated as secondary background information. To address this gap, our study introduces a comprehensive taxonomy of contextual factors in CI, developed through a systematic literature review. Employing open coding and axial coding methods, we identified and analyzed contextual factors from existing literature that influence the implementation and maintenance of CI. These factors were then structured using a card-sorting methodology, resulting in a taxonomy encompassing five main dimensions: Product, Team, Quality, Scale, and Process. The taxonomy's validity was further reinforced through a practitioner survey. We aim to use this comprehensive taxonomy to raise awareness of the importance of context in CI-related research, while in future work, this taxonomy can also help to identify different CI contexts and provide more targeted guidance and optimization strategies for implementing CI best practices across various scenarios.

IX. Acknowledgements

We thank João Neto and Kendra Sartori for their valuable contributions to this research. We also thank all participants of our online survey. We are grateful to Andy Zaidman for his insightful comments and suggestions on our work, and to the anonymous reviewers of an earlier version of this paper for their constructive feedback.

References

[1] R. Abdalkareem, S. Mujahid, and E. Shihab. A machine learning approach to improve the detection of ci skip commits. In *Trans. Softw. Eng.*, 2020.
[2] R. Abdalkareem, S. Mujahid, E. Shihab, and J. Rilling. Which commits can be CI skipped? In *Trans. on Software Engineering*, 2019.
[3] M. Abdellatif, A. Shatnawi, H. Mili, N. Moha, G. El Boussaidi, G. Hecht, J. Privat, and Y.-G. Guéhéneuc. A taxonomy of service identification approaches for legacy software systems modernization. In *Journal of Systems and Software*, 2021.
[4] S. Adolph, P. Kruchten, and W. Hall. Reconciling perspectives: A grounded theory of how people manage the process of software development. *Journal of Systems and Softwa.*, 2012.
[5] B. A. Akinnuwesi, F.-M. Uzoka, S. O. Olabiyisi, E. O. Omidiora, and P. Fiddi. An empirical analysis of end-user participation in software development projects in a developing country context. *The Electronic Journal of Information Systems in Developing Countries*, 2013.
[6] Amazon Builders' Library/Clare Liguori. Automating safe, hands-off deployments. https://aws.amazon.com/builders-library/automating-safe-hands-off-deployments/, 2020. online.
[7] D. Avison and J. Pries-Heje. Flexible information systems development: Designing an appropriate methodology for different situations. In *9th Int. Conf. on Enterprise Information Systems*, 2008.
[8] M. Bajec, D. Vavpotič, and M. Krisper. Practice-driven approach for creating project-specific software development methods. *Information and Software technology*, 2007.
[9] S. Baltes, J. Knack, D. Anastasiou, R. Tymann, and S. Diehl. (No) influence of continuous integration on the commit activity in GitHub projects. In *Int. Workshop on Softw. Analytics*, 2018.
[10] M. Beller, G. Gousios, and A. Zaidman. Oops, My Tests Broke the Build: An Explorative Analysis of Travis CI with GitHub. In *Int. Conf. Mining Softw. Repositories*, 2017.
[11] M. Beller, G. Gousios, and A. Zaidman. TravisTorrent: Synthesizing Travis CI and GitHub for Full-Stack Research on Continuous Integration. In *Int. Conf. Mining Softw. Repositories*, 2017.
[12] J. H. Bernardo, D. A. da Costa, and U. Kulesza. Studying the impact of adopting continuous integration on the delivery time of pull requests. In *Int. Conf. Mining Softw. Repositories*, 2018.
[13] J. H. Bernardo, D. A. da Costa, U. Kulesza, and C. Treude. The impact of a continuous integration service on the delivery time of merged pull requests. In *Empirical Softw. Eng.*, 2023.
[14] A. Bertolino, A. Guerriero, B. Miranda, R. Pietrantuono, and S. Russo. Learning-to-rank vs ranking-to-learn: Strategies for regression testing in continuous integration. In *Int. Conf. Softw. Eng.*, 2020.
[15] D. F. Birks, W. Fernandez, N. Levina, and S. Nasirin. Grounded theory method in information systems research: its nature, diversity and opportunities. *European Journal of Information Systems*, 2013.
[16] B. Boehm. Get ready for agile methods, with care. *Computer*, 2002.

