Understanding Similarities and Differences in Software Development Practices Across Domains

Conference Paper · May 2019

CITATIONS
0

READS
71

5 authors, including:

Markos Viggiato
University of Alberta
9 PUBLICATIONS **4** CITATIONS
SEE PROFILE

Johnatan Oliveira
Federal University of Minas Gerais
14 PUBLICATIONS **56** CITATIONS
SEE PROFILE

Eduardo Figueiredo
Federal University of Minas Gerais
119 PUBLICATIONS **2,102** CITATIONS
SEE PROFILE

Pooyan Jamshidi
University of South Carolina
84 PUBLICATIONS **1,524** CITATIONS
SEE PROFILE

Understanding Similarities and Differences in Software Development Practices Across Domains

Markos Viggiato[1], Johnatan Oliveira[2], Eduardo Figueiredo[2], Pooyan Jamshidi[3], Christian Kästner[4]
[1]*Dept. of Electrical and Computer Engineering, University of Alberta*, Edmonton, Canada
[2]*Computer Science Department*, *Federal University of Minas Gerais*, Belo Horizonte, Brazil
[3]*Computer Science and Engineering Department*, *University of South Carolina*, Columbia, United States
[4]*Institute for Software Research*, *Carnegie Mellon University*, Pittsburgh, United States
viggiato@ualberta.ca, {johnatan.si, figueiredo}@dcc.ufmg.br, pjamshid@cse.sc.edu, kaestner@cs.cmu.edu

***Abstract*—Since software engineering is globalized and not a homogeneous whole, we expect that development practices are differently adopted across domains. However, little is known about how practices are followed in different software domains (e.g., healthcare, banking, and Oil and gas). In this paper, we report the results of an exploratory and inductive research, in which we seek differences and similarities regarding the adoption of several widespread practices across 13 domains. We interviewed 19 worldwide developers with experience in multiple domains (i.e., cross-domain developers) from large multinational companies, such as Facebook, Google, and Macy's. We also run a Web survey to confirm (or not) the interview results. Our findings show that, in fact, different domains adopt practices in a different fashion. We identified that continuous integration practices are interrupted during important commerce periods (e.g., Black Friday) in the financial domains. We also noticed the company's culture and policies strongly influence the adopted practices, instead of the domain itself. Our study also has important implications for global software engineering practices. For instance, companies should provide targeted training for their development teams and new interdisciplinary courses in software engineering and other domains, such as healthcare, are highly recommended.**

***Index Terms*—Software Domains, Development Practices, Interview Study, Cross-domain Developers**

I. Introduction

Software development practices play an important role in the quality of the final software product [1, 2, 3, 4]. This subject has captured the attention from both academia and practitioners [4]. In fact, several approaches have been proposed aiming at providing ways for developers to follow the best global practices in specific domains, such as design patterns [5], agile methodologies [6], and more recently continuous integration and DevOps [7, 8, 9]. Furthermore, some previous works have focused on the investigation of a single practice [3, 10]. However, differences in domains' characteristics have not been considered in many cases (such as software development in industry and software engineering education). Current investigation of widespread development practices also considers software engineering as a homogeneous field, contrasting with previous findings, such as indicated by Murphy-Hill et al. [11]:

> *In a larger sense, this work represents a step towards understanding software development not as a homogeneous whole, but instead as a rich tapestry of varying practices involving diverse people across diverse domains.*

In this paper, we use *practice* as a general term to refer to the way software is developed and its characteristics as well, which includes not only the software development phase, but also design, maintenance and evolution phases. For instance, we may use *practice* to refer to a well-established software development methodology (e.g., agile) or a specific way of adopting continuous integration during software development. Despite the relevance of global software development practices, there is little exploratory research aiming at characterizing how these development practices are adopted across domains and little is known regarding which practices are applied (and how they are applied) in different software domains.

Shedding light on how and which practices are adopted is important for many reasons. First, practitioners will benefit from our results as we provide insights about development practices in software domains. For instance, this may be helpful for professionals and companies that are migrating to a new domain as they will be aware of the common practices and their use in that domain. Companies can also provide targeted training for their teams based on development practices adopted in the domain of interest. For instance, e-commerce companies do not uniformly use continuous integration practices throughout the year as they avoid releasing code changes in periods of high amounts of sales. In such case, developers should be instructed to only release code that is critical for the business. Furthermore, new customized tools may be developed targeting developers from specific domains as there is a need for domain-aware tools [12]. Second, the research community may also benefit from our study. Future researches may gain more generalization, since the results for one case (e.g., the investigation of a specific practice for one domain) may generalize for other software domains. Finally, software engineering education may take specific development practices into account as different domains have different development practices. For instance, a new branch of the software engineering course focused on game development may be suitable since this domain differs in many aspects from non-game software development [11]. New interdisciplinary

courses should also be thought for specific domains [13].

Our goal in this paper is to understand how development practices globally used by different companies differ in different software domains and whether there are specificities in their use, i.e., we aim to verify whether developers from different domains can adapt development practices to their specific context. We hypothesize that some domains may have similarities in the use of development practices and we also believe some domains may adopt practices in such a specific way that makes them very different from the others. However, works so far have not focused on the differences of domains regarding global development practices. In addition, previous studies have not investigated the adoption of development practices based on the perception of cross-domain developers, as we do here. In this paper, we adopt an exploratory and inductive research [14, 15, 16, 17] to seek for differences and similarities of several practices across 13 domains. To guide our study, we defined the following research questions:

- *RQ1: Which development practices are similar across domains?*
- *RQ2: Which development practices are specific to domain?*
- *RQ3: Which factors may impact the adoption of development practices in different software domains?*

To answer the research questions, we designed an exploratory, qualitative study in which we interviewed worldwide cross-domain professionals who have worked in more than one domain in the software industry. We argue (Section III) that developers with experience in more than one domain are capable of indicating differences in development practices with more confidence. We conducted 19 semi-structured interviews with developers from 13 software domains, such as social networks, healthcare, banking, e-commerce and games. Afterwards, we transcribed the recorded interviews looking for interesting themes. We then validated interview findings through a Web survey with developers around the world. It is important to highlight that cross-domain developers were difficult to be found and contacted, specially in some domains (e.g., aviation), in which we believe developers are highly specialized. In this paper, we do not present results regarding domains in which we believe the saturation was not reached. We briefly present interesting findings regarding those domains in Section V-C and emphasize the need for further investigation. We noted that 19 interviews were sufficient to reach the theoretical saturation [14], similarly to previous studies [11, 18, 19, 20]. For instance, when interviewing the 7^{th} e-commerce developer, all information provided by the interviewee was known by interviewers as previous e-commerce participants already provided them.

