The Journal of Systems & Software 169 (2020) 110699

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

Evaluating and strategizing the onboarding of software developers in large-scale globally distributed projects

Ricardo Britto [a,b,*], Darja Smite [b], Lars-Ola Damm [a], Jürgen Börstler [b]

[a] Ericsson AB, Ölandsgatan, 371 73, Karlskrona, Sweden
[b] Department of Software Engineering (DIPT)- Blekinge Institute of Technology (BTH), 371 79, Karlskrona, Sweden

ARTICLE INFO

Article history:
Received 29 September 2019
Received in revised form 2 June 2020
Accepted 16 June 2020
Available online 24 June 2020

Keywords:
Onboarding
Global Software Engineering
Large-scale software development

ABSTRACT

The combination of scale and distribution in software projects makes the onboarding of new developers problematic. To the best of our knowledge, there is no research on the relationship between onboarding strategies and the performance evolution of newcomers in large-scale, globally distributed projects. Furthermore, there are no approaches to support the development of strategies to systematically onboard developers. In this paper, we address these gaps by means of an industrial case study. We identified that the following aspects seem to be related to the observed onboarding results: the distance to mentors, the formal training approach used, the allocation of large and distributed tasks in the early stages of the onboarding process, and team instability. We conclude that onboarding must be planned well ahead and should consider avoiding the aspects mentioned above. Based on the results of this investigation, we propose a process to strategize and evaluate onboarding. To develop the process, we used business process modeling. We conducted a static validation of the proposed process utilizing interviews with experts. The static validation of the process indicates that it can help companies to deal with the challenges associated with the onboarding of newcomers through more systematic, effective, and repeatable onboarding strategies.

© 2020 Elsevier Inc. All rights reserved.

1. Introduction

Companies worldwide develop software in a globally distributed manner (Global Software Engineering — GSE) to achieve benefits such as reduced time-to-market and access to skilled people all over the world (Ramasubbu et al., 2011; Conchúir et al., 2009; Bondi and Ros, 2009; Herbsleb and Moitra, 2001). However, geographical, temporal, and cultural distances make coordination and communication more challenging in GSE environments.

GSE projects often involve a large number of people (large-scale projects[1]) and the scale amplifies the distribution-related challenges (Dikert et al., 2016). The combination of scale and global distribution may lead to problems, such as more software defects (Espinosa et al., 2007a), and schedule and budget overruns (Herbsleb and Mockus, 2003).

In addition to the challenges imposed by scale and global distribution, GSE projects are often associated with the development of products with long life cycles (Vierhauser et al., 2014; Eldh et al., 2010). Long life cycles are associated with large amounts of (often complex) legacy code (Vierhauser et al., 2014) and the onboarding of many different developers.

Onboarding of new developers (newcomers) may occur for different reasons: to replace retired developers, to replace developers that left the company for another job, or to increase the existing workforce. Developers may also be relocated within the company and, thus, onboarded to perform different types of tasks that require new competence build-up. While developers may be onboarded at multiple occasions during the life cycle of a large-scale GSE project, the onboarding processes may be more challenging in this context, due to the scale and distribution challenges mentioned above, and the difficulty to learn and change legacy code (Bohner, 2002; Chen and Rajich, 2001).

In our previous study (Britto et al., 2018), we investigated how the onboarding of software developers was carried out in three different globally distributed industrial cases. We learned that the companies employed onboarding strategies that are semi-formalized and that distance, distribution, and legacy code challenge the onboarding of software developers (Britto et al., 2018). However, the performance evolution of offshore newcomers onboarded in large-scale, globally distributed projects and how it

* Corresponding author at: Department of Software Engineering (DIPT)-Blekinge Institute of Technology (BTH), 371 79, Karlskrona, Sweden.
E-mail addresses: ricardo.britto@ericsson.com, ricardo.britto@bth.se (R. Britto), darja.smite@ericsson.com (D. Smite), lars-ola.damm@ericsson.com (L.-O. Damm), jurgen.borstler@bth.se (J. Börstler).

[1] Based on the findings of a systematic literature review, Dikert et al. define large-scale software undertakings as the ones that involve at least 50 human resources – not necessarily only developers, but also other staff collaborating in software development – or at least six teams (Dikert et al., 2016).

https://doi.org/10.1016/j.jss.2020.110699
0164-1212/© 2020 Elsevier Inc. All rights reserved.

relates to the employed onboarding strategies remains an unanswered question.

To address the gap mentioned above, we dove deeper in one of the cases investigated in our previous study (Britto et al., 2018). More specifically, we selected a case from Ericsson. The case is a large-scale software product that has been developed over the last 20 years. We investigated the onboarding of software developers located in a remote site. We aimed at identifying how the performance evolution of the newcomers relates to the employed onboarding strategy. Based on the findings of this investigation and previous research conducted by us, we aimed at developing a process to strategize and evaluate the onboarding of software developers.

In this paper, we address the following research questions:

- **RQ1** — How does the performance evolution of newcomers relate to the followed onboarding strategy in a large-scale globally distributed project?
- **RQ2** — How can companies systematically strategize and evaluate the onboarding of software developers in large-scale, globally distributed projects?

The main contribution of this paper is two-fold:

- Performance implications associated with the onboarding of offshore newcomers in a large-scale globally distributed project.
- A process to support the development of onboarding strategies and evaluation of onboarding results.

The remainder of this paper is organized as follows: Section 2 describes the background and related work. Section 3 presents the research design. Section 4 presents the results of the exploratory study. Section 5 presents the developed process, along with an illustration of its use. The results of a preliminary validation of the process are presented in Section 6. A discussion is presented in Section 7. Validity threats are discussed in Section 8. Finally, a summary of our conclusions and view on future work is provided in Section 9. Note that this paper is an extension of our previous work (Britto et al., 2019). While we have improved our previous work as a whole, the main contribution of this paper is the process that answers question RQ2.

2. Background and related work

In large-scale distributed projects, the onboarding of software developers occurs many times during a project's life cycle. Given the known challenges that are faced by developers in globally distributed software projects, onboarding developers in this context is also very challenging (Britto et al., 2018; Fagerholm et al., 2014a,b; Šmite and Wohlin, 2011). In this section, we present some important concepts related to onboarding and related literature. Note that we have provided a very detailed background on onboarding to facilitate the understanding of the process proposed in this paper.

2.1. Onboarding

Onboarding refers to the mechanism through which newcomers acquire the required knowledge, skills, and behaviors to become effective employees (Bauer and Erdogan, 2011; Van Maanen and Schein, 1979). Klein et al. (2015) affirm that the research on onboarding can be divided into four distinct perspectives:

- Stages through which newcomers progress (Buchanan, 1974; Feldman, 1976).
- Actors involved with the onboarding of newcomers (Morrison, 2002; Ashforth, 2001).
- Tactics and practices employed by organizations for onboarding newcomers (Bauer, 2011; Van Maanen and Schein, 1979; Klein and Heuser, 2008).
- Content to be learned by newcomers during the onboarding (Feldman, 1976; Chao et al., 1994).

Considering that the main focus of this paper is on onboarding tactics and practices and how they relate to the performance evolution of newcomers, we elaborate further on this perspective, describing the main models of onboarding: Van Maanen and Shein's model (Van Maanen and Schein, 1979), Jones' model (Jones, 1986) and Bauer's model (Bauer, 2011).

Van Maanen and Shein's model and Jones' model

Van Maanen and Schein (1979) proposed a theoretical explanation regarding role orientation in the context of onboarding. Jones' model (Jones, 1986) was built upon Van Maanen and Shein's Model (Van Maanen and Schein, 1979).

The original model, proposed by Van Maanen and Shein, categorizes onboarding tactics in six dimensions:

- **Collective vs. individual** — Collective onboarding occurs when a group of newcomers go through onboarding activities and acquire experiences together (e.g., boot camps). Individual onboarding occurs when newcomers go through separate from other newcomers (e.g., apprenticeship).
- **Formal vs. informal** — Formal onboarding relates to tactics in which newcomers are segregated from other employees. In contrast, informal onboarding relates to tactics that have no or little separation between newcomers and other employees.
- **Sequential vs. random** — Sequential onboarding refers to the extent to which discrete steps regarding the onboarding phases are specified for the newcomers. In contrast, random onboarding tactics do not specify any sequence of steps.
- **Fixed vs. variable** — Fixed onboarding occurs when there is a timetable (fixed time) associated with each step of the onboarding process. In contrast, in variable onboarding, there is no time associated with the onboarding steps.
- **Serial vs. disjunctive** — Serial onboarding takes place when experienced employees serve as models for newcomers. In contrast, disjunctive onboarding refers to the tactics wherein no guidelines or models are provided to newcomers.
- **Investiture vs. divestiture** — Investiture onboarding takes place when the newcomers should keep their personal characteristics (own skills, values, and attitudes). In contrast, divestiture takes place when an organization rejects and removes the personal characteristics of newcomers.

Jones (1986) reduced the original six dimensions of Van Maanen and Shein's Model (Van Maanen and Schein, 1979) to two:

- In **institutionalized** onboarding, there is a structured program, and newcomers receive formal orientation and mentoring. This dimension merged the following dimensions from the original model: collective, formal, sequential, fixed, and serial investiture.
- In **individualized** onboarding, newcomers start working directly and learn the norms, values, and expectations on-the-fly. This dimension combines the following dimensions: individual, informal, random, variable, disjunctive, and divestiture.

Companies considered as successful regarding the onboarding of newcomers have more formal onboarding programs (institutionalized onboarding) (Klein and Heuser, 2008; Bauer et al., 2007; Cable and Parsons, 2001).

Bauer's model

Bauer et al. proposed an onboarding model based on a series of empirical studies (Bauer, 2011; Bauer and Green, 1994; Bauer et al., 1998; Bauer and Erdogan, 2011; Bauer et al., 2007). It supports the design of onboarding programs, capitalizing on the fact that institutionalized onboarding is more successful than individualized onboarding (Klein and Heuser, 2008; Bauer et al., 2007; Cable and Parsons, 2001).

Bauer's model has a finer grain level than the previous models than Van Maanen and Shein's and Jones' models; it aggregates practices, techniques, methods, and technologies (functions) that are related to successful onboarding. We used Bauer's model as part of the process proposed in this paper.

According to Bauer, onboarding has four distinct levels (Four Cs), which are the building blocks of successful onboarding (Bauer, 2011):

- **Compliance** is related to teaching employees basic legal and policy-related rules and regulations.
- **Clarification** is related to ensuring that newcomers understand their new jobs and related expectations.
- **Culture** is related to providing newcomers with a sense of organizational norms, including both formal and informal.
- **Connection** is related to the interpersonal relationships and information networks that newcomers must establish.

The extent to which an organization focuses on each C determines its onboarding strategy. The combination of functions (tools, practices, recommendations, performance goals, and measurement milestones) constitutes an onboarding strategy, which is often formalized in an onboarding plan (Bauer, 2011). The success of an onboarding strategy is related to short-term and long-term outcomes. Short-term outcomes are associated with the adjustment of new employees to their new jobs (Bauer, 2011). Short-term results are related to high job satisfaction and low turnover (Bauer et al., 2007). Long-term outcomes of onboarding are related to attitudes and behaviors. Long-term successful onboarding is related to higher job satisfaction, organizational commitment, low staff turnover, high performance levels, career effectiveness, and low stress levels (Bauer, 2011; Maier and Brunstein, 2001; Meyer and Allen, 1988).