[17] L. J. Bourgeois III. Toward a method of middle-range theorizing. *Academy of management review*, 1979.

[18] J. Bowyer and J. Hughes. Assessing undergraduate experience of continuous integration and test-driven development. In *Int. Conf. Softw. Eng.*, 2006.

[19] M. Brandtner, S. C. Müller, P. Leitner, and H. C. Gall. SQA-Profiles: Rule-based activity profiles for Continuous Integration environments. In *Int. Conf. Softw. Analysis, Evolution and Reengineering*, 2015.

[20] L. Briand, D. Bianculli, S. Nejati, F. Pastore, and M. Sabetzadeh. The case for context-driven software engineering research: generalizability is overrated. *IEEE Software*, 2017.

[21] J. Campos, A. Arcuri, G. Fraser, and R. Abreu. Continuous test generation: Enhancing continuous integration with automated test generation. In *Int. Conf. Automated Softw. Eng.*, 2014.

[22] L. Carminati. Generalizability in qualitative research: A tale of two traditions. *Qualitative health research*, 2018.

[23] A. R. Chen, T.-H. P. Chen, and S. Wang. T-Evos: A Large-Scale Longitudinal Study on CI Test Execution and Failure. In *Trans. Softw. Eng.*, 2022.

[24] B. Chen, L. Chen, C. Zhang, and X. Peng. Buildfast: History-aware build outcome prediction for fast feedback and reduced cost in continuous integration. In *Int. Conf. Automated Softw. Eng.*, 2020.

[25] E. Chin. Redefining "context" in research on writing. *Written Communication*, 1994.

[26] P. Clarke and R. V. O'connor. The situational factors that affect the software development process: Towards a comprehensive reference framework. *Information and Softw. technology*, 2012.

[27] J. Cohen. A coefficient of agreement for nominal scales. In *Educational and psychological measurement*, 1960.

[28] G. Coleman and R. O'Connor. Using grounded theory to understand software process improvement: A study of Irish software product companies. *Information and Software Technology*, 2007.

[29] M. Cusumano, A. MacCormack, C. F. Kemerer, and B. Crandall. Software development worldwide: The state of the practice. *IEEE software*, 2003.

[30] A. Debbiche, M. Dienér, and R. Berntsson Svensson. Challenges when adopting continuous integration: A case study. In *15th Int. Conf. on Product-Focused Software Process Improvement*, 2014.

[31] V. Debroy, S. Miller, and L. Brimble. Building lean continuous integration and delivery pipelines by applying DevOps principles: a case study at Varidesk. In *Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2018.

[32] digital.ai. 16th State of Agile Report. https://digital.ai/resource-center/analyst-reports/state-of-agile-report/, 2022. online.

[33] S. Dösinger, R. Mordinyi, and S. Biffl. Communicating continuous integration servers for increasing effectiveness of automated testing. In *Int. Conf. Automated Softw. Eng.*, 2012.

[34] T. Durieux, R. Abreu, M. Monperrus, T. F. Bissyandé, and L. Cruz. An analysis of 35+ million jobs of Travis CI. In *Int. Conf. Softw. Maintenance and Evolution*, 2019.

[35] T. Durieux, C. Le Goues, M. Hilton, and R. Abreu. Empirical study of restarted and flaky builds on Travis CI. In *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020.

[36] T. Dybå, D. I. Sjøberg, and D. S. Cruzes. What works for whom, where, when, and why? On the role of context in Empirical Softw. Eng. In *Int. symposium on Empirical Softw. Eng. and measurement*, 2012.

[37] O. Elazhary, M.-A. Storey, N. A. Ernst, and E. Paradis. Adept: A socio-technical theory of continuous integration. In *Int. Conf. Softw. Eng. New Ideas and Emerging Results*, 2021.