Our findings suggest many differences. For instance, e-commerce developers usually avoid releasing code in periods of the year when there is large financial transactions. This means the interruption of continuous integration and continuous delivery practices within that domain. Furthermore, healthcare developers mentioned that requirements elicitation is easier in comparison to other domains as health professionals usually have a higher degree of education. However, this result was not confirmed by the survey participants, that mostly disagree with it (50%) or are neutral (10%). Although this can be seen as a negative result, it is important to note that negative or unexpected results may be found through exploratory studies. We also found other interesting results, such as the fact that social network domain does not have dedicated test teams. Instead, developers test and fix bugs themselves, using a *code-owner* approach.

The remainder of this paper is structured as follows. Section II presents related works. Section III explains how we designed our research. In Sections IV and V, we present the results and discussions, respectively. Section VI shows threats to validity and Section VII concludes the work and discusses next steps.

II. Related Work

Several studies have been proposed within the context of software development practices [1, 2, 3] and software domains [21, 22, 23, 13]. Regarding studies within the context of software domains, Murphy-Hill et al. [11] presented a study comparing game development to traditional software development. The work indicated substantial differences between video game development and other software development segments, such as the rare use of automated tests in game development. Richardson et al. [13] noticed that regulations and directives regarding medical device software were not being taken into account, and unregulated software was developed and used in healthcare organisations. This was a result of not trained software engineers, who lacked knowledge in regulations of software solutions for healthcare. The authors recommended that healthcare software systems should be developed by professional software engineers in interdisciplinary teams with healthcare professionals. Russo et al. [22] aimed at identifying some relevant concerns in the Italian banking IT sector, through an investigation of the opinions of several stakeholders. The authors identified 15 concerns, which were discussed in a framework inspired by the ISO 25010 standard.

Segura et al. [21] explored the applicability of some of the practices for variability management in software product lines to an e-commerce website. The authors used a feature model to represent the store input space and techniques for the automated analysis of feature models for the detection and repair of inconsistent and missing configuration settings. Their findings suggest that variability techniques could successfully address many of the challenges found when developing e-commerce websites. Other studies investigated development practices without focusing on specific software domains. Wright and Perry [2] reported the initial results of a study in which the authors interviewed 4 practicing release engineers to understand the faults and failures of release practices, how companies recover from them and how to predict and avoid the failures in the future. Their preliminary results indicate

that a more thorough process analysis and efforts at process standardization are necessary.

Unlike all previous works discussed above, in this paper we interview cross-domain professionals (i.e., developers who have worked in more than one target domain) with the aim of identifying development practices that are similar across domains and practices that are particular to specific software domains. To the best of our knowledge, this is the first exploratory study to investigate development practices across several domains by interviewing cross-domain developers. Our work also can be seen as a complementary study to the previous ones as we provide a more thorough insight of how development practices are being used in industry across several domains.

III. Research Design

A. Software Domains

We selected a set of software domains for our study based on previous works [22, 24, 13, 11], i.e. we selected domains which were already subject of research and, therefore, we believe they are well-known in the global software engineering community and easily understandable by industry professionals. Furthermore, we believe it is feasible to find worldwide software developers who have worked in such domains through our participant search procedure. Initially, we selected the following 13 domains: accounting, aviation, banking, business, e-commerce, educational, games, healthcare, mining, oil and gas, search engine, social network, and stock market.

B. Methodology

We conduct a qualitative study to help us better understand how global software development practices are used in different software domains. We follow an inductive research strategy, using a grounded, iterative approach to let development practice patterns of usage emerge from the interviews [25, 17]. This means we do not have previous categories to classify the use of development practices in different domains. To achieve our goal, we conducted semi-structured interviews with software professionals from industry with experience in multiple domains. The research methodology is composed of five stages: (i) participants selection in LinkedIn; (ii) interview design; (iii) conduct of interviews; (iv) transcription analysis; and (v) validation through a Web survey. The last stage was executed to confirm (or not) the main findings for domains in which we reached saturation. We noticed that 19 interviews were sufficient to gather interesting information regarding the adoption of development practices in different domains and to reach the saturation in some domains. In fact, previous interview studies performed a similar number of interviews, such as Murphy-Hill et al. [11] (14 interviews), Stacey and Nandhakumar [18] (20 interviews), Burger-Helmchen and Cohendet [19] (8 interviews), and Dagenais and Robillard [20] (22 interviews). We stopped conducting interviews as new interviews with participants from the following domains were not bringing new information: banking (with 6 interviews), e-commerce (with 8 interviews), and healthcare (with 5 interviews). Therefore, in this paper, we focus on presenting results from the aforementioned domains and we briefly indicate interesting findings from domains in which we have not yet reached the saturation (Section V-C), namely: oil and gas and social networks.

C. Interview Process

Our interview process is iterative and we use the open coding technique from grounded theory [14, 15, 26, 16]. The interview phases are simultaneous, i.e., the stages overlap. For instance, while conducting interviews with some participants, we may also continuously select additional participants and iteratively build the interview script according to the previous interviews. Next, we describe each stage in detail.