Bauer (2011) categorizes the onboarding functions as follows:

- **Recruiting** — In many organizations, recruiting is not integrated with the onboarding plans and is treated as a separate function. However, existing literature (Bauer, 2011; Bauer and Erdogan, 2011) shows that integration (e.g., through realistic job previews or early involvement of stakeholders) gives candidates more accurate information about the company and the job. As a result, functions of this category facilitate the adjustment of new employees, especially self-efficacy, role clarity, and knowledge of culture (Klein et al., 2006).
- **Orientation** — Formal orientation programs help newcomers to understand important aspects of their jobs and organizations, as the company's culture and values (Klein and Weaver, 2000). Moreover, they also help newcomers feel welcome by presenting them to other individuals within the organization. Computer-based orientation programs can help to keep consistency in the program in different locations.
- **Support tools and processes** — Tools and formal processes are of great value for onboarding success. According to Bauer (2011), there are three tools/processes that are related to successful onboarding: a written onboarding plan, stakeholder meetings, and onboarding online.
- **Coaching and support** — Mentors can teach newcomers about the company, provide advice, and help with job instruction. According to existing research, new employees with mentors acquire more knowledge about their new company than the ones without mentors (Ostroff and Kozlowski, 1993). Moreover, mentoring programs and opportunities for informal interaction with colleagues help the new employees to adapt more easily to the new work environment.
- **Training** — Training is essential to give the newcomers the confidence, clarity, and skills required for their job. Newcomers can receive training in hard skills and soft skills.
- **Feedback** — Newcomers need regular feedback and guidance to understand and interpret the reactions of their coworkers. Feedback can be mainly provided in two ways (Bauer, 2011): performance appraisals and 360-degree feedback, wherein the new employees are evaluated and receive developmental feedback; and employee-initiated information and feedback-seeking, wherein the new employees proactively seek feedback.

2.2. Onboarding in globally distributed projects

Studies with some relationship with the onboarding of newcomers in globally distributed projects often focus on comparing offshore teams with the original experienced developers in the prime development location. Time to acquire the required knowledge to perform as expected is of particular interest for companies that make decisions to onboard developers and teams in offshore locations. Mockus and Weiss (2001) studied individual developers working on non-trivial modification requests and found that offshore developers may reach full productivity in approximately 15 months, after three months of project training before the actual work. Perception-based studies and experience reports suggest that the learning process may take from 12 months (Ebert, 2007), up to three (Šmite et al., 2015) and five years (Šmite and Wohlin, 2012; Kommeren and Parviainen, 2007) or even longer (Boden et al., 2007), leading to longer onboarding than in collocated projects.

Most research related to onboarding/team performance either focuses only on the performance of teams in distributed projects or on how developers are onboarded, but not in how both things relate to each other or how developers can be onboarded systematically.

Researchers have compared productivity or quality performance of collocated and distributed teams, as summarized by Nguyen-Duc et al. (2014). The results of their study suggest that geographical dispersion harms team productivity (Espinosa et al., 2007b) and software quality (Cataldo and Nambiar, 2009; Jabangwe and Šmite, 2012; Jabangwe et al., 2013). Another relevant finding of studies in this line of research is that it may take long periods for developers in offshore sites to achieve full productivity (Mockus and Weiss, 2001; Šmite and van Solingen, 2016)

Some studies focused on the role of mentoring when onboarding newcomers. They identified that this practice relates to more active newcomers when they start contributing to open source projects (Fagerholm et al., 2014b,a), although it may not be enough to ensure they provide to those projects in the long run (Labuschagne and Holmes, 2015). It may also support overcoming many challenges that are faced by newcomers when they start contributing to open source projects (Steinmacher et al., 2015; Steinmacher and Gerosa, 2014). Mentoring and support are also related to successful onboarding (Sharma and Stol, 2020). It was identified as indispensable to onboard newcomers on offshore locations in globally distributed projects with a high

amount of complex legacy code (Britto et al., 2016a, 2018). Mentoring and peer-learning was also identified as an effective practice to avoid the isolation of newcomers, improving their job satisfaction (Johnson and Senges, 2010).

Another vein of research focuses on holistically researching onboarding. Empirical evidence shows that some onboarding practices are planned and executed locally, making it hard to avoid onboarding strategy fragmentation in projects with multiple sites (Britto et al., 2018).

In summary, existing literature shows that:

- Newcomers face many barriers before they can start working.
- To support overcoming the challenges faced by newcomers, mentoring seems to be a good practice as it is related to successful onboarding. It also promotes learning, which is essential for newcomers to achieve proper levels of performance.
- Together with practices such as mentoring, literature shows that more time is required to onboard newcomers in globally distributed projects, and it may be hard to keep control of how newcomers are onboarded in projects with many sites.
- There are no approaches to support the systematic development of onboarding strategies in globally distributed projects.

Although existing literature suggests that onboarding of developers in globally distributed projects is more challenging than in other contexts (as detailed above), many of the studies related to performance in global software projects have focused on comparing distributed teams to collocated teams. In contrast, the number of longitudinal studies of performance evolution of newcomers onboarded in remote locations is still scarce. Moreover, none related work investigated the implications of the employed onboarding strategies to the performance evolution of newcomers in large-scale, globally distributed projects. Finally, there are no approaches to support the development of strategies to onboard developers/teams and systematically evaluate onboarding results, which is found as very important to be successful when carrying out the onboarding of other roles (e.g., managers) (Bauer, 2011). This paper fills these gaps. Finally, The context of large-scale software projects involving a large amount of legacy code adds an original perspective, which has not been addressed explicitly in GSE onboarding/performance-related studies before.

3. Research design

To address RQ1, we have conducted an exploratory longitudinal case study (Runeson et al., 2012). To address RQ2, we employed business process modeling (BPM) to create a process. In this section, we describe the case and the research design of our investigation.

3.1. The case and unit of analysis

We selected the case investigated in this paper through convenience sampling in consultation with Ericsson representatives as a case suitable to answer our research question. The other cases investigated in our previous study (Britto et al., 2018) were not included in this investigation due to the lack of quantitative data.

The selected case and unit of analysis is a large-scale distributed endeavor associated with the development and maintenance of a large telecommunication software product in Ericsson. Its degree of distribution increased significantly during the period covered in our investigation. The product originated in Sweden and has evolved for more than 20 years; it comprises a considerable amount of complex legacy code. It was subject to many technical and methodological changes over the years (e.g., the introduction of agile practices).

Offshore locations were added to address the growing demands for resources and to implement market-specific customizations (USA, Italy, China, Turkey, India, and Poland). The developers in India and Poland were onboarded later in the product life cycle. The sites in China and Turkey were closed down due to business reasons and are not included in the analysis. The company transferred the primary responsibility for the product to India in 2017.

The software development teams are cross-functional and use agile practices in their daily work (e.g., daily stand meetings, sprints, and continuous integration). The teams have four to seven developers and receive tasks that can be product customizations, maintenance, product improvements, standardizations of market features, and business use cases. Each task resembles an independent project with a specific start and end date and expected results. One or more development teams are assigned to each task.

We based our analysis on a subset of the collected data, focusing on product customization tasks carried out by newcomers located in India. We chose product customization tasks mainly because it was the only task type for which we were able to track performance (productivity and autonomy), due to the availability of the data. External customers drive product customization tasks, and this demands a stricter effort/cost control by Ericsson, i.e., the associated data is audited regularly. Furthermore, Product customization tasks are adequate for this study because the newcomers mainly work with this type of task (90% of the work done by the newcomers at the time of our investigation).

We also used data from experienced teams located in the USA and Italy as a benchmark and to establish the performance goals for the newcomers. We did not include tasks carried out by newcomers from other locations due to two reasons: (i) The newcomers located in Poland never worked with product customization tasks; (ii) The teams located in Sweden, Italy, and the USA had been onboarded too long ago. This made it difficult to collect reliable data since part of our measurement approach depended on expert knowledge; people involved with those teams either had left Ericsson or were not able to recall sufficient details regarding the product customization tasks.

The newcomers onboarded in India had worked in the case for approximately two years by the time we finalized the data collection process. The newcomers were onboarded in three main waves: 11 newcomers in August 2014, seven newcomers in January 2015, and 10 newcomers in August 2015. Due to this multi-phased onboarding, the product customization tasks were never fulfilled by the same team of developers; the tasks were always conducted by an ad-hoc team formed by the available developers at a given point in time. Thus, we analyzed the tasks in chronological order to identify how the performance of the newcomers evolved. Note that while the three onboarding waves increased the total number of developers, it also replaced developers that were either reallocated to other products or left the company during the investigated period.

Fig. 1 presents the onboarding strategy employed to onboard the newcomers in India. Given the distributed nature of the project, some parts of the employed onboarding strategy were organized by the central project stakeholders in Sweden. In contrast, others occurred remotely and were handled by the local management.

The recruitment of the new employees was organized by the Indian site individually, while the demanded skill profiles were formulated in Sweden. The company does not provide realistic

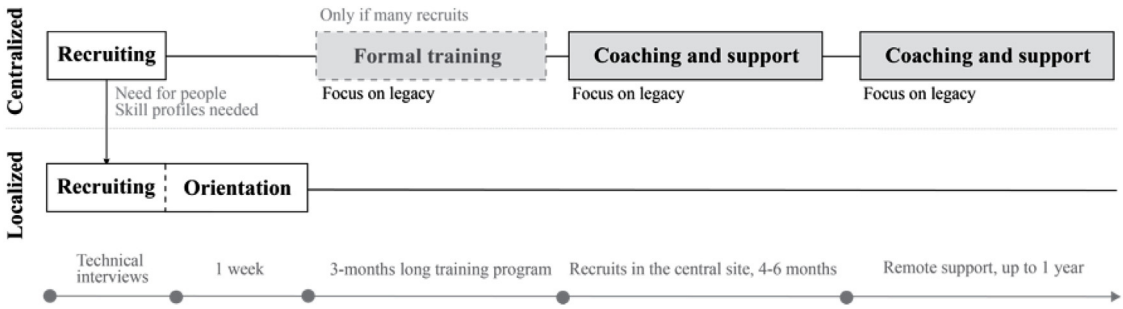

Fig. 1. Summary of the employed onboarding strategy (Britto et al., 2018).

job previews; however, senior developers (if any) participate in the recruitment process and the evaluation of new candidate developers.

The orientation of new developers is carried out informally, i.e., there is no formal orientation program. In general, the newcomers are given one week at the new job to familiarize themselves with the new environment, get to know the key people and co-workers (socialization), and acquire the basics about the existing ways of working. The way it is done depended on the number of people being onboarded: individual orientation when a few developers (less than three) are onboarded and group level orientation when many developers are onboarded (the case in the three main onboarding waves).

The employed learning process is mainly based on autonomous learning. The new developers receive, on average, three months of formal training about the product. Most of the learning is based on doing actual work. Senior developers, mainly software architects located in Sweden, are assigned as mentors (Britto et al., 2016a). The first wave of newcomers received training and mentoring on-site in Sweden for the initial five months. The second wave received training and mentoring in India for four months from two visiting senior developers from Sweden. Finally, the third wave just received remote mentoring, provided by software architects located in Sweden.

The company also uses other resources to make the onboarding of new developers successful. For each developer, an Excel spreadsheet was used to track their progress regarding the competence they must acquire. This spreadsheet also contains the primary source of knowledge they can use to obtain the required expertise. It is also used by immediate managers and mentors to follow the progression of new developers. Another tool is the corporate intranet, wherein documents about the ways of working and the product are available, and which is maintained centrally.

The newcomers receive continuous feedback on their work outcomes through code reviews. Local senior developers (if any) and software architects located in Sweden not only evaluate the performance but also transfer product knowledge to support the new developers. Based on the received feedback, newcomers may require more formal training. At the same time, the status of performance is used by the immediate managers locally to identify whether or not a newcomer needs more support. Such checks are performed together with the mentors weekly. Code reviews also serve as a track record used in permanent employment considerations.

3.2. Variables

In an ideal scenario, successfully onboarded software developers should be able to develop software autonomously, delivering code efficiently and effectively. All of this should be achieved in the shortest period possible. Thus, to investigate the performance evolution of newcomers holistically, one should look at how the productivity, autonomy, cost efficiency, and deliverables' quality associated with newcomers evolve. In our case, the available data only enabled us to focus on the productivity and autonomy of the newcomers.