[38] O. Elazhary, C. Werner, Z. S. Li, D. Lowlind, N. A. Ernst, and M.-A. Storey. Uncovering the benefits and challenges of continuous integration practices. *Trans. Softw. Eng.*, 2021.

[39] S. Elbaum, G. Rothermel, and J. Penix. Techniques for improving regression testing in continuous integration development environments. In *Int. Symp. Found. Softw. Eng.*, 2014.

[40] J. Fendler and H. Winschiers-Theophilus. Towards contextualised software engineering education: an African perspective. In *Proceedings of the 32nd ACM/IEEE Int. Conf. Softw. Eng.–volume 1*, 2010.

[41] B. Fitzgerald, G. Hartnett, and K. Conboy. Customising agile methods to software practices at Intel Shannon. *European Journal of Information Systems*, 2006.

[42] M. Fowler and M. Foemmel. Continuous integration, 2006.

[43] K. Gallaba. Improving the robustness and efficiency of continuous integration and deployment. In *Int. Conf. Softw. Maintenance and Evolution*, 2019.

[44] K. Gallaba, M. Lamothe, and S. McIntosh. Lessons from eight years of operational data from a continuous integration service: an exploratory case study of CircleCI. In *Int. Conf. Softw. Eng.*, 2022.

[45] K. Gallaba, C. Macho, M. Pinzger, and S. McIntosh. Noise and heterogeneity in historical build data: an empirical study of Travis CI. In *Int. Conf. Automated Softw. Eng.*, 2018.

[46] K. Gallaba and S. McIntosh. Use and misuse of continuous integration features: An empirical study of projects that (mis) use Travis CI. In *Trans. Softw. Eng.*, 2018.

[47] A. Gautam, S. Vishwasrao, and F. Servant. An Empirical Study of Activity, Popularity, Size, Testing, and Stability in Continuous Integration. In *Int. Conf. Mining Softw. Repositories*, 2017.

[48] T. A. Ghaleb, D. A. Da Costa, and Y. Zou. An empirical study of the long duration of continuous integration builds. In *Empirical Softw. Eng.*, 2019.

[49] T. A. Ghaleb, S. Hassan, and Y. Zou. Studying the interplay between the durations and breakages of continuous integration builds. In *Trans. Softw. Eng.*, 2022.

[50] B. Glaser and A. Strauss. *Discovery of grounded theory: Strategies for qualitative research*. Routledge, 2017.

[51] B. G. Glaser. Doing grounded theory: Issues and discussions, 1998.

[52] M. Golzadeh, A. Decan, and T. Mens. On the rise and fall of CI services in GitHub. In *Int. Conf. Softw. Analysis, Evolution and Reengineering*, 2022.

[53] Y. Gupta, Y. Khan, K. Gallaba, and S. McIntosh. The impact of the adoption of continuous integration on developer attraction and retention. In *Int. Conf. Mining Softw. Repositories*, 2017.

[54] A. Haghighatkhah, M. Mäntylä, M. Oivo, and P. Kuvaja. Test prioritization in continuous integration environments. In *Journal of Systems and Software*, 2018.

[55] J. G. Hall and L. Rapanotti. A design theory for software engineering. *Information and Softw. Technology*, 2017.

[56] F. Hassan. Tackling build failures in continuous integration. In *Int. Conf. Automated Softw. Eng.*, 2019.

[57] M. Hilton, N. Nelson, T. Tunnell, D. Marinov, and D. Dig. Trade-offs in continuous integration: assurance, security, and flexibility. In *Found. Softw. Eng.*, 2017.

[58] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig. Usage, costs, and benefits of continuous integration in open-source projects. In *Int. Conf. Automated Softw. Eng.*, 2016.

[59] R. Hoda. Socio-technical grounded theory for software engineering. *Trans. Softw. Eng.*, 2021.

[60] R. Hoda, P. Kruchten, J. Noble, and S. Marshall. Agility in context. In *Proceedings of the ACM international conference on Object oriented programming systems languages and applications*, 2010.