Participants selection. We propose an innovative method to select the interview participants, which is an important contribution of our work. We selected only cross-domain developers from global software industry, i.e., developers who have worked in more than one software domain. This selection criterion makes sure the developer has experienced more than one domain and, therefore, can confidently state the differences in the development practices' adoption. Table I presents information regarding the domains to which participants belong and years of experience with software development. We anonymously identify each participant by using the letter P followed by an identifier number (e.g., P1, P2, and so on until P19). On average, the interviewees have 11.7 years of professional experience and most holds at least one postgraduate degree, including masters and doctorate. Most of the interviewees currently work or have worked as developers for large multinational companies, with thousands of employees and whose services and products reach millions of users, such as *Facebook, Google, Macy's, General Electric*, and *Petrobras*. In addition, the participants workplaces are distributed around the world, such as participants who are currently working in the United States, Canada and Brazil. Some participants have experience in three or even four domains and for such cases we decided to do the interview with respect to the domains in which developers have the most experience.

To check that participants were in fact cross-domain, we carefully and manually inspected their LinkedIn accounts and we selected only developers who have worked in companies or projects within the targeted domains. In addition, developers should have at least 5 years of professional experience and 1 year of work within each domain. By following these criteria, we believe participants' statements are more confident regarding similarities and differences in adopted practices, which also brings more confidence to our results. We started with an opportunistic selection through a search in our LinkedIn contact lists. Furthermore, we implemented an algorithm to automatically look for software developers from each domain by performing text analysis on the developer LinkedIn. The algorithm returns the developers' name and LinkedIn account, which were manually validated by two authors. This double-checked procedure helps to ensure that all participants meet the

TABLE I
INTERVIEWEES INFORMATION.

Participant	Experience (Years)	Domain 1	Domain 2
P1	20	Banking	Healthcare
P2	9	Accounting	E-commerce
P3	8	E-commerce	Social Network
P4	10	E-commerce	Education
P5	12	Healthcare	Oil and Gas
P6	10	E-commerce	Search Engine
P7	16	Banking	E-commerce
P8	9	Education	Healthcare
P9	11	Accounting	E-commerce
P10	25	Banking	Mining
P11	7,5	Games	Mining
P12	16	Banking	Games
P13	5	Aviation	Healthcare
P14	17	Banking	Stock Market
P15	7	Healthcare	Stock Market
P16	15	Business	Stock Market
P17	8	Accounting	Education
P18	8	Banking	E-commerce
P19	10	Accounting	E-commerce

defined selection criteria. We contacted developers by email (when available anywhere online, such as on GitHub) or by the *LinkedIn InMail* functionality. The process of selecting candidates for the interviews took too long due to the manual validations and mainly the difficulty of findings cross-domain developers with experience in at least two domains. Finding cross-domain developers is even harder in some specific domains (e.g., aviation), as developers from these domains are highly specialized and usually do not have experience in other domain of our interest. We sent 62 emails to the cross-domain developers we identified and validated in LinkedIn, and we received confirmations from 24 developers. However, 5 developers declined later due to concerns regarding their companies' private information, even though we made it clear all the process would be anonymized and we were trying to understand general practices adopted. Thus, we interviewed 19 developers (response rate of 31%).

Interview design. To guide us during the interviews, we iteratively developed an interview script, which is composed of three main sections: background of the participant, general questions regarding differences in use of the software development practices, and specific questions regarding a set of practices, such as software testing and DevOps practices. Through the first section, we are interested in participants academic and professional background, such as the bachelor's degree, the highest academic degree, and years of experience. In the second section of the interview, we asked general questions regarding differences in software domains. In this part, we are interested in getting the participant's perception about the development practices in different domains without biasing our specific questions. Finally, in the last section, we asked specific questions about some development practices. In this section of the interview, we focus on the topics not mentioned by the interviewee in the second section in order to cover all target development practices. Our questions cover the following practices: *releasing practices* (e.g., regarding the deadlines of product releasing), *quality assurance* (which included test practices), *code review practices*, *continuous integration and delivery*, version control practices, and *practices related to the software architecture*, such as whether the team is aware and discusses architectural impacts caused by changes in the system [27].

Conduct of the interviews. After the usual consent process with each participant, we start the interview, planned to last no more than 40 minutes. We observed this period was sufficient to do a concise interview, since we could collect all information we needed. We also recorded all interviews with the consent of the participants. Most interviews were conducted through a conversation on Skype. However, when the participant was not available due to agenda incompatibility, we sent out the interview by email, doing a follow-up whenever necessary (e.g., to better understand some responses).

Interview transcription analysis. The last stage of the interview process is the transcription and analysis of the interview to extract all relevant information. Here we used the open coding technique. The first author of the paper carefully analyzed the transcriptions and came up with the most relevant and groundbreaking topics stated by the interviewees, which were discussed afterwards by all authors of the paper.

D. Survey Validation

To check whether practices mentioned by interviewees are in fact broadly adopted by developers from each domain, we designed an online survey. It is important to note that we validated adopted practices only for software domains in which we reached the saturation, which occurred for the following domains: banking (6 interviewees), e-commerce (8 interviewees), and healthcare (5 interviewees). The survey is composed of two main sections: background (common to all surveys) and questions regarding a software domain (specific to the survey of each domain). Through the first section, we intended to collect information related to the participants background such as education, software development experience and development experience within the specific domain. The second section contains concise and objective statements that present characteristics and adopted practices within the domains, as indicated by the interviewees. In this section, the survey participant is asked to indicate the agreement with the statement through a Likert-type scale.