In this paper, we used the measurement approach detailed in Britto et al. (2016b), which evolved during this investigation. We accounted for the following variables in our investigation (all variables use a ratio scale):

- **Task size T** is measured in complexity points and takes into account a task's extent and complexity to make tasks of different degrees of complexity easier to compare (Unit: Complexity Points). T_i is the size of task i and T_{ki} is the size of the proportion of task i that was completed by team k. Note that task size is measured using a group estimation process, which involves senior software architects.
- **Team productivity P** represents the effort that a team spends to complete a complexity point (Unit: Complexity Points/1000 Hours). P_{ki} is the productivity of team k on task i:

$$P_{ki} = \frac{T_{ki}}{E_{ki}} \tag{1}$$

  where E_{ki} is the total effort spent by team k on task i. The effort is obtained from time reports.
- **Team autonomy A** represents how independently a development team fulfills a task (Unit: Complexity Points/Hours). A_{ki} is the autonomy with which team k carried out task i:

$$A_{ki} = \frac{T_{ki}}{M^h_{ki} + 1} \tag{2}$$

  where M^h_{ki} is the effort in work hours spent by software architects providing mentoring and help to team k during the fulfillment of task i. The effort was obtained from mentoring time reports. Note that 1 is added to the denominator of Equation to avoid division by zero (numerical stability).

3.3. Data collection

In total, we collected data from 24 product customization tasks related to the developers onboarded in India.[2] Furthermore, we collected data related to nine product customization tasks carried out by mature developers located in the USA (4) and Italy (5), for benchmark purposes. The data encompasses tasks carried out between August 2014 (inception of the Indian site) and October 2016. To collect the data, we conducted semi-structured interviews, archival research, and workshops. All interviews and workshops were carried out in person in Sweden, except for one

[2] Note that we cannot share the data collected and analyzed in this paper under the non-disclosure agreement signed by the authors.

Table 1
Distributed product customization tasks.

Distributed task	Main location	Maturity level	Collaboration setup	Locations involved
Task 1	India	Immature	Two independent teams	India and Sweden
Task 3	India	Immature	Two independent teams	India and Italy
Task 9	India	Immature	Virtual team with developers located in India and one software architect located in Sweden	India and Sweden
Task 17	India	Immature	Virtual team with developers located in India and one software architect located in Sweden	India and Sweden
Task 18	India	Immature	Virtual team with developers located in India and one software architect located in Sweden	India and Sweden

Table 2
Archival research data sources — part I.

Data source	Description	Variables
Task time report spreadsheets	Documents that contain the developers and design leads involved in finished tasks, in addition to the actual effort spent by them to carry out the tasks.	We extracted team setup, number of developers, and task effort (E_i). We used this data to calculate team productivity (P_i) and to support the identification of a team's contribution to distributed tasks.
Mentoring time report spreadsheets	Documents that contain the architects and the hours they spent to support teams during the fulfillment of tasks.	From this type of document, we extracted the architects involved with each task and the respective mentoring hours (M_{ki}). We used this data to calculate team autonomy (A_i).
Slot-plan spreadsheets	Documents that contain the planning information related to developers assigned to particular work items. These spreadsheets helped to identify the start and end dates of the tasks.	We used slot-plans to confirm the team setup identified through the time report spreadsheets.

interview conducted with a senior developer located in the USA (via Skype).

Five of the 24 tasks carried out by the newcomers involved the participation of mature developers from other locations. The other 19 tasks were only fulfilled by the newcomers in India with mentoring provided by senior developers or software architects from Sweden. The mentoring was mainly provided by means of design workshops and code reviews. Table 1 describes the collaborations involved in the five distributed product customization tasks.

Archival research

A large amount of the data used in our investigation was collected through archival research. The mined data sources, with associated metrics and variables, are described in Tables 2 and 3. We aggregated all the data, which was verified through workshops and interviews.

Table 3
Archival research data sources — part II.

Data source	Description	Variables
Human resource management repository	Repository that contains information about personnel involved with the selected case.	From this data source, we extracted data regarding the experience of developers and software architects.
Code repository	Configuration management is carried out using Git in the selected case.	We extracted lines of code associated with each task, which were used to support the sanity check carried out regarding task complexity (C_i).
Product architecture specification	Document that contains the description of the product architecture and its main components.	It was used to support the measurement of task complexity (C_i).
Solution specifications	Documents that contain details about solution design.	It was used to support the measurement of complexity (C_i).

Workshops

We conducted six workshops to assess the complexity of the investigated tasks and verify (sanity check) the data collected through archival research. The workshops took place in 2016, between January and October. Six to eight software architects attended them. Each workshop took approximately 1.5 h. The participants' experience in the investigated product ranged from five to 15 years. All workshops were conducted jointly by the first and third authors of this paper.

In the first workshop with the architects, we defined and piloted the process to measure task complexity. It was consensus among the software architects that existing complexity metrics (e.g., cyclomatic complexity and lines of code) do not account for all the complexity related to product customization tasks in this product. Together with the software architects, we defined the following process to measure task complexity:

1. At the beginning of each workshop, the software architects selected one task, and the other tasks were to be measured concerning it.
2. For each task, the software architects attributed a positive integer number (complexity points) using a planning poker-based approach.
3. To check the assigned measures, the software architects consulted solution specification documents, the product architecture specification, and the product source code whenever needed.

The task complexity measurement took place from the second to the sixth workshop with the software architects. The results were recorded in a spreadsheet.

In the sixth workshop with the architects, we also conducted a data sanity check to validate both the assessed task complexity and the plots about performance evolution. It was based on summaries and plots of the collected data, including lines of code and task complexities. We asked the architects to provide explanations for the outliers. They realized they had not accounted for non-functional testing in two product customization tasks. As a consequence, they increased the amount of task complexity points associated with these two tasks. We kept notes about the developers' comments on the performance evolution results.

An additional workshop took place with the participation of five developers onboarded in India. The workshop took one hour and 20 min and was held in November 2016. We conducted this workshop with developers who spent three months in Sweden

as part of the fulfillment of one product customization task. The experience of the participants in the investigated product ranged from 1.5 to 2.5 years.

We asked the participants to provide information about the challenges they faced to achieve high performance. The developers were asked to write down, independently of each other, the problems that, in their opinion, impacted their learning processes. After 10 min, individual ideas were discussed within the group.

Semi-structured interviews

We conducted individual semi-structured interviews with three software architects in 2016 between April and October. Each interview took approximately one hour. The interviewees had from 5 to 15 years of experience in the investigated product. The first author of this paper conducted all interviews.

The main objectives of the interviews were:

- To identify teams' contribution to the distributed product customization tasks', and how much mentoring and help these teams received by software architects. We used this information to adjust T_i for tasks that involved several teams (T_{ki}).
- To collect additional information about the challenges the newcomers faced to achieve high performance (high productivity and autonomy).

We also conducted one semi-structured interview via Skype with a senior developer located in the USA. He had more than ten years of experience in the investigated product by the time of our investigation. The interview took 30 min and was conducted in September 2016. The main goal of this interview was to get the view of an experienced developer about the challenges newcomers face to achieve high performance in the product under investigation.

In all interviews, we took notes to facilitate post-interview analysis. The notes included key points related to the questions that were posed during the interviews. The notes were discussed with the respective interviewees to ensure that they reflected what was discussed during the interviews.

3.4. Data analysis

The data analysis involved quantitative and qualitative methods. To quantitatively evaluate the performance evolution of the newcomers, we employed Locally Weighted Scatterplot Smoothing (LOWESS) plots (Burchell and Vargas, 2016). This approach is adequate to identify trends in a dataset with a small number of observations than traditional linear regression analysis, which demands a considerable amount of observations (Green, 1991; Maxwell, 2000). Furthermore, we employed a Wilcoxon–Mann–Whitney and the Vargha–Delaney A measure (Neumann et al., 2015) to test for statical differences and calculate the effect sizes, respectively.

To analyze the qualitative data from the workshops and interviews, we followed the coding process described by Robson and McCartan (open coding) (Robson and McCartan, 2016).

We conducted three iterations of coding. In the first iteration, we employed a deductive approach, i.e., we used the functions described in Bauer's model (see Section 2) to link the employed onboarding strategy and the observed performance evolution of the newcomers. This was done by coding the interviews and workshop notes, which resulted in identifying two onboarding functions (mentoring and training).

In the second iteration, we screened the notes again to identify issues associated with the functions identified in the first coding iteration. As a result, we identified one item associated with mentoring (distance) and two problems related to training (expectation misalignment, on-the-job training with complex tasks, and on-the-job training with distributed tasks).

In the third iteration, we combined the initial onboarding function codes with the issue codes identified in the second iteration. The final result includes the three issues reported in this paper (the distance to mentors; the formal training approach used, which did not fit the socio-cultural background of the newcomers; allocation of large and distributed tasks in the early stages of the onboarding process). Note that a fourth issue (team instability) was identified during the quantitative analysis.

All the iterations were conducted by the first author and reviewed by the second author.

3.5. Business process modeling and static validation

To answer RQ2, we packed the findings from our exploratory study reported herein together with the findings and solutions reported in our previous research (Britto et al., 2016c,b,a, 2018). To develop the process, we used BPM (Holt, 2009). The resulting process provides a systematic way to strategize onboarding undertakings and evaluate onboarding results.

BPM is the activity of representing the business processes of an organization (von Rosing et al., 2014). A business process is defined as the combination of related activities to achieve a specific goal (e.g., product or service). A business process can be used to improve or facilitate the learning of existing processes. According to von Rosing et al. there are three main types of business processes (von Rosing et al., 2014):

- **Management processes** — Processes such as corporate governance, and strategic management, which govern the operation of an organization.
- **Operational processes** — Processes like purchasing, manufacturing, marketing, and sales, which are the core business of an organization and create the primary value stream.
- **Supporting processes** — Processes such as accounting, recruitment, and onboarding, which support the core business of an organization. This is the type of process proposed in this paper.

To carry out the process modeling, we followed these steps:

1. From our previous research (Britto et al., 2016c,b,a, 2018)[3] and the exploratory study reported herein, We identified relevant activities to support the development of onboarding strategies and evaluation of onboarding results.
2. We identified additional activities or artifacts required to enable a coherent process flow.
3. We combined all activities, defining the appropriate sequence for their execution and resulting artifacts.
4. We received informal feedback about the process from a domain expert (an Ericsson R&D manager with more than 15 years of experience).
5. We conducted a static validation (Gorschek et al., 2006) of the process by asking experienced Ericsson R & D managers to answer a questionnaire.

In Ericsson, there are two main types of managers: R&D managers and project managers. We selected R&D managers to participate in our static validation because they are the ones responsible for planning and executing the onboarding of newcomers.

To conduct the static validation, we first presented the process to 20 R&D managers (1-h session). Then, we gave a questionnaire to the managers. Each manager had two weeks to provide their

[3] Note that these studies were conducted in the order presented here.

own answers, although some of the managers took more than two weeks to provide their input. In the end, nine managers from three different Ericsson products answered the questionnaire. All nine respondents had more than ten years of experience.

The questionnaire had the following questions:

- **Q1** — Provide your opinion on each of the components of the process with respect to their relevance when onboarding new developers. Mark your choice by writing "Yes" in the cell of your choice.
- **Q2** — In your opinion, what are the benefits of using the proposed process? *(Open-ended question)*
- **Q3** — What are the drawbacks/limitations of the process? How can we improve/revise the process? *(Open-ended question)*
- **Q4** — Are there steps that should be removed from the process? If yes, then kindly list those items here and also state why do you recommend removing them? *(Open-ended question)*

Component	Strongly agree	Agree	Undecided	Disagree	Strongly disagree
Define onboarding context					
Strategize onboarding using adequate practices					
Evaluate onboarding results					
Store and use knowledge from previous onboarding undertakings					

We calculated the frequency of the answers associated with each process component's relevance (Q1). We evaluated the responses related to the open-ended questions and selected relevant quotes. The results are presented in Section 5.