[61] R. Hoda and J. Noble. Becoming agile: a grounded theory of agile transitions in practice. In *Int. Conf. Softw. Eng.*, 2017.

[62] S. Huang and S. Proksch. Online Appendix: Contextual Factors in Continuous Integration. https://doi.org/10.5281/zenodo.11316810, 2024. Zenodo.

[63] D. Hutton. Clean code: a handbook of agile software craftsmanship. In *Kybernetes*. Emerald Group Publishing Limited, 2009.

[64] M. R. Islam and M. F. Zibran. Insights into Continuous Integration Build Failures. In *Int. Conf. Mining Softw. Repositories*, 2017.

[65] O. Javed, J. H. Dawes, M. Han, G. Franzoni, A. Pfeiffer, G. Reger, and W. Binder. PerfCI: a toolchain for automated performance testing during continuous integration of Python projects. In *Int. Conf. Automated Softw. Eng.*, 2020.

[66] X. Jin and F. Servant. A cost-efficient approach to building in continuous integration. In *Int. Conf. Softw. Eng.*, 2020.

[67] X. Jin and F. Servant. What helped, and what did not? An evaluation of the strategies to improve continuous integration. In *Int. Conf. Softw. Eng.*, 2021.

[68] X. Jin and F. Servant. Which builds are really safe to skip? Maximizing failure observation for build selection in continuous integration. In *Journal of Systems and Softw.*, 2022.

[69] G. Johns. The essential impact of context on -organizational behavior. *Academy of management review*, 2006.

[70] P. Johnson, P. Ralph, M. Goedicke, P.-W. Ng, K.-J. Stol, K. Smolander, I. Exman, and D. E. Perry. Report on the second SEMAT workshop on general theory of software engineering (GTSE 2013). *ACM SIGSOFT Softw. Eng. Notes*, 2013.

[71] E. Kauhanen, J. K. Nurminen, T. Mikkonen, and M. Pashkovskiy. Regression test selection tool for python in continuous integration process. In *Int. Conf. Softw. Analysis, Evolution and Reengineering*, 2021.

[72] J. Kendall. Axial coding and the grounded theory controversy. In *Western journal of nursing research*, 1999.

[73] S. H. Khandkar. Open coding. In *University of Calgary*, 2009.

[74] S. Kim, S. Park, J. Yun, and Y. Lee. Automated continuous integration of component-based software: An industrial experience. In *Int. Conf. Automated Softw. Eng.*, 2008.

[75] D. Kirk and S. G. MacDonell. Investigating a conceptual construct for software context. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, 2014.

[76] B. Kitchenham. Procedures for performing systematic reviews. In *Keele, UK, Keele University*. Citeseer, 2004.

[77] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman. Systematic literature reviews in software engineering–a systematic literature review. In *Information and software technology*. Elsevier, 2009.

[78] B. A. Kitchenham, T. Dyba, and M. Jorgensen. Evidence-based software engineering. In *Int. Conf. Softw. Eng.*, 2004.

[79] S. Kumari. Transforming CRM DevOps at Airbnb: A Powerful Framework for Continuous Delivery. https://medium.com/airbnb-engineering/transforming-crm-devops-at-airbnb-a-powerful-framework-for-continuous-delivery-c84a9c18032e, 2023. online.

[80] A. Labuschagne, L. Inozemtseva, and R. Holmes. Measuring the cost of regression testing in practice: A study of java projects using continuous integration. In *Proceedings of the 2017 11th joint meeting on foundations of software engineering*, 2017.

[81] E. Laukkanen, M. Paasivaara, and T. Arvonen. Stakeholder perceptions of the adoption of continuous integration–a case study. In *Agile Conf.*, 2015.

[82] J. Liang, S. Elbaum, and G. Rothermel. Redefining prioritization: continuous prioritization for continuous integration. In *Int. Conf. Softw. Eng.*, 2018.