To find participants for the survey, we first mined global software repositories related to the target domains from GitHub in order to collect the names and emails from top-committers. We used specific search strings to make sure the repositories belong to the domains of our interest. To retrieve repositories from the banking domain, we used *bank*, and *banking* strings; for the e-commerce domain, we used *e-commerce*, *e-commerce* and *electronic commerce* strings; finally, for the healthcare domain, the following strings were used: *healthcare*, and *health*. Then, the repositories were manually validated to certify they are in fact software systems and they really belong to the domain. Finally, we automatically collected the name and email of top-committers (number of

commits greater than 100) from repositories so that we could send the survey by email. We believe users with more than 100 commits have sufficient knowledge within the domain and therefore are capable to answer our survey. With this procedure, we do not aim to find an extensive list of systems and committers. Instead, our goal is to find a representative number of worldwide developers with a good knowledge in the domain (top-committers) to answer our survey. We had to discard a very large number of committers since they did not meet our criteria of 100 commits and possibly were not capable of confidently answering the survey. In addition to this strategy, we searched for additional participants in LinkedIn since GitHub does not contain many popular repositories that meet our criteria (belonging to specific domains and developers with more than 100 commits). We looked for worldwide developers from the three domains within LinkedIn and sent the survey by email. After sending 329 emails, we received 37 complete responses from participants worldwide (response rate of approximately 11%), being 12 for banking, 14 for e-commerce and 11 for healthcare.

IV. Results

In this section, we report results from the interviews we conducted. Note that we present interview quotes that are supported by at least three interviewees from different companies. In parallel, we report the percentages of agreement (or disagreement) of survey participants regarding each practice reported by interviewees. Figure 1 shows a summary of our main results obtained from the interviews, which are discussed in Sections IV-A, IV-B, and IV-C. Rectangles indicate domains and ellipses indicate practices (e.g., continuous integration practices) or characteristics (e.g., interoperability). Arrows indicate which domains are related to practices/characteristics and arrow labels show how that practice/characteristic is adopted in that domain (e.g., continuous integration is interrupted in the banking domain) as reported by interviewees.

Table II presents an overview of the survey results for the statements regarding characteristics and adopted practices in each domain. The first column shows the statements presented to participants. Each statement is identified by a unique label. For instance, we use *S1.B* to identify the first statement of the banking survey. The second column presents the Likert distribution of the participants agreement regarding each statement.

A. Banking Domain

Continuous integration interruption. Interviewees from the banking domain often mentioned they are more careful with dates when financial transactions increase, mainly in the end of the year and beginning of each month. As a contrast to other domains, banking developers pointed out that in such periods they do not release large code changes to the servers, interrupting the continuous integration process in order to avoid inserting bugs in the systems during critical periods. They also stated that the priority is to fix bugs. Participant P10 said the following quote (supported by P1, P7, and P12):

> *Most of the banks have a freeze period about 30 days before the new year, when just emergency software updates are allowed.*

Besides Black Friday and new years period, participants also mentioned that development is modified prior to salary payments, when the traffic is usually high. Below, a quote from P18 with support of P1 and P7 (note that salaries are paid in the beginning of the month in the participant's country, which may not be the case for other locations):

> *We usually do not release large code changes to the server in the first days of the month, just before salary payments.*

According to the banking survey, 58.3% of participants agree with this practice (scores 4 and 5), while only 16.6% disagree (scores 1 and 2). In addition, 25% of participants were neutral (score 3). The survey responses indicate that developers usually adopt this practice in banking development.

Moderated regulatory demands. Developers also highlighted that banking systems are regulated by legal demands that come from the government, which is not common in other software domains. Hence, one common practice is to change the code to comply to a regulatory demand, such as stated by P10 and supported by P1 and P4:

> *Most of the time was used to enhance an existing feature, add a new one or comply to a regulatory change.*

According to the banking survey, 83.3% of participants agree with this practice (scores 4 and 5), while only 16.7% are neutral (score 3). No respondents disagrees. The responses indicate that in fact the banking domain requires specific changes (besides usual ones) to comply to regulatory demands.

Overly complex requirements. Another characteristic of the banking domain is regarding requirements engineering practices. Developers from banking often said that understanding what stakeholders really want may be difficult due to the context where the system will operate, many times requiring the understanding of financial terms. Participant P10 mentioned (supported by P14 and P18):

> *...it is hard to understand and put everything together because it involves abstract, complex, and structured financial operations.*

According to the banking survey, 83.4% of participants agree with this practice (scores 4 and 5), while only 8.3% are neutral (score 3) and 8.3% partially disagree (score 2). The responses strongly indicate the high complexity of requirements elicitation in the banking domain.

B. E-commerce Domain

Continuous integration interruption. Similarly to the banking domain, e-commerce developers also adopt practices of interrupting continuous integration, according to our interviewees. However, in this case, software development is oriented to commerce important dates, when the amount of sales increase. In such periods, participants mentioned that the priority is to fix bugs and give the best experience for users. Therefore,

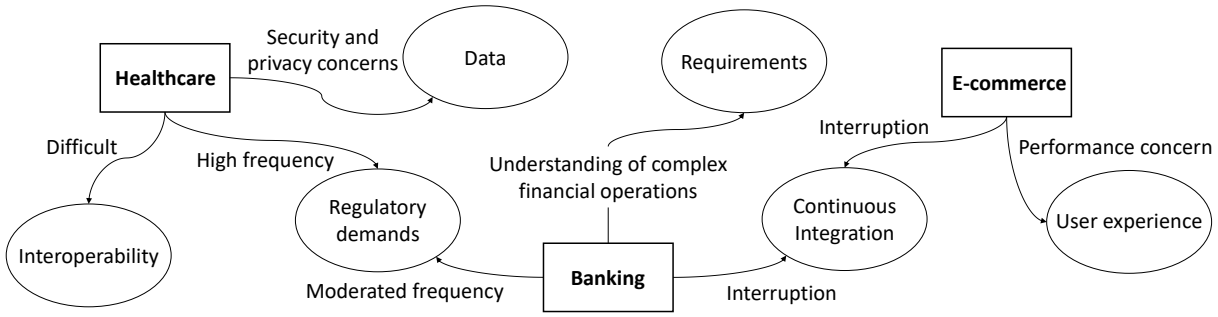
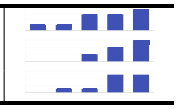
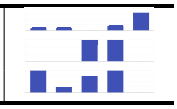
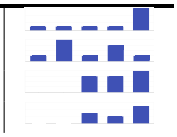


Fig. 1. Main adopted practices in domains. Banking domain is moderately regulated and interrupt continuous integration process in important commerce periods (e.g., Black Friday); e-commerce follows an user-centered development, focusing on non-functional requirements that provide a good user experience and also interrupt continuous integration process; and healthcare is highly regulated, focuses on patient data privacy and security and requirements elicitation may be easier than in other domains.