4. Results and interpretation of the exploratory case study — RQ1

In this section, we present and interpret the results associated with RQ1. We first show the findings related to the newcomers' performance evolution. Second, we show how the observed performance evolution results relate to the employed onboarding strategy.

4.1. The newcomers' performance evolution

Productivity

The productivity goal for the newcomers was defined based on the productivity of senior software developers located in the USA and Italy. The results show that the newcomers never reached the productivity goal. The maximum productivity achieved by the newcomers was **46.44** complexity points per 1000 h (CP). On average, the senior developers were **3.57** times more productive than the newcomers. i.e., the maximum observed productivity of the senior developers (**120.07** complexity points per 1000 h) was **2.59** times bigger than the maximum observed newcomers' productivity. Even the minimum observed senior developers' productivity (**58.41** complexity points per 1000 h) was bigger than the maximum newcomers' productivity.

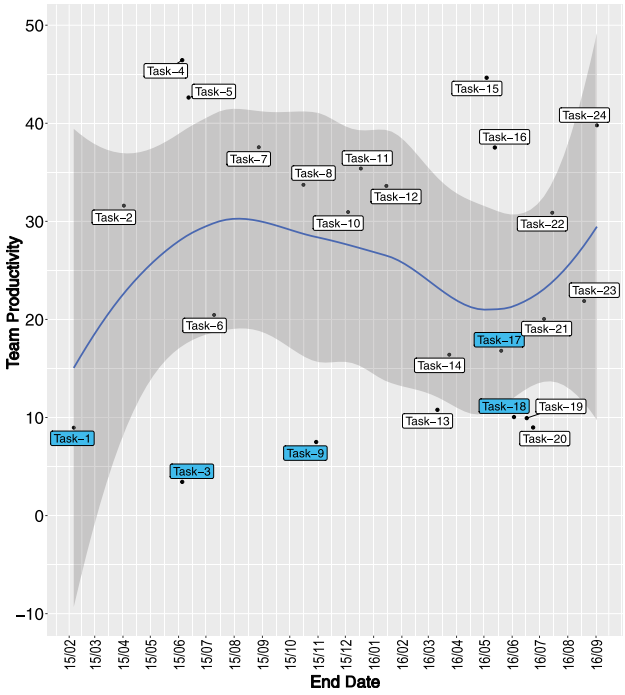

Fig. 2. Productivity evolution for the newcomers (24 tasks). The tasks highlighted in blue involved more than one development site (see Table 1 located in Section 3.3 for more details). The gray area means the confidence interval associated with the line generated using LOWESS (see Section 3.4 for more details).

Fig. 2 shows how the newcomers' productivity evolved during the investigated period. The *x*-axis shows the delivery date for each investigated product customization task. In contrast, the *y*-axis shows the productivity for the respective tasks,[4] (in CP).

Although the productivity varied a lot, it is possible to see that it increased initially and peaked after about six months (August 2015–1508). The productivity then decreased gradually during about eight months to finally return to the top-level of the first year at the end of the second year.

It is possible to see in Fig. 2 that the newcomers had their best performance when they were not required to work on tasks in collaboration with teams located on other sites. The results show that the average productivity of the newcomers in those tasks is **3.11** (median **3.52**) times higher than their productivity in tasks wherein they collaborated with teams located on other sites (tasks 1, 3, 9, 17 and 18, blue in Fig. 2).

We applied the Wilcoxon–Mann–Whitney test and identified that the difference mentioned above is statistically significant (p-value $= 1.37E-03$). We also used the Vargha–Delaney A measure to calculate the effect size of distribution on productivity. The resulting A measure is equal to **0.94**, i.e., 94% of the time, the newcomers had a better productivity in tasks where they did not collaborate with teams located on other sites.

Autonomy

The autonomy goal for the newcomers was defined based on the autonomy of senior software developers located in the USA and Italy (autonomy in relation to the main site of the case, which was located in Sweden). The results show that the newcomers never reached the autonomy goal. On average, the

[4] In Figs. 2 and 3, the tasks with a blue label (1, 3, 9, 17 and 18) were conducted in a distributed fashion. They are described in more details in Table 1.

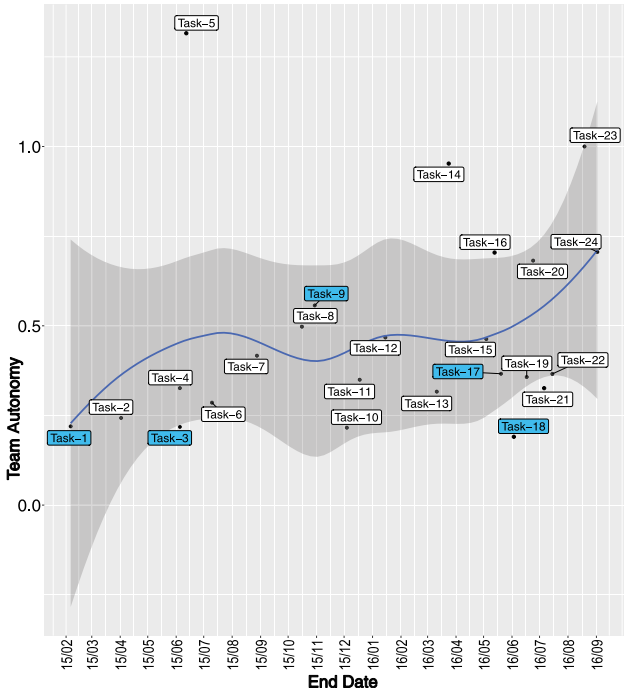

Fig. 3. Autonomy evolution of the newcomers (24 tasks). The tasks highlighted in blue involved more than one development site (see Table 1 located in Section 3.3 for more details). The gray area means the confidence interval associated with the line generated using LOWESS (see Section 3.4 for more details).

senior developers are **1.67** times more autonomous than the newcomers.

Although the newcomers did not evolve much concerning their productivity, they became more independent. This is showed by the maximum observed autonomy achieved by them (**1.32**), which was very close to the maximum observed autonomy of the senior developers (**1.46**).

Fig. 3 shows how the newcomers' autonomy evolved during the investigated period. The *x*-axis shows the delivery date for each investigated product customization task, while the *y*-axis shows autonomy for the corresponding tasks. Compared to Fig. 2, we can see that the variation of autonomy over time is smaller than the change in productivity. We can observe an initial increase in autonomy. After that, we have a year-long period of stagnation. In the last five months (from June 2016 onwards), we can see a further increase in autonomy.

In relation to how the tasks were fulfilled (with or without global distribution), there is no apparent difference regarding autonomy in Fig. 3. The results show that the average autonomy of the newcomers in tasks without global distribution is **1.69** (median **1.90**) times higher than in globally distributed tasks (tasks 1, 3, 9, 17 and 18, blue in Figure). However, after applying the Wilcoxon–Mann–Whitney test, we identified that this difference is not statistically significant (p-value = **0.1**).

4.2. The relationship between the observed performance evolution and the employed onboarding strategy

As mentioned above in this section, the newcomers did not achieve the established performance goals; the observed productivity was far from the target, while the perceived autonomy evolved more than the productivity and got closer to the respective goal.

As shown in Section 3, the Indian site started with 11 developers in August 2014. In January 2015, seven developers were added (a total of 18 developers), followed by another ten developers in August 2015 (a total of 28 developers). Furthermore, ten developers were added to replace developers that left the project (three left in the first half of 2015 and seven in the second half of 2015). The onboarding of new developers meant that there were a large number of newcomers who needed training and mentoring, which may explain why the productivity of the newcomers did not evolve as expected.

A factor that seems to have amplified the challenges imposed by staff turnover is the employed mentoring approach. The most substantial productivity improvement occurred when the first wave of newcomers was trained and mentored on-site in Sweden (between August 2014 and January 2015) and when two senior developers from Sweden traveled to India to provide on-site mentoring in India (between February and May 2015). From June 2015 onward, the newcomers received only remote mentoring by software architects located in Sweden. This matches the period wherein their productivity goes down in Fig. 2. While remote mentoring is better than no mentoring, the results show that the newcomers performed best when they received on-site mentoring.

Regarding autonomy, at first glance, the stagnation period seems counter-intuitive. It is fair to assume that the addition of new developers, together with remote mentoring, should have called for more mentoring hours because of the inefficiency of remote communication. In contrast, we found that the newcomers perceive the mentoring functions less accessible and often do not reach out for help that often. This might also explain why productivity went down since the knowledge gap that was compensated by the short-term involvement of software architects later consumed more effort from the newcomers.

So, how does the observed performance evolution relates to the employed onboarding strategy? We learned that the choice of having **remote mentoring** seems to have negatively impacted performance evolution (especially productivity). A software architect confirmed this: *"I believe that the reason the newcomers in India are struggling is that the product is complex to learn, and it is tough for us to help them just using code reviews, emails, and Skype"*. A newcomer also mentioned this as a challenge: *"Since we didn't get that much training about the product before starting working, we have to keep learning stuff. The two primary sources we have are the software architects and documentation. However, the architects are in Sweden, which complicates the communication with them, and the documentation sometimes does not have everything, and it is hard to find the required knowledge"*.

The software architects provided a large amount of feedback to the newcomers through code reviews. They did this to mitigate the challenges of communicating over a distance. However, as a newcomer commented, geographical distance and temporal distance affect how code review is operationalized: *"We learn a lot from the feedback we get from code reviews. However, sometimes it takes quite a while to get the feedback, because the experienced reviewers are located either in Sweden or the USA"*.

The use of code reviews instead of rich communication media also affected the intensity of connection between the newcomers and software architects. We found that videoconferencing is used only for critical tasks and during critical phases. Status meetings were held twice per week for high priority tasks and to deal with task software design matters. A newcomer highlighted that he would like to have more videoconference meetings to support their onboarding: *"I believe that we should have more videoconferences. For example, during the time we spent in Sweden, things worked much better because we could talk to the architects any time to clarify any doubt about the product. More frequent videoconferences would allow us to have more frequent feedback"*.

Another aspect of the employed onboarding strategy seems to have also negatively impacted performance evolution: **The poor fit of the formal training and the newcomers' expectations**. The main site of the case (located in Sweden) employs a learning process mainly based on autonomous learning, i.e., newcomers do not have the entire required knowledge before starting real work. While this approach worked well in Sweden, the USA, and Italy, it seems to challenged the newcomers onboarded in India. Although the teams located in India received more formal training than the teams located in other locations (e.g., USA, Sweden, and Italy), their general perception is that it was still insufficient.

One possible explanation for this could be rooted in the cultural differences (Nicholson and Sahay, 2001). The learning-by-doing approach used in during the onboarding of newcomers expects pro-active learners and might have clashed with the expectations of the Indian newcomers. This misalignment or perceived "lack of training" seems to harm their performance evolution as mentioned by one of the newcomers: *"We just get a very short training about the product before starting doing the work. It is more like a helicopter view, and we just learn something when we start working. In addition, we don't get enough training in the used technologies. This for sure impacts how we learn and how we work"*.

An aspect that apparently amplified the problems mentioned above is the amount and complexity of the legacy code associated with the product developed in the investigated case. The product includes several subsystems and comprises millions of lines of code, including a significant amount of legacy code that was accumulated during more than 20 years of development. Most interviewees emphasized that the product size and complexity makes it challenging to learn and to work with. For example, a senior developer located in the USA site mentioned: *"The product is very complex. Although I have been working in this product for more than ten years, there are parts that I still don't have enough knowledge about to do things independently"*. One of the newcomers also highlighted the product complexity: *"We have been working with the product for more than two years and we just managed to have a reasonable understanding regarding just one of its main modules"*.