[83] J. A. P. Lima and S. R. Vergilio. A multi-armed bandit approach for test case prioritization in continuous integration environments. In *Trans. Softw. Eng.*, 2020.

[84] M. Lindvall, V. Basili, B. Boehm, P. Costa, K. Dangle, F. Shull, R. Tesoriero, L. Williams, and M. Zelkowitz. Empirical findings in agile methods. In *XP/Agile Universe*, 2002.

[85] P. Liu, X. Sun, Y. Zhao, Y. Liu, J. Grundy, and L. Li. A first look at CI/CD adoptions in open-source android apps. In *Int. Conf. Automated Softw. Eng.*, 2022.

[86] G. C. LLC. 2022 State of DevOps Report. https://dora.dev/, 2022. online.

[87] A. MacCormack, W. Crandall, P. Henderson, and P. Toft. Do you need a new product-development strategy? *Research-Technology Management*, 2012.

[88] K. Madampe, R. Hoda, and J. Grundy. A faceted taxonomy of requirements changes in agile contexts. In *Trans. Softw. Eng.*, 2021.

[89] M. Manglaviti, E. Coronado-Montoya, K. Gallaba, and S. McIntosh. An Empirical Study of the Personnel Overhead of Continuous Integration. In *Int. Conf. Mining Softw. Repositories*, 2017.

[90] D. Margan and S. Čandrlić. The success of open source software: A review. In *Int. Convention on Information and Communication Technology, Electronics and Microelectronics*, 2015.

[91] A. Miller. A hundred days of continuous integration. In *Agile Conf.*, 2008.

[92] G. S. Nery, D. A. da Costa, and U. Kulesza. An empirical study of the relationship between continuous integration and test code evolution. In *Int. Conf. Softw. Maintenance and Evolution*, 2019.

[93] K. V. Paixão, C. Z. Felício, F. M. Delfim, and M. d. A. Maia. On the interplay between non-functional requirements and builds on continuous integration. In *Int. Conf. Mining Softw. Repositories*, 2017.

[94] J. A. Prado Lima, W. D. Mendonça, S. R. Vergilio, and W. K. Assunção. Cost-effective learning-based strategies for test case prioritization in continuous integration of highly-configurable software. *Empirical Softw. Eng.*, 2022.

[95] M. M. Rahman and C. K. Roy. Impact of continuous integration on code reviews. In *Int. Conf. Mining Softw. Repositories*, 2017.

[96] T. Rausch, W. Hummer, P. Leitner, and S. Schulte. An empirical analysis of build failures in the continuous integration workflows of java-based open-source software. In *Int. Conf. Mining Softw. Repositories*, 2017.

[97] P. Rostami Mazrae, T. Mens, M. Golzadeh, and A. Decan. On the usage, co-usage and migration of CI/CD tools: A qualitative analysis. In *Empirical Softw. Eng.*, 2023.

[98] I. Saidani, A. Ouni, M. Chouchen, and M. W. Mkaouer. Bf-detector: an automated tool for CI build failure detection. In *Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2021.

[99] I. Saidani, A. Ouni, and M. W. Mkaouer. Detecting Continuous Integration Skip Commits Using Multi-Objective Evolutionary Search. In *Trans. Softw. Eng.*, 2021.

[100] M. Santolucito, J. Zhang, E. Zhai, J. Cito, and R. Piskac. Learning CI configuration correctness for early build feedback. In *Int. Conf. Softw. Analysis, Evolution and Reengineering*, 2022.

[101] E. A. Santos and A. Hindle. Judging a commit by its cover: Correlating commit message entropy with build status on Travis-CI. In *Int. Conf. Mining Softw. Repositories*, 2016.

[102] K. Schwaber. Scrum development process. In *OOPSLA'95 BODI Workshop*, 1997.

[103] P. K. Sidhu, G. Mussbacher, and S. McIntosh. Reuse (or lack thereof) in travis ci specifications: An empirical study of ci phases and commands. In *Int. Conf. Softw. Analysis, Evolution and Reengineering*, 2019.