TABLE II
SURVEY RESULTS WITH PRESENTED STATEMENTS AND LIKERT-SCALE AGREEMENT DISTRIBUTION.

Banking	
S1.B - Code changes are less frequently released in periods of the year when large financial transaction are performed.	
S2.B - The banking segment is moderately regulated, many times requiring changes in the system to fulfill regulatory demands.	
S3.B - Requirements elicitation is hard because it envolves the understanding of complex financial operations.	
E-commerce	
S1.E - Code changes are often not released in periods of high amount of sales, such as in Black Friday and Christmas.	
S2.E - This segment focuses on user-centered non-functional requirements, such as usability, security and performance.	
S3.E - Code is pushed into production with less frequency compared to other software segments.	
Healthcare	
S1.H - The healthcare segment is highly regulated, with frequent legal demands.	
S2.H - Requirements elicitation is relatively easier compared to other segments	
S3.H - Interoperability of systems from different workplaces is usually difficult.	
S4.H - Privacy, reliability and security of patient data are major concerns in healthcare software.	

developers may change their usual continuous integration practices (i.e., they stop sending large changes to the servers) aiming at focusing on the most important tasks, such as bug fixing. A quote from P6 supported by P2, P4, P7 and P9:

> *We have code freezes a few weeks prior to holidays seasons, when only critical or major bug fixes could be introduced. A week prior to the holidays absolutely no code was checked in unless critical to the business.*

According to the e-commerce survey, 83.4% of participants agree with this practice (scores 4 and 5), while only 16.6% disagree (scores 1 and 2). The high agreement percentage suggests this practice is in fact widely adopted by e-commerce developers.

Focus on user experience. According to e-commerce interviewees, developers give a special attention to specific user-centered non-functional requirements, mainly performance, usability and security. According to them, the user-focused development aims at providing the best user experience as possible, since a low performance system may prevent the user from concluding a purchase. E-commerce development practices include stress tests to guarantee the system will provide a good experience for users. P3 mentioned (supported by P6, P18 and P19):

> *Performance is critical for user experience. We have stress-test environments where the numbers are pushed to limits (visits, users, transactions, and many other metrics that could be extrapolated).*

According to the e-commerce survey, 50% of participants partially agree with this practice (score 4), while 50% are neutral (score3). This may indicate that focusing on the user-experience is a generic characteristic, being important in other domains as well.

Less frequent continuous delivery. Finally, interviewees also mentioned that continuous delivery is less frequent in e-commerce development in comparison to other software domains, since code changes are extensively tested before being put into production. This happens to make sure no bug would be inserted into the system, which could cause a bad experience for the user and reduce the number of visitors of the website. P6 stated (supported by P2 and P4):

> *We had less frequent pushes to production in e-commerce domain due to extensive code change tests.*

According to the e-commerce survey, 41.6% of participants disagree with this practice (scores 1 and 2), while 25% are

neutral (score3) and 33.3% partially agree (score 4). The high percentage of disagreement and neutrality may indicate that this practice is not commonly adopted in the e-commerce domain and it may only reflect interviewees experience within their companies. Looking at this results, we believe that less frequent continuous delivery is strictly related to companies policies and culture, as it may reflect the personal experience of interviewees who mentioned that.

C. Healthcare Domain

Frequent regulatory demands. The healthcare domain is a well-established and largely known software domain within both academia and industry. This domain has peculiarities that differ it from the others, such as the regulations that health systems usually must follow [13, 28]. In fact, interviewees from the healthcare domain corroborate with this belief. For instance, they mentioned that when a legal demand arrives, the developer team needs to focus on implementing this new demand, giving it the highest priority. P5 stated (supported by P1, P8 and P13):

> *Health domain is more regulated and oriented by legal demands, which come with a preestablished date.*

According to the healthcare survey, 70% of respondents agree (scores 4 and 5) with this characteristic. More specifically, 60% completely agree with it, indicating that in fact healthcare software is higly regulated. Scores 1, 2, and 3 received 10% of responses each.

Clearer requirements. Regarding the requirements engineering practices, it is common believed that this phase of the software development is really difficult and complex [1]. However, participants from the healthcare domain contradicted this belief, claiming that requirements elicitation in healthcare domain is not as difficult as in other domains, such as Oil and Gas (pointed out by P5) and banking (stated by P10). As they said, despite the common lack of time of health professionals, the requirements in this domain are clearer due to the (usually) higher qualification of health professionals (e.g., medical doctors). Therefore, such professionals can easily understand and keep a conversation with IT professionals, making the requirements elicitation relatively easier and clearer, as P5 mentioned (supported by P1 and P13):

> *Requirements are clearer in healthcare due to the higher qualification of health professionals (medical doctors).*

According to the healthcare survey, 50% of respondents disagree (scores 1 and 2) with this characteristic. In addition, 10% are neutral (score 3) and 40% agree (scores 4 and 5). This agreement distribution indicates that requirements elicitation in healthcare domain may be strictly dependent on the personal experience of developers and the health companies for which they have worked. Therefore, it may reflect a characteristic of the companies' policies and culture, instead of an intrinsic characteristic of the healthcare domain itself. Furthermore, we believe that age and maturity of companies strongly influence requirement engineering practices, as older companies may have acquired experience with requirements elicitation, making it easier as indicated by interviewees.