In addition to legacy code complexity, the product also involves many different technologies, which in some cases are company-specific. This means that the newcomers also needed to deal with a complex set of technologies, making the process of acquiring the required knowledge particularly tricky. One newcomer pointed out that because some of these technologies are Ericsson-specific, it reduces the sources from where newcomers can acquire the required knowledge (e.g., it is not possible to use commonly used sites like Stack Overflow[5] to learn about these technologies): *"The product involves a huge technology stack. Besides, some of the used technologies do not have comprehensive support from the community outside Ericsson. This means that we depend only on internal people to learn about this type of technology"*.

The innate complexity of the product and the strategy of **involving early on the newcomers in some large complex globally distributed tasks** seems to have made the onboarding process more difficult, contributing to the observed performance evolution.

Finally, another aspect that also seems to have negatively affected the newcomers' performance evolution was the decision of **not keeping the newcomers in stable formal teams** (team instability). As mentioned before, none of the tasks was carried out by the same constellation of developers. Team instability is a known hindering factor for group learning and affects the performance of software developers negatively (Narayanan et al., 2009; Šmite and van Solingen, 2016). Although none of the interviewees mentioned this, we identified the team instability during the quantitative data analysis.

[5] www.stackoverflow.com.

5. A process to strategize and evaluate the onboarding of software developers — RQ2

We packed as a process the findings from our exploratory study reported herein together with the findings and solutions reported in our previous research (Britto et al., 2016c,b,a, 2018). To do so, we used BPM (Holt, 2009). The resulting process provides a systematic way to strategize onboarding undertakings and evaluate onboarding results. The resulting process is presented in Fig. 4.

The process has five activities and a knowledge repository, whose role is to allow for documentation and reuse of onboarding experiences and lessons learned. The knowledge repository can be implemented in different forms, like a wiki. This process was designed to be used within the context of large-scale, globally distributed projects. However, it can be adapted to be used in any other context wherein software developers and teams are onboarded.

The activities, an example of how to use the process, and the results of the static validation are presented in the remainder of this section.

5.1. Define context

Classifying, describing, and documenting the context of an onboarding undertaking is essential to allow for identifying the associated needs, particularities, and challenges (Petersen and Wohlin, 2009). It also provides more systematic reuse of previous onboarding experiences and facilitates knowledge mining (Vegas et al., 2009).

Define context is the first activity of the proposed process. In this activity, an onboarding undertaking context is classified using the extended version (Britto et al., 2016c) of Smite et al.'s GSE taxonomy (Šmite et al., 2014), whose dimensions are presented in Tables 4–6. The taxonomy allows for classification on site (Table 4) and relationship-between-pair-of-sites levels (Tables 5 and 6).

In the context of the proposed process, the taxonomy is to be used following these steps:

1. Identify the source site(s) (the main source(s) of knowledge and where the key decision-makers are located) and the target sites (locations where new software developers/teams will be onboarded). Note that it can be the case that the target and source sites are the same, which happens when new teams will be onboarded in the same location where the main source of knowledge and the decision-makers are located.
2. Classify each involved site using the dimensions described in Table 4.
3. Classify each relationship between a pair of sites using the dimensions described in Tables 5 and 6. Note that if the source and target sites are the same, this step is not necessary.

5.2. Strategize onboarding undertaking

Before starting any onboarding undertaking, it is essential to define the appropriate strategy for the associated context. The combination of tools, practices, recommendations, performance goals, and measurement milestones constitutes an onboarding strategy, which is formalized in an onboarding plan.

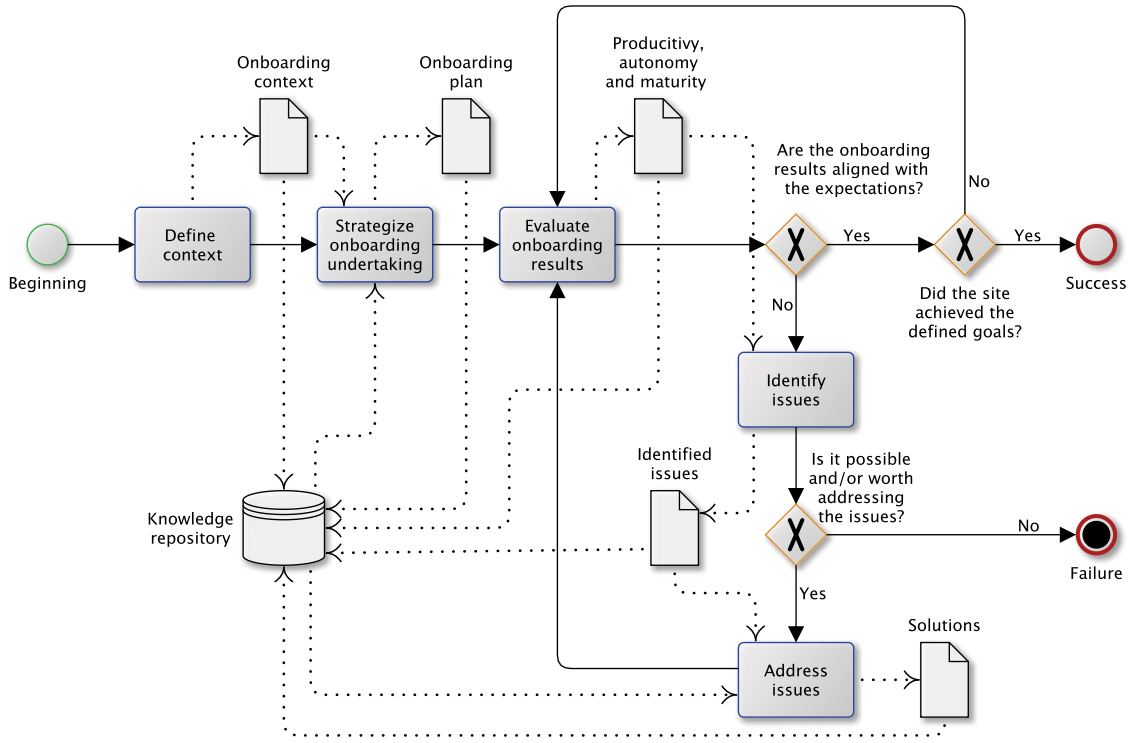

Fig. 4. The proposed process.

Table 4
Dimensions of the GSE taxonomy on site level.

Dimension	Categories	Description
Language distance	No distance, Small, Medium, large	The language distance is measured in terms of the distance between a site's mother tongue and English (language distance index), which is a number that varies from 0 to 1. There is **no distance** when a site's mother tongue is English or there is no need for a *lingua franca*. The distance is **small** when the a site's language distance index is smaller or equal to 0.4. It is considered **medium** when a site's language distance index is greater than 0.4 and smaller than 0.57. Finally, it is considered **large** when it is greater than 0.57 and smaller or equal to 1.
Software process type	Agile, Plan-driven	A site is to be classified as **agile** if its software process is mainly based on agile practices. Otherwise, it is to be classified as **plan-driven**.
Power distance	Small, Large	A site has **large** power distance when its power distance index is greater than 50; otherwise, it is considered **small**.
Uncertainty avoidance	Weak, Strong	A site has **strong** uncertainty avoidance when its uncertainty avoidance index is greater than 63; otherwise, it is considered **weak**.

Table 5
Dimensions of the GSE taxonomy on relationship-between-pair-of-sites level - Part I.

Dimension	Categories	Description
Location	Onshore, Offshore	A sourcing can be delegated to a site in the same country, i.e., onshore, or to a site in another country, i.e., offshore.
Legal entity	Insourcing, Outsourcing	Independently from the location, a sourcing can be transferred to a different branch (site) of the company, i.e., insourcing, or subcontracted to a different legal entity (company), i.e., outsourcing.
Geographical distance	Close, Distant, Near, Far	In onshore projects, the geographical distance is considered: **close** when it is possible to have relatively frequent face-to-face meetings, since no flights are required to go from one site to the other; **distant** when at least one flight is required to have face-to-face meetings, which yields time and cost increases. In offshore projects, the geographical distance is considered: **near** when the required flying time is less than two hours; **far** when the flying time is longer than two hours and staying overnight is usually required.
Temporal distance	Similar, Different, Small, Large	In onshore projects, the temporal distance is considered: **similar** when there is a time difference of one hour or less; **different** when the time difference between two sites is longer than one hour. In offshore projects, the temporal distance is considered: **small** when there is a time distance between sites of four hours or less; **large** when there is a time distance between two sites of more than four hours.

The decision makers must account for relevant tools, practices, methods, and techniques to strategize an onboarding undertaking, such as the ones identified by Britto et al. (2018, 2019):

- Integrate recruitment and onboarding.
- Provide realistic job previews.
- Create an onboarding plan.

Table 6
Dimensions of the GSE taxonomy on relationship-between-pair-of-sites level - Part II.

Dimension	Categories	Description
Software process distance	Equal, Similar, Different	The software processes of two sites are considered **equal** when they use the same workflows, roles, and practices to develop software. They have **similar** processes when they have the same software process type, but the workflows, roles, and practices are not the same. The processes are considered different when there are no commonalities.
Communication model	Low synchronicity, high synchronicity, balanced synchronicity	It is **low** when the communication is mainly based on asynchronous media (e.g., email). It is **high** when communication is mainly based on synchronous media (e.g., instant messaging tools). It is **balanced** when the communication model has both synchronous and asynchronous media, and each media type is used for its most adequate purpose.

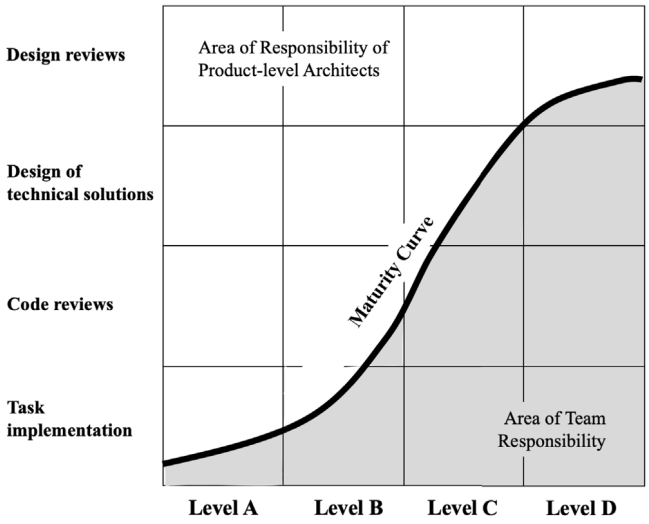

Fig. 5. The authority matrix (Britto et al., 2016a) showing the maturity levels A–D.

- Involve key stakeholders in the recruitment and onboarding process.
- Provide formal orientation for newcomers.
- Provide mentoring for newcomers.
- Evaluate the progress of newcomers.
- Provide feedback for newcomers.
- Take into account cultural differences when developing the learning process.
- Ensure collocated mentoring to support the learning process of offshore teams when there is not enough competence locally.
- Facilitate group learning.
- Use code reviews to support learning.

It is necessary to define performance indicators of interest and baselines to define performance goals and milestones. Based on the exploratory study reported in this paper and Britto et al. (2016a), we set productivity, autonomy, and maturity as performance indicators. Productivity and autonomy are defined in Section 3. Maturity (Britto et al., 2016a) relates to the amount of knowledge associated with a product under development/maintenance, and knowledge about the associated technology stack. The bigger the knowledge, the bigger the maturity.

The performance goals for a developer or team are defined based on the average performance of high-performance developers or teams (baseline), with the addition of some acceptable margin of variation (up and down the average, to be defined by the decision-makers when strategizing an onboarding undertaking). Performance is measured using the measurement approach described in Section 3.