[104] SlashData. State of Continuous Delivery Report 2023: The Evolution of Software Delivery Performance. https://cd.foundation/state-of-cd-2023/, 2023. online.

[105] R. Souza and B. Silva. Sentiment Analysis of Travis CI Builds. In *Int. Conf. Mining Softw. Repositories*, 2017.

[106] D. Spencer and T. Warfel. Card sorting: a definitive guide. In *Boxes and arrows*, 2004.

[107] D. Ståhl and J. Bosch. Modeling continuous integration practice differences in industry software development. In *Journal of Systems and Softw.*, 2014.

[108] D. Ståhl and J. Bosch. Industry application of continuous integration modeling: a multiple-case study. In *Int. Conf. Softw. Eng. Companion*, 2016.

[109] D. Ståhl, K. Hallén, and J. Bosch. Achieving traceability in large scale continuous integration and delivery deployment, usage and validation of the eiffel framework. In *Empirical Softw. Eng.*, 2017.

[110] D. Ståhl, T. Mårtensson, and J. Bosch. The continuity of continuous integration: Correlations and consequences. In *Journal of Systems and Software*, 2017.

[111] K.-J. Stol and B. Fitzgerald. Uncovering theories in software engineering. In *SEMAT Workshop on a General Theory of Softw. Eng. (GTSE)*, 2013.

[112] K.-J. Stol and B. Fitzgerald. The abc of software engineering research. *ACM Trans. on Software Engineering and Methodology (TOSEM)*, 2018.

[113] K.-J. Stol, P. Ralph, and B. Fitzgerald. Grounded theory in software engineering research: a critical review and guidelines. In *Int. Conf. Softw. Eng.*, 2016.

[114] J. G. Süß and W. Billingsley. Using continuous integration of code and content to teach software engineering with limited resources. In *Int. Conf. Softw. Eng.*, 2012.

[115] C. P. Team. CMMI® for Development, Version 1.3. *Preface, SEI, CMU*, 2006.

[116] J. Tronge, J. Chen, P. Grubel, T. Randles, R. Davis, Q. Wofford, S. Anaya, and Q. Guan. BeeSwarm: enabling parallel scaling performance measurement in continuous integration for HPC applications. In *Int. Conf. Automated Softw. Eng.*, 2021.

[117] R. Turner, A. Ledwith, and J. Kelly. Project management in small to medium-sized enterprises: Matching processes to the nature of the firm. *International journal of project management*, 2010.

[118] R. Van Der Valk, P. Pelliccione, P. Lago, R. Heldal, E. Knauss, and J. Juul. Transparency and contracts: continuous integration and delivery in the automotive ecosystem. In *Int. Conf. Softw. Eng. Softw. Engineering in Practice*, 2018.

[119] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov. Quality and productivity outcomes relating to continuous integration in GitHub. In *Found. Softw. Eng.*, 2015.

[120] C. Vassallo, F. Palomba, and H. C. Gall. Continuous refactoring in ci: A preliminary study on the perceived advantages and barriers. In *Int. Conf. Softw. Maintenance and Evolution*, 2018.

[121] C. Vassallo, S. Proksch, H. C. Gall, and M. Di Penta. Automated reporting of anti-patterns and decay in continuous integration. In *Int. Conf. Softw. Eng.*, 2019.

[122] C. Vassallo, G. Schermann, F. Zampetti, D. Romano, P. Leitner, A. Zaidman, M. Di Penta, and S. Panichella. A tale of CI build failures: An open source and a financial -organization perspective. In *Int. Conf. Softw. Maintenance and evolution*, 2017.

[123] S. Vöst. Vehicle level continuous integration in the automotive industry. In *Found. Softw. Eng.*, 2015.

[124] Y. Wang, M. V. Mäntylä, Z. Liu, and J. Markkula. Test automation maturity improves product quality—Quantitative study of open source projects using continuous integration. In *Journal of Systems and Software*, 2022.