Difficult interoperability. Interviewees from healthcare also mentioned the difficult they usually face regarding interoperability practices of systems from different companies. For instance, even though there are some standards, hospitals may have surprisingly different information patterns, which difficult the communication among them., as P5 mentioned (supported by P1 and P8):

> *Although there are standards, hospitals, for example, rarely switch information because they have different information formats.*

According to the healthcare survey, 70% of respondents agree (scores 4 and 5) with this practice. In addition, 30% are neutral (score 3). The responses indicate that in fact interoperability is a challenge in the healthcare domain.

Data security and privacy concerns. Healthcare participants often mentioned the importance of reliability, privacy and security regarding patient data. The whole development process is concerned with the patient data, always trying to keep them reliable in order to avoid possible serious consequences. For instance, participant P5 stated (supported by P1, P13 and P15):

> *If I switch patient data, I can give wrong diagnoses and indicate wrong medications.*

According to the healthcare survey, 70% of respondents agree (scores 4 and 5) with this practice, while 30% are neutral (score 3). The responses suggest that developers in fact consider data security and data privacy major concerns in the healthcare software development process.

V. Discussion

In this section, we discuss the results obtained from the interviews and from the survey. It is important to note that we answer the research questions based on practices and characteristics from domains in which there was agreement between the interviewees and the survey participants. This gives more confidence to our conclusions as broader and more diverse set of developers agree with that practice.

We noticed that both banking and e-commerce domains share a common practice of interrupting the continuous integration process in periods of the year when the amount of sales increase, such as Black Friday and new year. Furthermore, regulatory demands are common in the banking and healthcare domains, usually requiring efforts from the development team to implement changes into the system to comply to regulatory requirements.

Answering RQ1: We found two similarities of practices across domains. First, continuous integration practices are adopted in a similar way in the banking and e-commerce domains, which suggests that other financial-related domains may also follow this practice. Second, regulatory-driven changes are common in the banking and healthcare domains, which must adapt their workflow to comply to regulatory demands.

Requirements elicitation in the banking domain is different from the other domains we investigated, since an understanding of complex financial operations is necessary to precisely capture requirements needs. The healthcare domain is different from other domains regarding interoperability. For instance, many health companies may have different information patterns, which may hinder information switching between companies. Other domains (e.g., mining, banking, and oil and gas) have widely used standards that ease information switching whenever necessary.

Answering RQ2: We found two main practices specific to domain. First, requirements engineering practices are adopted in an unique way by the banking domain, involving the comprehension of complex financial operations. Second, practices related to interoperability are more difficult in the healthcare domain in comparison to others, due to different standards used by health companies.

Through the third research question, we are interested in capturing the main factors that can influence which development practices the companies adopt. Based on our interpretations of the interviews, we noticed that the company's policy and culture play an important role when deciding about the software development process. Many times, the software engineering team is required to follow specific practices due to the company way of work. For instance, as we already discussed, we identified that less frequent continuous delivery practices in e-commerce and requirements engineering practices in healthcare resulted from companies' policies and culture. Furthermore, the age and maturity also have a strong impact on adopted practices. We realized that companies may change or adapt practices throughout the years, also as a result of the emergence of new technologies and development processes.

Answering RQ3: The companies' policy and culture are important factors that guide the development process, therefore impacting the adopted practices. Moreover, age and maturity also may influence the practices' adoption and their way of use.

A. Implications for Global Software Engineering Practices

In this section, we elaborate on the three main practical implications our results can have based on the joint analysis of the interview findings and the survey responses. First, companies should provide targeted training for their employees, not only software developers, but also training for people from other positions (e.g., software architect and technology leader). The training should focus on specific domains' characteristics and how development practices are adopted within the company's domain(s).

Second, professionals should update themselves regarding which and how practices are adopted in domains, specially if they are looking for a new job. For instance, developers who work (or intend to work) with banking software should understand (at least basic) financial operations as this my strongly aid the requirements elicitation.

Third, software engineering education professionals should consider specificities of different software domains. We believe new teaching approaches that consider the domain should be developed. For instance, new specific undergraduate or graduate courses may be interesting. Interdisciplinary courses may also be a good idea, as Richardson et al. [13] recently suggested an interdisciplinary course of software engineering and healthcare.

B. Contrast with Current Beliefs

Our results give insights about characteristics and ways global development practices are used in specific software domains and some results may be surprising for many practitioners or contradict the current common sense. In this section, we present how our results are surprising or contrast with current beliefs regarding development practices in software domains.

Continuous integration may not always be a homogeneous global software engineering practice in some domains. This practice has emerged recently aiming at automating the compilation, building and testing of code, with weekly and even daily integration [29, 30, 31] and some studies have investigated continuous integration flexibility, costs and benefits [32, 33]. Most developers keep adopting this practice based on how everyone uses, but the academy has not investigated so far whether there are differences in continuous integration usage. Surprisingly, we identified that developers from banking and e-commerce (i.e., financial domains) usually interrupt continuous integration in critical commerce periods, such as Black Friday, aiming at avoiding inserting subtle bugs in the systems, which would be catastrophic for the company. We did not identify this practice in the other domains we investigated at all, suggesting it possibly is exclusive from financial domains.

C. Results for Other Domains

In this section, we present other interesting findings from the interviews in which we did not reach the saturation. Therefore, these results provide insights regarding some domains and we emphasize the need for further investigation focusing on these specific domains.

Releasing practices flexibility in Social Network and Search Engine domains. Interviewees from social network and search engine domains often mentioned the flexibility they usually have regarding many aspects, such as the release deadlines. We may expect that software development has extremely strict deadlines of releasing a product, as indicated by interviewees from banking and e-commerce domains. However, this seems not to be the rule for social network domain, as participant P3 said:

> *...developers prioritize product and technology excellences. There is less pressure for the deadline itself.*

Participant P6 reported how developers are assigned to the projects. We may expect developers are told what they need to develop and they just do it. However, a common practice in social network systems is that developers have the freedom to choose the project and the feature they work on, as P6 mentioned:

> *I have complete freedom to choose what kind of project I'm going to work on, what I want to do.*

Although domains usually have a dedicated testing team, such as in banking (stated by P1) and e-commerce (stated by P7), interviewee P3 pointed out, as a contrast to other domains, that tests are performed by the developers themselves in the social network domain. More specifically, the developer who implemented a feature, for example, is responsible for testing it. This code-owner based approach has been adopted only recently in systems with modern architectures, such as microservices [34, 35]. Therefore, the adoption of this practice may be a result of architectural decisions in this domain. A quote from P3:

> *...there is no test team. The developer is responsible for creating all integration, web-driven, and unit tests.*

Finally, we concluded that social network and search engine domains are quite peculiar, presenting unexpected management practices (decisions about the projects in which developers work and deadline policy) and test practices.