5.3. Evaluate onboarding results

To identify if an onboarding undertaking is progressing as expected, it is necessary to measure the performance of the onboarded developers or teams using the performance indicators described above and compare the results with the defined performance goals. If the onboarding undertaking is not progressing as expected, it may be the case that the company needs to act (e.g., providing more formal training). The metrics associated with each performance indicator are presented in Section 3. Maturity is defined as follows:

- **Maturity** M_{ki} is measured as the maturity of team k or developer k before carrying out task i. It is measured using the authority matrix presented Britto et al. (2016a). The authority matrix is showed in Fig. 5.

The matrix has a maturity curve that shows how the responsibilities of software architects and developers/teams change as they get more mature. In its y-axis, there are the four main activities that developers/teams and architects are involved, while the x-axis has the four defined maturity levels (Britto et al., 2016a):

- In **Level A** — A developer or team on this level has no or very basic knowledge about the code and architecture of the product. They require a lot of mentoring and support from the architects, even when implementing (i.e., coding, testing, and documenting) non-complex technical solutions. Software architects guide the Level-A developers/teams during the implementation and are actively involved, both with reviewing and approving code.
- **Level B** — A developer or team on this level has sufficient knowledge to implement non-complex technical solutions without much guidance. More knowledgeable team members and design leads of Level-B teams can review the code implemented by the team, but software architects are still required to review the majority of the design and code.
- **Level C** — It represents experienced and mature developers or teams with good knowledge of the product architecture, which can implement complex technical solutions that have a significant architectural impact. They are capable of reviewing and approving their code, and software architects are only involved in approval when critical components of the software architecture are affected (a critical component contains the core functionality of the product that executes key operations). More solution design work is also delegated to Level-C teams. Software architects support these Level-C teams/developers by mentoring the design of technical solutions and providing on-demand guidance in the initial stages of implementation.
- **Level D** — It represents very experienced and rather autonomous developers or teams that are capable of implementing complex technical solutions independently, even the ones that affect critical components of the product architecture. The code implemented by them does not need to be approved by software architects. They can also drive the

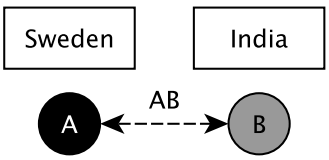

Fig. 6. Sites involved in the case.

design and review of technical solutions. Software architects are involved only in the technical solution design.

5.4. Identify and address issues

The results of the previous activity (evaluate onboarding results) must be compared to the defined performance goals. Deviations can indicate that there are issues in the onboarding undertaking that need to be addressed. Problems may be related to insufficient amounts of formal training and inadequate amounts of mentoring. It may be the case that it is not worthwhile for a company to invest time and money to address an issue. For example, when a newcomer or team composed of newcomers is presenting poor performance after many measurement milestones, decision makers may decide to close down the associated site.

An onboarding undertaking is considered successful when the onboarded developers or teams achieved the defined performance goals. The issues and associated solutions (if any) should be documented in a knowledge repository (e.g., wiki).

5.5. Using the process

To illustrate the use of the process, we used the case investigated in this paper (described in Section 3).

In the investigated case, there are two sites (Fig. 6), one located in Sweden (A — source site), and the other located in India (B — target site). Since there are just two sites in this example, there is just one relationship (AB). The result of employing the taxonomy to describe the context of the example is presented in Table 7 (site level) and 8 (relationship-between-pair-of-sites level).

The onboarding was strategized in the following way:

- The average productivity, autonomy, and maturity level of high-performance teams located in Italy and the USA were used to define the performance goals of the newcomers.
- Training and mentoring were provided on-site in Sweden for the first newcomers so that they could help with the onboarding of other newcomers later on. This initial training in Sweden was followed by on-site mentoring provided by senior software developers. They went from Sweden to India and stayed for four months. The senior developers also provided training for the newcomers onboarded during this period.
- The onboarding was mainly strategized by the decision makers located in Sweden, but the recruitment process was planned and conducted by local personnel in India.
- The performance of the onboarded newcomers was to be continuously evaluated and they were to take the main responsibility for the product after two years.

An evaluation of the newcomers' performance (see Section 4) showed that they were not improving as expected by the decision makers; after two years, they were still on average 3.57 times less productive and 1.67 times less autonomous than the target performance goals (the productivity and autonomy levels of the benchmark teams). Furthermore, no team composed of newcomers in India managed to progress to a mature level (C or D). Thus, it was necessary to extend the time for the Indian site to take over the main responsibility regarding the product. Furthermore, the decision makers decided to prepare some developers from the Indian site to become software architects. The idea was to have people capable of providing on-site mentoring in India. To do so, the selected developers stayed in Sweden for six months and received specific training and mentoring.

Table 7
Classification on the site level for the case.

Dimension	A	B
Language distance	No distance (all employees had to be fully fluent in English)	No distance (all employees had to be fully fluent in English)
Software process type	Agile	Agile
Power distance	Small	Large
Uncertainty avoidance	Strong	Week

Table 8
Classification at the relationship level for the example.

Dimension	AB	Detail
Location	Offshore	The target site is located in a different country.
Legal entity	Insourcing	Both sites are part of the same company (Ericsson)
Geographical distance	Far	The fastest flight between the two sites is longer than two hours.
Temporal distance	Small	There is a time zone difference of 3 h and a half.
Software process distance	Equal	Both sites employ the same software workflows, role and practices.
Communication model	Balanced synchronicity	The communication between the sites is done via different types of media, such as Skype, email and videoconferencing.

6. Static validation of the process

The ideal way to evaluate a process like the one proposed in this paper is by using it in a live project. However, we were not able to do so by the time of our investigation. To mitigate this limitation, we conducted a static validation (Gorschek et al., 2006) of the process by asking experienced Ericsson R&D managers to answer a questionnaire.

The feedback provided by the nine respondents indicates that the process is useful and can help managers to onboard newcomers in an effective way. At the same time, they have shared a few concerns that may make it difficult to use the whole process. The results of the static validation are presented next.

Table 9 contains the results associated with **Q1** (relevance of the different process's components) and shows that all nine respondents of the questionnaire see the significance of all activities of the process. Moreover, **Evaluate onboarding results** and **Store and use knowledge from previous onboarding undertakings** received more *Strongly Agree* answers, which is aligned with the answers provided to the open-ended questions (Q2–Q5).

Regarding **Q2** (the benefits of the process), most respondents believe the main benefit of the process is that it helps to systematize the onboarding of newcomers, making it more repeatable, less dependent on individuals, and ensuring that all required steps are carried out. One of the respondents said: *"It requires you to think first and not only teach what you know, which I see is often the case when we ask teammates to onboard new colleagues without any clear instructions"*. Another respondent had a similar opinion:

Table 9
Results of Q1.

Component	Strongly agree	Agree	Undecided	Disagree	Strongly disagree
Define onboarding context	4	**5**	0	0	0
Strategize onboarding using adequate practices	**6**	3	0	0	0
Evaluate onboarding results	**8**	1	0	0	0
Store and use knowledge from previous onboarding undertakings	**8**	1	0	0	0

"It provides structure into the onboarding process. A checklist is often a good way to secure that all required actions are performed. It also makes it easy to handover between different R&D managers, securing that the process is relatively uniform". The same respondent mentioned that the process may help to identify early on the strengths and weaknesses of newcomers: *"One can get early on a feeling of where the individual has strengths and weaknesses"*.

The fact that the process enables the description of an onboarding undertaking's context was highlighted by one of the respondents as an advantage: *"Context is easy to forget since we directly jump into the technical aspect. With this process, we do not miss to explain what purpose we serve with our technical solutions and how we do it using organization, tools, processes, etc"*.

One of the respondents highlighted that the process helps to document experiences gained with onboarding undertakings, which can help with the onboarding of other newcomers and avoid "reinventing the wheel": *"The benefit is to log and store success stories and what experiences are key to achieve what we aim for"*.

Concerning **Q3** (the drawbacks and limitations of the process), most respondents shared their concern about the administration time required to use the process, especially about the evaluation of onboarding results: *"The main drawback is the risk that it will take too much time in administration, which may mean that the process will not be used in the long run"*. The time and cost associated with maintaining the tools and artifacts related to the process were also mentioned by another manager: *"I think managing the process will take time and money. The tools related to the process will need to be maintained"*.

Another aspect that concerns some of the respondents is that the process can be seemed like a hurdle by some managers, which may lead them to return to ad-hoc ways of onboarding newcomers: *"At the start, the process may be perceived as a hurdle for managers. If this happens, it is easy to slip into old habits, which are ad-hoc"*.

In the opinion of all respondents, there are no steps that should be removed from the process (**Q4**). Moreover, they believe the process is flexible enough and does not require any additional step (**Q5**). However, one of the respondents suggested the creation of a template to facilitate the usage of the process: *"I believe you need to create a template to fill in the onboarding context, like a spreadsheet. You should also provide a spreadsheet to facilitate tracking the status of each developer, like their competence in different areas"*.

7. Discussion

Our results support and extend many prior findings related to the performance of newcomers onboarded in large-scale or just distributed software projects. Our quantitative analysis of performance data confirms the possible dramatic decreases in productivity, as in the case of 80% decrease described by Carmel and Tjia (2005), and the importance of mentoring as a compensating and supporting practice (Fagerholm et al., 2014a,b).

The performance analysis also supports the view that it takes a considerable time for offshore developers to acquire the required knowledge on a distance, similar to Espinosa et al. (2007b) and Šmite and van Solingen (2016). Although we refer to the small performance improvements as unexpected, performance gaps are not surprising.

Although we did not find the true time it takes to acquire the required knowledge, our results indicate that time to minimize performance gaps is certainly longer than three years. Our results are more pessimistic than those of Ebert (2007), who suggests that the learning process may take 12 months. Our results are more aligned with the findings by Mockus and Weiss (2001); they found that remote developers may have up to 3-year learning curves (Mockus and Weiss, 2001).

Given the relatively small performance improvements, it is unreasonable to believe that the remote site will achieve comparable performance in five years, as found by Šmite and Wohlin (2012), and Kommeren and Parviainen (2007). It is likely to take longer than five years, as in the case of Boden et al. (2007) or never achieve full recovery as in the case described by Carmel and Tjia (2005). Therefore, we believe that companies shall plan for large performance gaps and a long time to learning when onboarding developers in large-scale projects with a large amount of legacy code, as described in prior research (Bohner, 2002; Chen and Rajich, 2001), legacy code increases the difficulty to learn.

Jones (1986), based on Van Maanen and Shein work (Van Maanen and Schein, 1979), proposed a model that categorizes onboarding programs into institutionalized (fully formal) and individualized (partially formal). Existing research shows that successful onboarding of newcomers relates to formal onboarding programs (institutionalized onboarding) (Klein and Heuser, 2008; Bauer et al., 2007; Cable and Parsons, 2001). Thus, it is important to strategize onboarding undertakings by systematically developing onboarding programs. The process proposed in this paper supports careful planning, which is needed to: (i) design the stages through which the newcomers are expected to progress (Buchanan, 1974; Feldman, 1976); (ii) to select actors involved with the onboarding (Morrison, 2002; Ashforth, 2001); (iii) to employ tactics and practices for onboarding (Bauer, 2011; Van Maanen and Schein, 1979; Klein and Heuser, 2008); and (iv) to plan the content to be learned (Feldman, 1976; Chao et al., 1994).

Concerning designing the stages through which the newcomers are expected to progress, we found that for highly complex products with large amounts of legacy code, the 4–6 months long co-located mentoring (coaching and support) might be insufficient. We suspect that longer on-site mentoring could help newcomers to progress faster. Increased synchronous interaction between the mentors and the newcomers could be another solution to facilitate the learning. This would also require selecting mentors with excellent communication skills.