[125] D. G. Widder, M. Hilton, C. Kästner, and B. Vasilescu. I'm leaving you, Travis: a continuous integration breakup story. In *Int. Conf. Mining Softw. Repositories*, 2018.

[126] D. G. Widder, M. Hilton, C. Kästner, and B. Vasilescu. A conceptual replication of continuous integration pain points in the context of Travis CI. In *Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2019.

[127] R. Wieringa and M. Daneva. Six strategies for generalizing software engineering theories. *Science of computer programming*, 2015.

[128] R. Wieringa, M. Daneva, and N. Condori-Fernandez. The structure of design theories, and an analysis of their use in software engineering experiments. In *2011 International symposium on Empirical Softw. Eng. and measurement*, 2011.

[129] A. S. Yaraghi, M. Bagherzadeh, N. Kahani, and L. C. Briand. Scalable and accurate test case prioritization in continuous integration contexts. In *Trans. Softw. Eng.*, 2022.

[130] L. Yu, E. Alégroth, P. Chatzipetrou, and T. Gorschek. Automated NFR testing in continuous integration environments: a multi-case study of Nordic companies. *Empirical Softw. Eng.*, 2023.

[131] Y. Yu, H. Wang, V. Filkov, P. Devanbu, and B. Vasilescu. Wait for it: Determinants of pull request evaluation latency on github. In *Int. Conf. Mining Softw. Repositories*, 2015.

[132] F. Zampetti, G. Bavota, G. Canfora, and M. Di Penta. A study on the interplay between pull request review and continuous integration builds. In *Int. Conf. Softw. Analysis, Evolution and Reengineering*, 2019.

[133] F. Zampetti, S. Geremia, G. Bavota, and M. Di Penta. CI/CD Pipelines Evolution and Restructuring: A Qualitative and Quantitative Study. In *Int. Conf. Softw. Maintenance and Evolution*, 2021.

[134] F. Zampetti, V. Nardone, and M. Di Penta. Problems and solutions in applying continuous integration and delivery to 20 open-source cyber-physical systems. In *Int. Conf. Mining Softw. Repositories*, 2022.

[135] F. Zampetti, S. Scalabrino, R. Oliveto, G. Canfora, and M. Di Penta. How open source projects use static code analysis tools in continuous integration pipelines. In *Int. Conf. Mining Softw. Repositories*, 2017.

[136] F. Zampetti, C. Vassallo, S. Panichella, G. Canfora, H. Gall, and M. Di Penta. An empirical characterization of bad practices in continuous integration. In *Empirical Softw. Eng.*, 2020.

[137] C. Zhang, B. Chen, L. Chen, X. Peng, and W. Zhao. A large-scale empirical study of compiler errors in continuous integration. In *Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2019.

[138] C. Zhang, B. Chen, J. Hu, X. Peng, and W. Zhao. BuildSonic: Detecting and repairing performance-related configuration smells for continuous integration builds. In *Int. Conf. Automated Softw. Eng.*, 2022.

[139] C. Zhang, B. Chen, X. Peng, and W. Zhao. BuildSheriff: Change-aware test failure triage for continuous integration builds. In *Int. Conf. Softw. Eng.*, 2022.

[140] F. Zhang, A. Mockus, Y. Zou, F. Khomh, and A. E. Hassan. How does context affect the distribution of software maintainability metrics? In *Int. Conf. Softw. Maintenance*, 2013.

[141] L. Zhang, B. Cui, and Z. Zhang. Optimizing continuous integration by dynamic test proportion selection. In *Int. Conf. Softw. Analysis, Evolution and Reengineering*, 2023.

[142] Y. Zhao, A. Serebrenik, Y. Zhou, V. Filkov, and B. Vasilescu. The impact of continuous integration on other software development practices: a large-scale empirical study. In *Int. Conf. Automated Softw. Eng.*, 2017.

[143] C. Ziftci and J. Reardon. Who broke the build? Automatically identifying changes that induce test failures in continuous integration at Google scale. In *Int. Conf. Softw. Eng. Softw. Eng. Practice Track*, 2017.