Automatic fault-recovery in Oil and Gas domain. The participant from the oil and gas domain pointed out that this domain must take into account the need for an automatic fault-recovery module, which is present during the entire development process, from the requirements until the delivery and operation. In addition, the software system must be extremely robust, given the environmental conditions of operations (e.g., an oil platform in the middle of the ocean). One of the reasons behind these needs is that the systems remain physically inaccessible for a long period of time, since professionals do not have continuously access to the location where the software is deployed, which is common in other domains (e.g., healthcare as pointed out by P13). Remote connections may also be difficult given the location of the system. A quote from P5:

> *Oil and Gas requires more robust and autonomous solutions since the system is hard to reach for a long time.*

VI. Limitations and Threats to Validity

The study presented in this paper has some limitations that could potentially threaten our results, as we explain next. First, one may point a company from a domain we investigated and may say the company does not adopt the practices as we presented. However, our findings are based on interview participants' perceptions and their experience, and therefore our results may not generalize to all companies, as each one can adopt development practices based on its own culture and policy. Note that, in this study, we focus on large companies, such as *Facebook*, *Google*, *Petrobras*, and *Macy's*. Therefore, our results may not hold for small companies possibly with informal software engineering processes. This kind of limitation is characteristic of qualitative studies, as previously studied [36, 37]. However, Flyvbjerg [38] demonstrated that even individual cases (i.e., studies limited to one company) contributed to discoveries in several fields, such as physics and social sciences. Therefore, even within a limited context of a few companies and participants, we believe our results can impact how companies from the studied software domains can adopt development practices.

Second, another limitation of our study is related to our methodology for finding cross-domain developers. We rely on a semi-automated search for interview participants, manually validating LinkedIn profiles returned by an algorithm we implemented. However, we may have misclassified developers as cross-domain (e.g., assigning a domain in which the developer has never worked). This may have caused a reduction in the response rate for the interview since there would be wrong information regarding the domains in which the developers we contacted have worked. To mitigate such issue, we have performed a double check for each participant before contacting.

Third, one may point that our interview results are based only on participants personal experience. However, we selected practitioners with a diverse background. This scenario composed of several large companies and different work locations bring more generalization to our results since we believe that biases (e.g., from a specific sort of company or a specific location) are attenuated. In addition, the Web survey collected responses from developers worldwide wit different backgrounds, which supports our interview results regarding adopted practices within domains.

Finally, during the interviews, we asked questions about specific development practices. One may argue that this is a limiting factor and does not allow the interviewee to provide information about a wider range of topics. However, in the last section of the interview, the participant could talk about any desired topic, including information not previously given.

VII. Conclusion and Next Steps

In this paper, we report the results of an exploratory qualitative study in which we conducted 19 semi-structured interviews with worldwide cross-domain developers who have worked in several multinational companies from different domains around the world. We also run a Web survey to check whether development practices revealed by interviews are widely adopted.

Our findings suggest that different domains in fact adopt development practices in different manners. For instance, financial domains interrupt the continuous integration process in commerce critical periods (e.g., Black Friday), when the amounts of sales sharply increase. The company's culture and policies also impact decisions about development practices, as previously suggested by Bogart et al. [39]. We emphasize the need for further investigation regarding practices and domains' characteristics in which there was disagreement

between interviewees and survey participants, such as the less frequent continuous delivery in e-commerce systems and clearer requirements practices in healthcare. It is important to note that we do not claim our results should be universally adopted by companies within the domains we investigated. Instead, a careful analysis is recommended for each case. Here we are providing insights that might possibly be adopted, given their successful use in global software engineering industry so far. As future work, we intend to conduct a series of studies focusing on domains in which we did not reach the saturation, but interesting information was collected, such as for social networks and oil and gas domains. In addition, highly specialized domains that potentially have interesting practices need focused studies as well, such as aviation.

VIII. ACKNOWLEDGMENTS

This research was partially supported by Brazilian funding agencies: CNPq (Grant 290136/2015-6 and 424340/2016-0), CAPES, and FAPEMIG (Grant PPM-00651-17).

REFERENCES

[1] B. Yost, M. Coblenz, B. Myers, J. Sunshine, J. Aldrich, S. Weber, M. Patron, M. Heeren, S. Krueger, and M. Pfaff, "Software development practices, barriers in the field and the relationship to software quality," in *10th Int'l Symposium on Empirical Software Engineering and Measurement*, 2016.

[2] H. K. Wright and D. E. Perry, "Release engineering practices and pitfalls," in *34th Int'l Conference on Software Engineering*, 2012.

[3] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida, "Investigating code review practices in defective files: An empirical study of the qt system," in *12th Working Conference on Mining Software Repositories*, 2015.

[4] M. Stavnycha, H. Yin, and T. Römer, "A large-scale survey on the effects of selected development practices on software correctness," in *2015 Int'l Conference on Software and System Process*, 2015, pp. 117–121.

[5] E. Gamma, *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.

[6] J. Highsmith and A. Cockburn, "Agile software development: The business of innovation," *Computer*, vol. 34, no. 9, pp. 120–127, 2001.

[7] P. M. Duvall, S. Matyas, and A. Glover, *Continuous integration: improving software quality and reducing risk*. Pearson Education, 2007.