We also learned that the company failed to align the training and support activities with the needs and expectations of the remote developers. The learning process employed in the project was mainly based on autonomous learning and expected proactive learners. This might have clashed with the expectations of the Indian newcomers. According to Nicholson and Sahay (2001),

the education system in India is more traditional and emphasizes more discipline than the ones in European countries. Indian developers are more used to a receiving learning approach. Clashing learning approaches could have, therefore, inhibited their progress.

Concerning the content of what shall be learned, the choice of complex tasks might have also negatively impacted the learning experiences. Research related to job satisfaction (Ford and Parnin, 2015) indicates that a perceived lack in programming experience and fear of failure cause frustration and may lead developers to leave their jobs. They are also related to a long period to learn what is necessary to do the required work (Ford and Parnin, 2015). The presence of a considerable amount of legacy code was perceived as an additional complexity driver.

Globally distributed projects often involve multiple sites, which may have different national cultures. Considering that national cultures (Hofstede et al., 2010) influence organizational cultures, it is hard to create the same corporate culture in locations with different national cultures (Nidhra et al., 2013). This may result in fragmented onboarding strategies within the same company and different formality levels regarding onboarding (Britto et al., 2018). Since formal onboarding processes are perceived as more effective (Bauer, 2011), the existence of different formality levels may lead to situations where a company is successful in onboarding developers in a particular location and fails in others.

While it is essential to account for the particularities of involved locations, it is equally important to consider onboarding functions (tools, practices, methods, and techniques Bauer, 2011) that are proved to be effective (Bauer, 2011). Thus, the process put forward in this paper can help companies to systematically account for the particularities of involved locations and effective onboarding practices when strategizing onboarding of developers. This can help to avoid onboarding strategy fragmentation and different effectiveness levels among various sites.

Our study has some implications for research and practice. We have employed a novel approach to the performance evolution of newcomers in large-scale distributed projects. However, we have investigated just one case. Further research is necessary to strengthen the empirical evidence.

Another important aspect is the duration of the investigation. We believe that investigating newcomers for an extended period may enrich our understanding of their performance evolution. For example, it may allow identifying when offshore newcomers will reach and sustain the same level of performance of senior onshore developers.

In our investigation, we analyzed historical data and talked to the newcomers many months after the starting of their onboarding. We believe it may be beneficial to start this type of investigation together with the onboarding of the newcomers. In doing so, it may be possible to reveal things that are not possible to identify with the same research design employed in our investigation.

Finally, we have shown with our investigation that companies in similar contexts should strive for onboarding newcomers more systematically and formally. The process proposed in this paper may support companies in doing so.

8. Threats to validity

The validity threats associated with our investigation are discussed using the categories reliability, internal, construct and external validity described by Runeson et al. (2012). Furthermore, we also discuss conclusion validity (Trochim et al., 2015), since we applied statistical inferences in this investigation.

We minimized **reliability** threat by involving several researchers in the design and execution of our investigation. Furthermore, our observations and findings were verified with the company representatives to avoid false interpretations. We also designed and followed an explicit case study protocol, following the guidelines by Runeson et al. (2012).

However, the approach we used to assess task complexity was highly dependent on the involved software architects. It might therefore be very hard to obtain the same values with other informants. Since we used an iterative process for task complexity assessment (see Section 3.3), we would expect similar results, though.

The same applies to the approach used to adjust productivity and autonomy of distributed product customization tasks. Finally, the data on task effort and mentoring effort was based on time reports provided by the developers and software architects respectively. As investigated by Perry et al. (1994), self time reporting is fairly unreliable. To mitigate this threat, we carried out a sanity check of the collected data, by comparing the reported hours with slot plan documents associated with the investigated newcomers.

Regarding **internal** validity, the facets of performance we investigated are influenced by several confounding factors. We made an attempt to mitigate this threat by interviewing people with different roles and by using existing literature (data triangulation). Regarding the qualitative part of our research, the main internal validity threats are investigator bias and interviewee bias. To mitigate these threats, three researchers were involved with the design of the interview, workshop guides, and static validation questionnaire (investigator triangulation). We mitigated the second threat in the case study by interviewing people with different roles (data triangulation). However, it was not possible to do the same when conducting the process static validation; we only involved managers. the ideal approach would be to involve managers and developers recently onboarded, but there were no suitable developers by the time we evaluated the process.

The main threat to **construct** validity is that we used only one method to measure a construct. To mitigate this threat in the case study, we collected data from different sources (data triangulation). Furthermore, we conducted a sanity check together with the software architects to validate our measures (task complexity and autonomy). To support these sanity checks, we used plots and descriptive statistics to identify inconsistencies (no inconsistencies were found).

Since task complexity was obtained using expert judgment, this is also a potential threat to construct validity (expert bias). This threat was mitigated by involving several experts with many years of experience in the product under investigation and employing an iterative assessment approach (based on planning poker). Existing literature shows that group discussions lead to better effort estimates than expert judgment (Moløkken-Østvold and Jørgensen, 2004).

Regarding the static validation of the proposed process, we piloted the questionnaire with the help of a senior R&D manager. To mitigate expert bias, we asked different managers to answer the questionnaire (nine). However, the fact that no developer answered the questionnaire is a limitation in itself and a threat to the construct validity of the static validation.

Since we employed the case study method, our findings are strongly bound by the context of our study (**external** validity). In addition, the investigated case involved only one product in one company. To mitigate this threat, we made an attempt to describe the context of our study in as much detail as possible. This makes it easier to relate the present case to similar cases.

It is hard to claim that the proposed process is meaningful or useful without evaluating it in a living project, which we

could not do during the course of our investigation. Instead, we conducted a static validation to get the opinion of practitioners about the proposed process. However, it is hard to generalize the results of the conducted static validation due to the small number of respondents (nine) and the fact that all respondents were affiliated to the same company. However, the results are important as a sanity check of the relevance of the proposed process.

The main threats to **conclusion** validity are the low reliability of measures (the amount of noise related to a measure) and low statistical power. To mitigate the first threat, we conducted a sanity check of the collected data, as mentioned above. Regarding statistical power, we were limited to data covering product customization tasks carried out by the newcomers during two years.

9. Conclusions and future work

This paper presents the results of a case study conducted in Ericsson, which aimed at investigating the performance evolution of offshore newcomers onboarded in a large-scale globally distributed project and how it relates to the employed onboarding strategy. The overall conclusion is that some aspects of the used onboarding strategy seem to be associated with the observed performance evolution, which was lower than expected by the company.

We identified the following aspects in the onboarding strategy employed in the investigated case that relate to the observed performance evolution: (i) the distance to mentors; (ii) the formal training approach used, which did not fit the socio-cultural background of the newcomers; (iii) the allocation of large and distributed tasks in the early stages of the onboarding process; and (iv) team instability.

The results of this paper indicate that onboarding in globally distributed projects that involve a considerable amount of complex legacy code must be planned well ahead. It might take a long time until newcomers acquire all the knowledge required to perform as well as experienced teams. Organizations must be prepared to provide extended periods of mentoring by expensive and potentially scarce resources (e.g., software architects).

In our investigation, we covered approximately two years of data. Including an extended period might provide further insights into the performance evolution of newcomers. Therefore, we plan to continue collecting data about this case.

In this paper, we also put forward a process to strategize and evaluate the onboarding of newcomers in globally distributed large-scale projects. The results of the conducted static validation indicate that the process can help companies to strategize and evaluate the results of onboarding undertakings in a more formal, systematic, and repeatable way. However, the proposed process has not been used to plan any onboarding undertaking from scratch. Rather, some of its activities were used in an on-going undertaking (the case investigated in this paper) and not in a systematic way. We believe it is worthwhile to investigate how effective the process is to strategize and evaluate the onboarding of newcomers in real, globally distributed large-scale projects.

CRediT authorship contribution statement

Ricardo Britto: Conceptualization, Methodology, Investigation, Writing - original draft. **Darja Smite:** Supervision, Validation, Writing - review & editing. **Lars-Ola Damm:** Investigation. **Jürgen Börstler:** Supervision, Writing - review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This research was funded by the Swedish Knowledge Foundation under the KK-Hög grant 2009/0249 and is a part of the TEDD research project. The authors are very thankful to Ericsson and all the company employees involved in and being sincerely interested in our research.

References

Ashforth, B.E., 2001. Role Transitions in Organizational Life: An Identity-Based Perspective. Lawrence Erlbaum Associates, Mahwah, NJ, p. 353.
Bauer, T.N., 2011. Onboarding New Employees: Maximizing Success. Tech. rep., SHRM Foundation, pp. 1–54.
Bauer, T.N., Bodner, T., Erdogan, B., Truxillo, D.M., Tucker, J.S., 2007. Newcomer adjustment during organizational socialization: A meta-analytic review of antecedents, outcomes and methods. J. Appl. Psychol. 92, 707–721.
Bauer, T.N., Erdogan, B., 2011. Organizational socialization: The effective onboarding of new employees. In: APA Handbook of Industrial and Organizational Psychology, Vol 3: Maintaining, Expanding, and Contracting the Organization. pp. 51–64.
Bauer, T.N., Green, S.G., 1994. Effect of newcomer involvement in work-related activities: a longitudinal study of socialization. J. Appl. Psychol. 79 (2), 211–223.
Bauer, T.N., Morrison, E.W., Callister, R.R., 1998. Organizational socialization: A review and directions for future research. Res. Pers. Hum. Resour. Manage. 16 (January), 149–214.
Boden, A., Nett, B., Wulf, V., 2007. Coordination practices in distributed software development of small enterprises. In: International Conference on Global Software Engineering (ICGSE 2007). pp. 235–246.
Bohner, S.A., 2002. Software change impacts-an evolving perspective. In: Software Maintenance, 2002. Proceedings. International Conference on. pp. 263–272.
Bondi, A.B., Ros, J.P., 2009. Experience with training a remotely located performance test team in a quasi-agile global environment. In: Proceedings of the 2009 Fourth IEEE International Conference on Global Software Engineering - ICGSE'09. IEEE Computer Society, Washington, DC, USA, pp. 254–261.
Britto, R., Cruzes, D.S., Smite, D., Sablis, A., 2018. Onboarding software developers and teams in three globally distributed legacy projects: A multi-case study. J. Softw.: Evol. Process 30 (4), e1921.
Britto, R., Šmite, D., Damm, L., 2016a. Software architects in large-scale distributed projects: An ericsson case. IEEE Softw. 33 (6), 48–55.
Britto, R., Smite, D., Damm, L., Börstler, J., 2019. Performance evolution of newcomers in large-scale distributed software projects: An industrial case study. In: 2019 ACM/IEEE 14th International Conference on Global Software Engineering (ICGSE). pp. 1–11.
Britto, R., Šmite, D., Damm, L., 2016b. Experiences from measuring learning and performance in large-scale distributed software development. In: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement. In: ESEM '16, ACM, New York, NY, USA, pp. 17:1–17:6.
Britto, R., Wohlin, C., Mendes, E., 2016c. An extended global software engineering taxonomy. J. Softw. Eng. Res. Dev. 4 (1), 1–24.
Buchanan, B., 1974. Building organizational commitment: The socialization of managers in work organizations. Adm. Sci. Q. 19 (4), 533–546.
Burchell, J., Vargas, M., 2016. The Hitchhiker's Guide to ggplot2 in R. Leanpub.
Cable, D.M., Parsons, C.K., 2001. Socialization tactics and person-organization fit. Pers. Psychol. 54 (1), 1–23.
Carmel, E., Tjia, P., 2005. Offshoring Information Technology: Sourcing and Outsourcing to a Global Workforce. Cambridge University Press.
Cataldo, M., Nambiar, S., 2009. On the relationship between process maturity and geographic distribution: An empirical analysis of their impact on software quality. In: Proceedings of the the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering. In: ESEC/FSE '09, ACM, New York, NY, USA, pp. 101–110.
Chao, G.T., O'Leary-Kelly, A.M., Wolf, H.J., Klein, S., Gardner, P.D., 1994. Organizational socialization: Its contents and consequences. J. Appl. Psychol. 79 (5), 730–743.
Chen, K., Rajich, V., 2001. RIPPLES: tool for change in legacy software. In: Software Maintenance, 2001. Proceedings. IEEE International Conference on. pp. 230–239.

Conchúir, E., Ågerfalk, P.J., Holmström, H., Fitzgerald, B., 2009. Global software development: Where are the benefits? Commun. ACM 52 (8), 127–131.
Dikert, K., Paasivaara, M., Lassenius, C., 2016. Challenges and success factors for large-scale agile transformations: a systematic literature review. J. Syst. Softw. 119, 87–108.
Ebert, C., 2007. Optimizing supplier management in global software engineering. In: International Conference on Global Software Engineering (ICGSE 2007). pp. 177–185.
Eldh, S., Brandt, J., Street, M., Hansson, H., Punnekkat, S., 2010. Towards fully automated test management for large complex systems. In: 2010 Third International Conference on Software Testing, Verification and Validation. pp. 412–420.
Espinosa, J.A., Nan, N., Carmel, E., 2007a. Do gradations of time zone separation make a difference in performance? a first laboratory study. In: Second IEEE International Conference on Global Software Engineering - ICGSE'07. pp. 12–22.
Espinosa, J.A., Slaughter, S.A., Kraut, R.E., Herbsleb, J.D., 2007b. Familiarity, complexity, and team performance in geographically distributed software development. Organ. Sci. 18 (4), 613–630.
Fagerholm, F., Guinea, A.S., Münch, J., Borenstein, J., 2014a. The role of mentoring and project characteristics for onboarding in open source software projects. In: Proceedings of the ACM-IEEE 8th International Symposium on Software Engineeering and Measurement - ESEM'14. pp. 1–10.
Fagerholm, F., Sanchez-Guinea, A., Borenstein, J., Münch, J., 2014b. Onboarding in open source projects. IEEE Softw. 31 (6), 54–61.
Feldman, D.C., 1976. A contingency theory of socialization. Adm. Sci. Q. 21 (3), 433–452.
Ford, D., Parnin, C., 2015. Exploring causes of frustration for software developers. In: 2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering. pp. 115–116.
Gorschek, T., Garre, P., Larsson, S., Wohlin, C., 2006. A model for technology transfer in practice. IEEE Softw. 23 (6), 88–95.
Green, S.B., 1991. How many subjects does it take to do a regression analysis. Multivariate Behav. Res. 6 (3), 499–510.
Herbsleb, J.D., Mockus, A., 2003. An empirical study of speed and communication in globally distributed software development. IEEE Trans. Softw. Eng. 29 (6), 481–494.
Herbsleb, J., Moitra, D., 2001. Global software development. IEEE Softw. 18 (2), 16–20.
Hofstede, G., Hofstede, G.J., Minkov, M., 2010. Cultures and Organizations: Software of the Mind, third ed. McGraw-Hill.
Holt, J., 2009. A Pragmatic Guide to Business Process Modelling. In: British Comp Society Series, British Computer Society.
Jabangwe, R., Petersen, K., Šmite, D., 2013. Visualization of defect inflow and resolution cycles: Before, during and after transfer. In: 2013 20th Asia-Pacific Software Engineering Conference (APSEC). pp. 289–298.
Jabangwe, R., Šmite, D., 2012. An exploratory study of software evolution and quality: Before, during and after a transfer. In: 2012 IEEE Seventh International Conference on Global Software Engineering. pp. 41–50.
Johnson, M., Senges, M., 2010. Learning to be a programmer in a complex organization: A case study on practice-based learning during the onboarding process at google. J. Workplace Learn. 22 (3), 180–194.
Jones, G.R., 1986. Socialization tactics, self-efficacy, and newcomers' adjustments to organizations.. Acad. Manage. J. 29 (2), 262–279.
Klein, H.J., Fan, J., Preacher, K.J., 2006. The effects of early socialization experiences on content mastery and outcomes: A mediational approach. J. Vocat. Behav. 68, 96–115.
Klein, H.J., Heuser, A., 2008. The learning of socialization content: A framework for researching orientating practices. Res. Pers. Hum. Resour. Manage. 27, 278–336.
Klein, H.J., Polin, B., Leigh-Sutton, K., 2015. Specific onboarding practices for the socialization of new employees. Int. J. Sel. Assess. 23 (3), 263–283.
Klein, H.J., Weaver, N.A., 2000. The effectiveness of an organizational-level orientation training program in the socialization of new hires. Pers. Psychol. 53, 47–66.
Kommeren, R., Parviainen, P., 2007. Philips experiences in global distributed software development. Empir. Softw. Eng. 12 (6), 647–660.
Labuschagne, A., Holmes, R., 2015. Do onboarding programs work? In: Proceedings of the IEEE/ACM 12th Working Conference on Mining Software Repositories. pp. 381–385.
Maier, G., Brunstein, J.C., 2001. The role of personal work goals in newcomers' job satisfaction and organizational commitment: A longitudinal analysis. J. Appl. Psychol. 86 (5), 1034–1042.
Maxwell, S.E., 2000. Sample size and multiple regression analysis. Psychol. Methods 5 (4), 434–458.
Meyer, J.P., Allen, N.., 1988. Links between work experiences and organizational commitment during the first year of employment: A longitudinal analysis. J. Occup. Psychol. 61 (3), 195–209.
Mockus, A., Weiss, D.M., 2001. Globalization by chunking: a quantitative approach. IEEE Softw. 18 (2), 30–37.
Moløkken-Østvold, K., Jørgensen, M., 2004. Group processes in software effort estimation. Empir. Softw. Eng. 9 (4), 315–334.
Morrison, E.M., 2002. Newcomers' relationships: The role of social network ties during socialization. Acad. Manage. J. 45, 1149–1160.
Narayanan, S., Balasubramanian, S., Swaminathan, J.M., 2009. A matter of balance: Specialization, task variety, and individual learning in a software maintenance environment. Manage. Sci. 55 (11), 1861–1876.
Neumann, G., Harman, M., Poulding, S., 2015. Transformed vargha-delaney effect size. In: Barros, M., Labiche, Y. (Eds.), 7th International Symposium on Search-Based Software Engineering. In: SSBSE'15, pp. 318–324.
Nguyen-Duc, A., Cruzes, D.S., Conradi, R., 2014. The impact of global dispersion on coordination, team performance and software quality - A systematic literature review. Inf. Softw. Technol. 57, 277–294.
Nicholson, B., Sahay, S., 2001. Some political and cultural issues in the globalisation of software development: Case experience from britain and India. Inf. Organ. 11 (1), 25–43.
Nidhra, S., Yanamadala, M., Afzal, W., Torkar, R., 2013. Knowledge transfer challenges and mitigation strategies in global software development-A systematic literature review and industrial validation. Int. J. Inf. Manage. 33 (2), 333–355.
Ostroff, C., Kozlowski, S.W.J., 1993. The role of mentoring in the information gathering processes of newcomers during early organizational socialization. J. Vocat. Behav. 42, 170–183.
Perry, D.E., Staudenmayer, N., Votta, L.G., 1994. People, organizations, and process improvement. IEEE Softw. 11 (4), 36–45.
Petersen, K., Wohlin, C., 2009. Context in industrial software engineering research. In: Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement. In: ESEM'09, IEEE Computer Society, Washington, DC, USA, pp. 401–404.
Ramasubbu, N., Cataldo, M., Balan, R.K., Herbsleb, J.D., 2011. Configuring global software teams: A multi-company analysis of project productivity, quality, and profits. In: Proceedings of the 33rd International Conference on Software Engineering - ICSE'11. pp. 261–270.
Robson, C., McCartan, K., 2016. Real World Research, fourth ed. Wiley, p. 560.
Runeson, P., Höst, M., Rainer, A., Regnell, B., 2012. Case Study Research in Software Engineering: Guidelines and Examples. John Wiley & Sons, ISBN: 978-1118104354.
Sharma, G.G., Stol, K.-J., 2020. Exploring onboarding success, organizational fit, and turnover intention of software professionals. J. Syst. Softw. 159, 110442.
Šmite, D., van Solingen, R., 2016. What's the true hourly cost of offshoring? IEEE Softw. 33 (5), 60–70.
Šmite, D., Wohlin, C., 2011. Strategies facilitating software product transfers. IEEE Softw. 28 (5), 60–66.
Šmite, D., Wohlin, C., Galviņa, Z., Prikladnicki, R., 2014. An empirically based terminology and taxonomy for global software engineering. Empir. Softw. Eng. 19 (1), 105–153.
Steinmacher, I., Gerosa, M.A., 2014. Choosing an appropriate task to start with in open source software communities: A hard task. In: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 8658 LNCS, pp. 349–356.
Steinmacher, I., Silva, M.A.G., Gerosa, M.A., Redmiles, D.F., 2015. A systematic literature review on the barriers faced by newcomers to open source software projects. Inf. Softw. Technol. 59 (March), 67–85.
Trochim, W., Donnelly, J.P., Arora, K., 2015. Research Methods: The Essential Knowledge Base. Cengage.
Van Maanen, J., Schein, E.H., 1979. Toward a theory of organizational socialization. Res. Organ. Behav. 1, 209–269.
Vegas, S., Juristo, N., Basili, V.R., 2009. Maturing software engineering knowledge through classifications: A case study on unit testing techniques. Softw. Eng. IEEE Trans. 35 (4), 551–565.
Vierhauser, M., Rabiser, R., Grünbacher, P., 2014. A case study on testing, commissioning, and operation of very-large-scale software systems. In: Companion to the Proceedings of the 36th International Conference on Software Engineering. In: ICSE Companion. New York, NY, USA, pp. 125–134.
von Rosing, M., von Scheel, H., Scheer, A., 2014. The Complete Business Process Handbook: Body of Knowledge from Process Modeling to BPM. Elsevier Science.
Šmite, D., Calefato, F., Wohlin, C., 2015. Cost savings in global software engineering: Where's the evidence? IEEE Softw. 32 (4), 26–32.

Šmite, D., Wohlin, C., 2012. Lessons learned from transferring software products to India. J. Softw. Evol. Proc. 24, 605–623.

Ricardo Britto is a data-driven change leader at Ericsson and an adjunct lecturer of the Department of Software Engineering at Blekinge Institute of Technology. Britto received a Ph.D. in Software Engineering from the Blekinge Institute of Technology (BTH), Sweden. Between 2009 and 2013, Britto worked as researcher and project manager at Federal University of Piaui-Brazil. His research interests include large-scale agile software development, global software engineering, search-based software engineering and software process improvement. Contact him at ricardo.britto@ericsson.com, ricardo.britto@bth.se.

Darja Šmite is a professor of the Department of Software Engineering at Blekinge Institute of Technology and a part-time research scientist at SINTEF ICT. She has dedicatedly focused on understanding the impact of globalization and offshoring in software companies for more than a decade. Her other research interests include large-scale agile software development, and software process improvement. Šmite received a Ph.D. in Computer Science from the University of Latvia. Contact her at darja.smite@bth.se.

Lars-Ola Damm works at Ericsson and over the years has had several positions in different organizations within the company, including technical roles and driving global improvement initiatives. Since 2011, he has been a manager of software development teams working in large distributed projects. His main research interest is software process improvement. Damm received a Ph.D. in Software Engineering from the Blekinge Institute of Technology (BTH), Sweden. Contact him at lars-ola.damm@ericsson.com.

Jürgen Börstler is a professor at and head of the Department of Software Engineering at Blekinge Institute of Technology. Previously, he has been a professor of Computer Science at Umeå University, Sweden. He received a Ph.D. in Computer Science from Aachen University of Technology (RWTH), Germany. His main research interests are in requirements engineering, object-oriented methods, Software process improvement, software measurement, Software comprehension, and computer science education. He also is a founding member of the Scandinavian Pedagogy of Programming Network.