[8] B. Fitzgerald and K.-J. Stol, "Continuous software engineering: A roadmap and agenda," *Journal of Systems and Software*, vol. 123, pp. 176–189, 2017.

[9] L. Chen, "Continuous delivery: Huge benefits, but challenges too," *IEEE Software*, vol. 32, no. 2, pp. 50–54, 2015.

[10] S. Gregory, "How common is common enough in requirements-engineering practice?" *IEEE Software*, vol. 35, no. 3, pp. 20–23, 2018.

[11] E. Murphy-Hill, T. Zimmermann, and N. Nagappan, "Cowboys, ankle sprains, and keepers of quality: How is video game development different from software development?" in *36th Int'l Conference on Software Engineering*, 2014.

[12] A. Mori, G. Vale, M. Viggiato, J. Oliveira, E. Figueiredo, E. Cirilo, P. Jamshidi, and C. Kastner, "Evaluating domain-specific metric thresholds: an empirical study," in *Int'l Conference on Technical Debt*, 2018.

[13] I. Richardson, L. Reid, and P. OLeary, "Healthcare systems quality: development and use," in *Int'l Workshop on Software Engineering in Healthcare Systems*, 2016.

[14] K. J. Stol, P. Ralph, and B. Fitzgerald, "Grounded theory in software engineering research: A critical review and guidelines," in *38th Int'l Conference on Software Engineering*, 2016.

[15] A. Strauss and J. M. Corbin, *Basics of qualitative research: Grounded theory procedures and techniques.* Sage Publications, Inc, 1990.

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

[17] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.

[18] P. Stacey and J. Nandhakumar, "A temporal perspective of the computer game development process," *Information Systems Journal*, vol. 19, no. 5, pp. 479–497, 2009.

[19] T. Burger-Helmchen and P. Cohendet, "User communities and social software in the video game industry," *Long Range Planning*, vol. 44, no. 5-6, pp. 317–343, 2011.

[20] B. Dagenais and M. P. Robillard, "Creating and evolving developer documentation: understanding the decisions of open source contributors," in *18th Int'l Symposium on Foundations of Software Engineering*, 2010.

[21] S. Segura, A. B. Sánchez, and A. Ruiz-Cortés, "Automated variability analysis and testing of an e-commerce site.: an experience report," in *29th Int'l Conference on Automated software engineering*, 2014.

[22] D. Russo, P. Ciancarini, T. Falasconi, and M. Tomasi, "Software quality concerns in the italian bank sector: the emergence of a meta-quality dimension," in *39th Int'l Conference on Software Engineering: Software Engineering in Practice Track*, 2017.

[23] G. Fairbanks, K. Bierhoff, and D. D'Souza, "Software architecture at a large financial firm," in *21st symposium on Object-oriented programming systems, languages, and applications*, 2006.

[24] M. Linares-Vásquez, S. Klock, C. McMillan, A. Sabané, D. Poshyvanyk, and Y.-G. Guéhéneuc, "Domain matters: bringing further evidence of the relationships among anti-patterns, application domains, and quality-related metrics in java mobile apps," in *22nd Int'l Conference on Program Comprehension*, 2014.

[25] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. El Emam, and J. Rosen-

berg, “Preliminary guidelines for empirical research in software engineering,” *IEEE Transactions on software engineering*, vol. 28, no. 8, pp. 721–734, 2002.
[26] K. Charmaz and L. L. Belgrave, “Grounded theory,” *The Blackwell encyclopedia of sociology*, 2007.
[27] M. Paixao, J. Krinke, D. Han, C. Ragkhitwetsagul, and M. Harman, “Are developers aware of the architectural impact of their changes?” in *32nd Int’l Conference on Automated Software Engineering*, 2017.
[28] K. Roed and G. Ellingsen, “Users as heterogeneous engineers-the challenge of designing sustainable information systems in health care,” in *44th Hawaii Int’l Conference on System Sciences*, 2011.
[29] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, “Quality and productivity outcomes relating to continuous integration in github,” in *10th Joint Meeting on Foundations of Software Engineering*, 2015.
[30] D. Ståhl and J. Bosch, “Modeling continuous integration practice differences in industry software development,” *Journal of Systems and Software*, vol. 87, pp. 48–59, 2014.
[31] S. Elbaum, G. Rothermel, and J. Penix, “Techniques for improving regression testing in continuous integration development environments,” in *22nd Int’l Symposium on Foundations of Software Engineering*, 2014.
[32] M. Hilton, N. Nelson, T. Tunnell, D. Marinov, and D. Dig, “Trade-offs in continuous integration: assurance, security, and flexibility,” in *11th Joint Meeting on Foundations of Software Engineering*, 2017.
[33] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, “Usage, costs, and benefits of continuous integration in open-source projects,” in *31st Int’l Conference on Automated Software Engineering*, 2016.
[34] P. Jamshidi, C. Pahl, N. C. Mendona, J. Lewis, and S. Tilkov, “Microservices: The journey so far and challenges ahead,” *IEEE Software*, vol. 35, no. 3, pp. 24–35, May 2018.
[35] L. Prechelt, H. Schmeisky, and F. Zieris, “Quality experience: a grounded theory of successful agile projects without dedicated testers,” in *38th Int’l Conference on Software Engineering*, 2016.
[36] A. Begel and T. Zimmermann, “Analyze this! 145 questions for data scientists in software engineering,” in *36th Int’l Conference on Software Engineering*, 2014.
[37] D. Lo, N. Nagappan, and T. Zimmermann, “How practitioners perceive the relevance of software engineering research,” in *10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 415–425.
[38] B. Flyvbjerg, “Five misunderstandings about case-study research,” *Qualitative inquiry*, vol. 12, 2006.
[39] C. Bogart, C. Kästner, J. Herbsleb, and F. Thung, “How to break an api: Cost negotiation and community values in three software ecosystems,” in *24th Int’l Symposium on Foundations of Software Engineering*, 2016.